FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Teaching Reinforcement Learning in ROS 1/2 using a Mobile Robot and its Simulation

Vítor Emanuel Moreira Ventuzelos

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Armando Jorge Miranda de Sousa

Second Supervisor: Gonçalo da Mota Laranjeira Torres Leão, MSc.

July 25, 2022

Teaching Reinforcement Learning in ROS 1/2 using a Mobile Robot and its Simulation

Vítor Emanuel Moreira Ventuzelos

Mestrado em Engenharia Informática e Computação

July 25, 2022

Abstract

Robot Operating System (ROS) is the most prominent architecture to program and control robots and is widely used in both academic and industrial settings. As such, it is taught in advanced courses from various higher education institutions, and multiple tutorials and guides have been written and published, both online and in paper. However, ROS is structured differently from common programming architectures, which causes some attrition in the learning process, and most introductions rely solely on virtual simulators, in part due to the high cost of most robotic platforms, which leaves the user with little experience on the operation and maintenance of physical equipment and a skewed perspective of what ROS truly does.

This project aims to provide a comprehensive curriculum to be used in Intelligent Robotics courses, with a friendlier introduction to ROS, paired with the development of a simple, low-cost, and reliable robot to serve as a teaching platform, allowing students to acquire experience with both simulators and hardware while also making the learning experience more approachable and enjoyable.

Throughout this work, the foundation for an educational package for teaching Intelligent Robotics is put together, comprising three elements: a lightweight two-dimensional simulation based on the Flatland simulator, a small ground robot with a 3D printed chassis, multiple virtual sensors based on a single wide lens camera and a combination of a Raspberry Pi 4 single board computer and an Arduino microcontroller as its brains, and a set of introductory lessons to ROS1, ROS2 and Reinforcement Learning in robotics. The lessons include sample code in both C++ and Python, in order to be accessible to a wider audience. The simulation and the real robot were tailored together so that code runs similarly on either, without the need for modifications. Special focus was placed on providing a framework to simplify the introduction to Reinforcement Learning.

The lessons developed were tested with users with various degrees of experience in robotics in order to evaluate their pertinence and encouraging results were obtained, with all three lessons satisfying the proposed learning outcomes. The remaining components of the package presented positive results as well, as the virtual sensors developed for the camera proved effective in detecting obstacles and the agents trained through the adapted RL framework providing favorable results in both simulated and real environments.

Keywords: ROS, Intelligent Robotics, Educational Robots

Resumo

Robot Operating System (ROS) é a arquitetura mais proeminente para programar e controlar robôs, e é amplamente usado tanto em ambientes académicos como industriais. Como tal, é ensinado em unidades curriculares avançadas de várias instituições do ensino superior, e vários tutoriais e guias foram escritos e publicados, tanto online como em papel. No entanto, ROS é estruturado de forma diferente das arquiteturas de programação mais comuns, o que causa algum atrito no processo de aprendizagem, e a maioria das introduções apoiam-se unicamente em simuladores virtuais, em parte devido ao alto custo da maioria das plataformas robóticas, o que deixa o utilizador com pouca experiência na operação e manutenção de equipamento físico e uma perspetiva distorcida do que ROS realmente faz.

Este projeto pretende providenciar um currículo abrangente com uma introdução mais amigável a ROS para unidades curriculares de Robótica Inteligente, em conjunto com o desenvolvimento de um robô simples, fiável e de baixo custo para servir de plataforma de ensino, permitindo aos estudantes adquirir experiência tanto com simuladores como com hardware e ao mesmo tempo tornando a aprendizagem mais acessível e agradável.

Ao longo deste trabalho, as fundações para um pacote educacional para o ensino de Robótica Inteligente são construídas, contendo três elementos: uma simulação bidimensional leve baseada no simulador Flatland, um pequeno robô terrestre com chassis imprimido em 3D, vários sensores virtuais baseados numa única camara de lente ampla e uma combinação de um computador de placa única Raspberry Pi 4 e um microcontrolador Arduino como cérebro, e um conjunto de lições introdutórias a ROS1, ROS2 e Aprendizagem por Reforço em robótica. As lições incluem código de exemplo tanto em C++ como em Python, para as tornar accessíveis a uma maior audiência. A simulação e o robô real foram modelados em conjunto para que código corra de forma semelhante em ambos, sem necessidade de modificações. Foco especial foi colocado em providenciar uma estrutura para simplificar a introdução a aprendizagem por reforço.

As lições desenvolvidas foram testadas com utilizadores possuíndo vários níveis de experiência em robótica para avaliar a sua pertinência e resultados encorajadores foram obtidos, sendo que todas as três lições satisfizeram os resultados de aprendizagem propostos. Os restantes componentes do pacote também apresentaram resultados positivos, tendo os sensores virtuais desenvolvidos para a câmara provado ser eficazes a detetar obstáculos e os agentes treinados através da estrutura adaptada de aprendizagem por reforço providenciado resultados favoravéis tanto em ambientes simulados como reais.

Keywords: ROS, Robótica Inteligente, Robôs Educacionais

Acknowledgements

This chapter is dedicated to giving thanks those who accompanied and supported me throughout this work, without whom it would never have been possible.

To my supervisors, for guiding me and providing me with resources to develop the project.

To my colleague Alexandre Miranda, whose help was invaluable in testing key components of the project.

To my family, for providing me with the means to come this far and encouraging me to keep going even further.

And finally, to my friends, for supporting me the whole time, and for preserving my sanity.

Vítor Ventuzelos

"You've got to make a statement,
you've got to look inside of yourself and say:
'What am I willing to put up with today?"'

Arin Hanson

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 1
- 1.3 Objectives . . . 2
- 1.4 Research Questions . . . 2
- 1.5 Document Structure . . . 3

2 Programming and Working with Robots **4**
- 2.1 Robot Operating System . . . 4
  - 2.1.1 History . . . 4
  - 2.1.2 Distinction Between ROS1 and ROS2 . . . 4
  - 2.1.3 Learning ROS . . . 5
- 2.2 Real Robots . . . 5
  - 2.2.1 Simulation Engines . . . 5
  - 2.2.2 AI Applied to Robotics . . . 5
  - 2.2.3 Real World Applications . . . 6
- 2.3 Conclusion . . . 6

3 Robotics Education **7**
- 3.1 Educational Robot Platforms . . . 7
  - 3.1.1 Simulated Platforms . . . 7
  - 3.1.2 Physical Robot Platforms . . . 8
- 3.2 How and When Robotics is Taught . . . 10
  - 3.2.1 Pre-University Level . . . 10
  - 3.2.2 Higher Education Level . . . 10
  - 3.2.3 ROS1 Courses and Introductions . . . 10
  - 3.2.4 ROS2-Focused Courses . . . 11
- 3.3 Conclusion . . . 11

4 Educational Package Design **12**
- 4.1 Introduction . . . 12
- 4.2 Requirements . . . 13
  - 4.2.1 Simulation Engine . . . 13
  - 4.2.2 Physical Platform . . . 13
  - 4.2.3 Lesson Plan . . . 14
- 4.3 Simulator Selection . . . 14
- 4.4 Architecture . . . 15
  - 4.4.1 Use Case Example . . . 16

5 Educational Package Development **18**
5.1 Simulation Development . . . 18
5.1.1 Simulation Assets . . . 18
5.1.2 Introductory Tutorials . . . 20
5.2 Physical Platform Development . . . 21
5.2.1 Robot Setup . . . 21
5.2.2 Virtual Sensors . . . 22

6 Reinforcement Learning Lesson **26**
6.1 Framework Adaptation . . . 26
6.2 Learning Experiments . . . 27
6.2.1 Original State Collection and Original Reward Function . . . 28
6.2.2 Original State Collection and Modified Reward Function . . . 28
6.2.3 Modified State Collection and Modified Reward Function . . . 29
6.2.4 Results . . . 29
6.3 Agent Application on Real Robot . . . 30
6.3.1 Direct Port Experiment . . . 30
6.3.2 Improvements and Results . . . 30
6.4 Lesson Development and Analysis . . . 31
6.5 Summary . . . 32

7 Conclusions and Future Work **35**
7.1 Conclusions . . . 35
7.2 Future Work . . . 36
7.2.1 Simulation . . . 36
7.2.2 Physical Robot Platform . . . 36
7.2.3 Reinforcement Learning . . . 36
7.2.4 Lesson Plan . . . 37

References **38**

A Introductory Tutorials **42**
A.1 ROS1 Introduction . . . 42
A.2 ROS2 Introduction . . . 50
A.3 RL Introduction . . . 59

B Feedback Forms **64**

C Form Results **76**
C.1 ROS1 Tutorial . . . 76
C.2 ROS2 Tutorial . . . 77
C.3 RL Tutorial . . . 78

D Detailed Virtual Sensor Images **80**
D.1 Simulated LiDAR . . . 80
D.2 Object Detection . . . 80
D.3 Proximity Sensor . . . 80
D.3.1 Calibration . . . 80

E Reinforcement Learning Results **89**

List of Figures

3.1 Physical Educational Robot Platforms 8

4.1 Abstract Architecture 15
4.2 Environment Component Architectures 16
4.3 Agent Component Architecture 17
4.4 Sequence Diagram 17

5.1 Robot Representations 19
5.2 Simulated Sensors 19
5.3 ROS1 Learning Outcome Results 22
5.4 ROS2 Learning Outcome Results 23
5.5 SERP robot 24
5.6 Virtual laser sensor distance equations 25
5.7 Sensor comparison in different lighting conditions 25

6.1 Default Reinforcement Learning Model Parameters 28
6.2 Reinforcement Learning Agent Trajectories 33
6.3 RL Learning Outcome Results 34

D.1 Simulated LiDAR in well lit conditions 80
D.2 Simulated LiDAR in low light conditions 81
D.3 Object Detection in well lit conditions 81
D.4 Object Detection in low light conditions 82
D.5 Proximity Sensor in well lit conditions 82
D.6 Proximity Sensor in low light conditions 83
D.7 Ray at $-\frac{1}{4}\pi$ 83
D.8 Ray at $-\frac{1}{8}\pi$ 84
D.9 Ray at 0 84
D.10 Ray at $\frac{1}{8}\pi$ 85
D.11 Ray at $\frac{1}{4}\pi$ 85
D.12 Ray at $-\frac{3}{4}\pi$ 86
D.13 Ray at $-\frac{7}{8}\pi$ 86
D.14 Ray at π 87
D.15 Ray at $\frac{7}{8}\pi$ 87
D.16 Ray at $\frac{3}{4}\pi$ 88

E.1 Test with Configuration 1, Complete Environment 89
E.2 Test with Configuration 1, Close-up 90
E.3 Test 1 with Configuration 2, Complete Environment 91

E.4 Test 1 with Configuration 2, Close-up 92
E.5 Test 2 with Configuration 2, Complete Environment 93
E.6 Test 2 with Configuration 2, Close-up 94
E.7 Test 1 with Configuration 3, Complete Environment 95
E.8 Test 1 with Configuration 3, Close-up 96
E.9 Test 2 with Configuration 3, Complete Environment 97
E.10 Test 2 with Configuration 3, Close-up 98

List of Tables

4.1 Simulator Comparison 15

5.1 ROS1 Learning Outcome Results 21
5.2 ROS2 Learning Outcome Results 21
5.3 Virtual laser sensor distance calculation 24

6.1 Reward function parameter values 29
6.2 RL Learning Outcome Results 32

C.1 ROS1 Tutorial feedback results 77
C.2 ROS2 Tutorial feedback results 78
C.3 RL Tutorial feedback results 79

Abbreviations

AI	Artificial Intelligence
OS	Operating System
ROS	Robot Operating System
RT	Real Time
SBC	Single Board Computer
SDK	Software Development Kit
RL	Reinforcement Learning
LiDAR	Light Detection And Ranging

Chapter 1

Introduction

1.1 Context

Throughout the last two decades, the fields of Robotics and Artificial Intelligence (AI) have become exponentially more prominent. These two fields are deeply interconnected, and examples can be found everywhere, from children's toys like Sony's Aibo and Lego's Mindstorms, to home appliances like roombas and industrial machines like robotic arms used in assembly lines. As robotics progressed, multiple forms of programming robots were developed, but Robot Operating System (ROS), originally conceived at the Stanford Artificial Intelligence Lab in 2007 and later picked up by Google [26], has become the standard for robot development, for the most part. Currently, this framework has been undergoing a transition to a new design, ROS2, with compatibility and quality of life improvements, which is already receiving support from related institutions [32].

With the growing use of robotics, the education about robotics follows suit, with the amount of literature on Robotics Education published each year increasing [22]. Nonetheless, a considerable portion of this literature uses proprietary software and hardware, and architectures that are incompatible with ROS. Paired with how recent ROS2 is, a void still exists for education materials based on this framework, which is slowly being filled by paid online courses and video tutorials.

1.2 Motivation

Despite there being numerous robotic platforms that can be used for teaching robotics, most of them, if not all, suffer from recurring issues that make their use in education difficult. Notable among them are high costs and complexity, low replicability, and often closed source software, and a lack of a supporting simulator. In addition, many of these platforms have little to no support for ROS, requiring the student to learn an entirely different framework or development kit. Due to this, the need for a robotic kit that fits all the needs of an introductory ROS course surfaced.

Although robotics courses are lectured in multiple institutions, these are often placed on Master's and Doctoral programs. The reasoning for this choice relates to the need for students to already be familiar with multiple programming languages, capable of using sensors that provide only raw data such as cameras and sonars, and well versed in the development of algorithms and AI. However, initiatives for using robotics in pre-university courses, from primary to high school, have been surfacing in recent years, proving that this field does not need to be relegated to advanced university programs. With this in mind, the opportunity to develop a ROS based Intelligent Robotics course arose.

1.3 Objectives

This project aims to provide two elements:

- A robot platform that is open-source, low-cost and can be replicated by an education institution to match the needs of a class of students. This platform is equipped with a modular ROS-based framework that allows programming with both ROS1 and ROS2. It can be used by students with varying levels of knowledge, and is accompanied by a representation of the robot in a lightweight simulator for AI training and testing of algorithms, so that both are interchangeable and developed software can effortlessly be ported between them.
- A set of exercises and examples to enable teaching of Intelligent Robotics in both ROS1 and ROS2 as a university course for students of diverse backgrounds, including Electronics Engineering, Software Engineering, Computer Science as well as International students, and with different levels of knowledge, allowing it to be placed at most points in academic programs.

Additionally, feedback from current teachers and students of related courses will be used to validate the developed solution.

1.4 Research Questions

This work poses the following research questions:

RQ1 - *Is a low-cost, replicable and open-source robot platform for education feasible?* Although there are various platforms available, they all present shortcomings that hinder teaching.

RQ2 - *Can ROS-based robotics be taught at an earlier stage in university programs?* Currently robotics are often placed in MSc and PhD programs, with younger students having little to no contact with the field.

RQ3 - *How can Artificial Intelligence applied to robotics be taught using ROS?* Robotics and AI are often taught as separate subjects, and even when taught together, ROS isn't always involved, as purpose-built simulations are used.

1.5 Document Structure

The following two chapters present a literature review on the topic of this work. Chapter 2 focuses on the state of the art of intelligent robotics, as well as current standards robot development, with special attention to the ROS Software Development Kit (SDK). Chapter 3 focuses on robotics education, examining what is used to teach robotics at each level of education, ranging from primary school to PhD programs.

Chapters 4 through 6 present the solution developed throughout this project. Chapter 4 describes the proposed educational package, along with its components and interactions between them. Chapter 5 details the development of the environment elements of the package, starting with the simulation and the introductory lessons based on it and following up with the real robot and its components. Chapter 6 focuses on the development of a Reinforcement Learning (RL) model designed to train an agent which can serve as the controller for a simulated or real robot, along with a lesson meant to introduce students to AI in robotics.

Lastly, Chapter 7 summarizes the contributions of this project and the conclusions derived from the results obtained, and suggests future lines of work to improve and expand the educational package.

Chapter 2

Programming and Working with Robots

2.1 Robot Operating System

ROS is a SDK that facilitates the creation of distributed systems, by use of isolated nodes which can communicate with each other via messages published and subscribed through topics and by use of services. This architecture allows for compartmentalization of functionalities, and is compatible with multiple programming languages, notable among them C++ and Python. Additionally, the independent existence of each node enables nodes written in different languages to coexist. ROS is used by multiple industry-leading companies [32], and has become a standard for robotics, thanks to the modularity usually required in the field.

2.1.1 History

ROS originated at the Stanford Artificial Intelligence Lab in 2007, and was adopted by Google in 2008. It was maintained and developed by Willow Garage until 2013, at which point the Open Source Robotics Foundation, commonly known as Open Robotics, took its place [26]. Throughout the years, the SDK has gained support from multiple institutions, with its codebase receiving contributions from them as well as from volunteers all over the world [32].

2.1.2 Distinction Between ROS1 and ROS2

The transition from ROS1 to ROS2 was planned in such a way that no functionalities were lost, only old functionalities receiving improvements and new ones being added. Most relevant among these changes are the added support for most Operating Systems (OS), namely Windows and MacOS, whereas ROS1 was designed for Linux-based OSs, and the addition of Real-Time (RT) features, with inclusion of the Data Distribution Service protocol for communication between

nodes and Life Cycles which facilitate controlling the states of the nodes, among other necessary features to enable RT [24]. Additionally, the syntax of the core commands was altered, removing ambiguity when using both frameworks simultaneously and the build toolset was changed to add more capabilities and simplify the original process. A bridge package was also developed, to allow interaction between ROS1 and ROS2 nodes, effectively enabling the use of packages that have not been ported yet [36].

2.1.3 Learning ROS

Many ways of learning ROS are available, although there is much more material for ROS1 than ROS2. For the original framework, there are guidebooks, with varying degrees of detail [27, 26], video guides, online courses, and the tutorials in the official documentation [30]. Meanwhile, for the new version, there are mostly just comparisons between the two [36] and the official documentation, which is more extensive than the former version's counterpart [33]. This aspect will be explored in more detail in chapter 3.

2.2 Real Robots

Robots have existed for decades, both in professional and entertainment applications, with new developments and robots being deployed to support human workers every year [23].

2.2.1 Simulation Engines

A key tool in robot development is simulation. Physical robots can be very expensive, and minimal errors in programming can incur massive damages, making the use of simulated environments essential to lower costs and increase safety. Another advantage of these environments is the control over the simulation speed, allowing multiple runs of a simulated tasks to happen in the time that a single real run would take. This feature is especially helpful when training AIs through reinforcement learning, which is extensively used in the field of robotics.

In robot development, both 2D and 3D simulators are used. Some 3D examples are Gazebo, which has had ROS compatibility for years and is often the first simulator encountered by ROS beginners [31], and MuJoCo, a more recent and powerful physics simulation engine that has started being used for robot simulations together with ROS [11]. On the other hand, some 2D examples are the STDR simulator, a simple, dependable and very lightweight tool that had its last update for ROS Kinetic [41], and the Flatland simulator, a spiritual successor to the previous tool that remains lightweight yet improves on the simulation quality considerably [10].

2.2.2 AI Applied to Robotics

Robots are, by definition, automatons, which implies that they should be autonomous. Robot control has been done through algorithms ever since remote control by humans was deemed insufficient, and AI has been used for this effect extensively. This union originated the field of

intelligent robotics, and nowadays most robots use AI, be it to process raw data from sensors or to decide which actions to take based on the collected information. In recent years, this field has advanced rapidly, inevitably raising ethical [39] and legal [46] questions.

2.2.3 Real World Applications

Robots can be found everywhere nowadays. Some of the most notable applications are the following:

- Mobile manipulators used in assembly lines for the automotive industry, which can move heavy components with ease and perform dangerous tasks such as welding and painting with minimal risk to human employees [23].
- Wheeled platforms and automatic sorting machines used in warehouses, which expedite shipping processes and inventory management [7].
- Toys, such as the Aibo, that are able to interact with children and simulate both real and fictional entities, while also showcasing the potential of the field to possible future researchers and developers [35].
- Research robots such as the Mars Exploration Rovers [25], which perform tasks in environments that are inhospitable to humans.

2.3 Conclusion

It can be concluded that the prevalence of robots in all aspects of society is only going to increase, and the use of ROS along with it. Development will have to keep up with the demand for new, better robots, and researchers must take into account not only the requirements of the robot's environment, but also those of the society around it.

Chapter 3

Robotics Education

Robotics Education has become more prominent in recent years, with an upwards trend in literature production, even if irregular [22], and it has spread to most levels of education, from primary schools to university programs [3].

3.1 Educational Robot Platforms

Throughout the years, multiple environments were created or adapted from development tools to facilitate learning how to build and control robots. In this section, some of the more successful and prominent examples will be presented, along with highlights of their strong suits that should be kept in mind when developing new educational tools.

3.1.1 Simulated Platforms

The easiest learning platforms to acquire for robotics are often those that are purely virtual. There is great variety of simulation environments, with options for various programming languages and user levels of knowledge [42]. These platforms can cater to both those whose computers can handle the simulation programs and to those who cannot, with browser based alternatives for the most prominent simulators [38].

For those who want to learn ROS, one of the first stops on the journey is the ROS Wiki, whose tutorials include examples using RViz, the simplest visualization tool often used with ROS, and Gazebo, one of the more widespread simulation environments currently adapted to function with ROS [30]. The more prominent introductory literature to ROS also bases itself on the use of simulators, as can be seen both in O'Kane's and Newman's books [27, 26], both of which use examples based once again on RViz and Gazebo, with the latter also resorting to the Baxter Simulator for a more specialized approach to robotic arms.

When ROS is not a requirement, learners might opt for other approaches, however simulation remains as the most accessible platform. These can range from simulators with integrated

programming environments in visual programming languages aimed at smaller children like the Tactode robotic simulator [2], to standalone open source simulators used in academic and industrial applications like the Webots simulator [21].

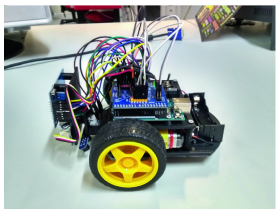
3.1.2 Physical Robot Platforms

Small and relatively simple physical robots are not as readily available as simulated environments, but are still fairly common and accessible in educational environments. Commonly, these platforms are wheeled, often including controllable wheel motors, various sensors, a Single Board Computer (SBC) as a controller and a plastic chassis. The options are diverse, and some of the most relevant are:

(a) Alphabot2 [12]

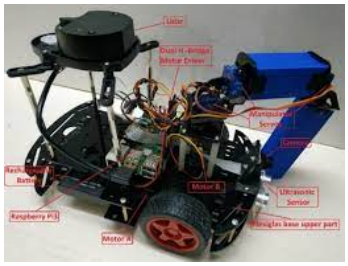
(b) Duckietown [14]

(c) Robobo [6]

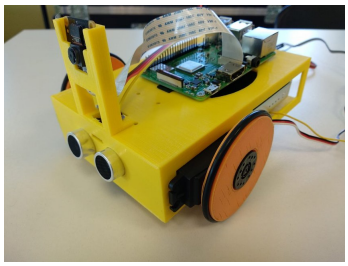


(d) Robot@Factory Lite [8]

(e) LEGO Mindstorms [17]



(f) EUROPA Robot [20]



(g) PyBoKids PiBot [44]

Figure 3.1: Physical Educational Robot Platforms

- The alphabot2 plaftorm, a small circular robot with multiple variations to support either an Arduino or a Raspberry Pi as the controller, and many different sensors. This robot's chassis is simply made up of its circuit boards, making it nearly impossible to rearrange any elements of it, and providing impractical extras such as the 8-directional joystick mounted upon it [12]. This robot provides no support for ROS [20] and no accompanying simulation, although external efforts have been made to develop one [28].

- The Duckietown platform, consisting of a simulated city with roads, wheeled robots and traffic lights, and even drones in some instances. This project started as a class at MIT [47, 37], and developed into an international education initiative, currently using simple

wheeled robots, with a hard plastic chassis, cameras as the main sensors, and either a Raspberry Pi or a Jetson Nano as a controller. Despite its simplicity, Duckietown robots have become somewhat costly due to using specialized circuit boards and Jetson Nano SBCs as controllers. Although their architecture is ROS based, it is complex and very specific steps must be taken to execute code within the robots [14]. Additionally, there is an accompanying simulator, although developed code is not directly portable unless the AI Driving Olympics interface is used [9].

- The Robobo robot, an innovative platform that uses a smartphone as its controller, compatible with multiple programming languages and frameworks, including ROS [5, 6]. This robot has already been applied as a teaching tool, along with a tailored curriculum, and has had reportedly good results [4]. Despite its simplicity, it is not easily replicable due to using closed-source hardware and the cost is driven up by the requirement of a smartphone, as it is not compatible with every model.

- The robots developed for the Robot@Factory Lite competition, which is part of the Portuguese Robotics Open, present a platform where the competitors can build their robot in different ways, within the provided specifications, and subsequently tackle a challenge with it. For use in a larger course, the platform would require modifications, as it is fairly specialized, but a good starting point nonetheless [8].

- The LEGO Mindstorms kits, which provide controllable parts that allow the construction of robots with nearly any form imaginable, and support multiple programming languages [17]. These have applied to education many times, for example to teach autonomous robotics [1], or even in tandem with mobile devices [43]. This platform does not have a native accompanying simulation, however there are efforts to create one [19]. The main weaknesses of these kits are the use of closed-source hardware and software and their relatively high cost compared to regular small electronics.

- The EUROPA project, which developed a wheeled robot with a wide array of sensors and a robotic arm mounted on it, provides an impressive platform to teach robotics, if not slightly too complex for students in earlier stages of their study program. This project took in consideration many of the robots analysed in this document, and provides an excellent example of the methodology to follow [20].

- The PyBoKids project, which presents multiple approaches to a low-cost open source educational robot, namely an Arduino based robot and a Raspberry Pi based robot, with a 3D printed chassis and accessible sensors. Both approaches possess multiple sensors, although only the Raspberry Pi version includes a camera, and both have a representation for use in the Gazebo simulator, as an accompanying simulation [44, 45].

3.2 How and When Robotics is Taught

3.2.1 Pre-University Level

Although robots have been in the hands of children for a long time, notably in the form of the Aibo and Furby toys [35], only recently has the focus of Robotics Education expanded to the younger crowd [22]. Programming is ever more present in both official curricula and extracurricular activities, using beginner-friendly languages like Scratch and Python, and low-cost platforms like Arduino and Raspberry Pi connected to small actuators such as LEDs. Some robot platforms were built for use in these levels of education, like the Robobo platform, which supports both Scratch and Python [6], the robotic simulator developed for use with Tactode, a tangible programming language [2], and the PyBoKids project, which aims to teach robot programming in Python to pre-university students [45]. Additionally, the consensus seems to be that in these environments learning by doing is the most effective method [3].

One other notable type of initiative that encourages robotic education at these levels is competitions such as RoboCup, an international competition with various robotics challenges ranging from playing soccer to simulated rescue missions. This particular example includes categories oriented to children, in an effort to make learning robotics appealing to all ages [13].

3.2.2 Higher Education Level

In university environments, robotics courses can often be found as electives in MSc and PhD programs, as well as in workshops and other extracurricular activities. However, these are often difficult for students who are still near the start of their studies, as robotics often requires knowledge of programming languages, algorithms and AI, computer vision, and electronics [40]. The use of ROS adds to this difficulty, as the framework must be learned before it can be used effectively.

3.2.3 ROS1 Courses and Introductions

Throughout ROS1's lifecycle, many learning materials were published, ranging from simple tutorials to detailed books. The most direct form of learning can be found in the form of simple tutorials in the ROS1 Wiki [30], and most other introductions are based on this to some degree.

One of the most extensive and detailed introductions is ROS By Example, by Goebel, composed of two volumes which detail all functionalities of the framework, along with multiple simulators and packages. However, this introduction has only been updated up to ROS Indigo, and is somewhat outdated [15, 16].

A simpler and friendlier guide is "A Gentle Introduction to ROS", by O'Kane, which focuses solely on teaching the main features of ROS1, and finishes by directing the reader to other tools where they can learn about external packages and simulators [27]. A middle ground can be found in Newman's book, which is based on Goebel's work, but still not as extensive, maintaining the focus on learning the more central aspects of ROS while leaving aside some packages and simulators that are more advanced or less useful [26]. Both these books use ROS Indigo, an outdated

distribution that has reached End-of-Life status, but since the main concepts of ROS have not changed since, the instructions are still mostly accurate [29].

Other means of learning can be found in the form of online courses[1], paid or otherwise, and video tutorials[2].

3.2.4 ROS2-Focused Courses

Due to how recent ROS2 is, the external tools for learning how to use it are still scarce, with most resources focusing on the differences between the versions [36]. However, the official documentation is far more extensive, with build and installation guides for most OS types, tutorials on most features of the framework, how-to guides and a description of its main concepts [33].

3.3 Conclusion

The interest in robotics is spreading to all levels of education, but its presence is not yet solidified. Future roboticists already play with robot toys, but still find a wall when seeking knowledge about these systems, one that requires prior expertise in various other fields. Current efforts are making the field more approachable, and even complex systems such as ROS are becoming friendlier to beginners, suggesting that robotics will be accessible to all in a near future.

[1] https://www.udemy.com/course/ros-tutorials/?referralCode=CE60EFAB8BA458EC024C
[2] https://www.youtube.com/playlist?list=PL8dDSKArO2-m7hAjOgqL5uV75aZW6cqE5

Chapter 4

Educational Package Design

4.1 Introduction

The proposed educational package is meant to contain a replicable toolkit for students to use in the learning process, providing a platform where they can experiment and learn through trial and error, rather than simply studying the theory, and a lesson plan which, paired with the previous element, should provide a practical first contact with ROS-driven robotics to students, allowing them to apply the theoretical parts of each lesson in experiments with real and simulated environments almost instantly.

The primary focus is set on elaborating a foundation for the toolkit, which requires a physical mobile robot and an accompanying simulation that can be used in most of the students' and the institutions' computers, as well as introductory lessons to both ROS1 and ROS2 and an introduction to Reinforcement Learning, thus proving the package's capabilities throughout the range of a robotics course's lessons.

As an educational tool meant to serve as the basis for an intelligent robotics course, the package has the following learning outcomes for the students:

- Summarize key concepts of ROS, such as nodes, topics, packages and workspaces.
- Prepare a machine to be able to develop basic ROS packages (install ROS, create a workspace and set up environment variables).
- Build a ROS package capable of reading from a sensor or sending data to an actuator.
- Perform experiments with a virtual robot in a simulated environment.
- Integrate a physical robot into an existing ROS-based system.
- Use visualization tools (namely RViz and simulator interfaces) to follow the progress of a robotic system.

- Build a RL system for a robot, where training and execution can be performed either with real hardware or via simulation.

4.2 Requirements

4.2.1 Simulation Engine

One of the key elements of the package is a robotic simulation, where environments for experiments can be set up. The simulation must be usable in most computers, directly or through virtual machines, so every student can have access to their own simulation and experiment not only in groups, but also individually. As such, the following requirements are set for the simulation engine:

- Lightweight, that is, does not require excessive amounts of processing power, specialized hardware components or unique software dependencies to run reliably.
- Controllable timescale, allowing the simulation to be sped up, for example to train AI agents, and slowed down, for example to allow a clearer analysis of the results.
- Support for ROS1 and ROS2, as the goal of the package is to provide an environment to learn robotics in both versions.
- Not in End-Of-Life or deprecated status, as ROS is still an evolving architecture and bugs may need to be corrected in the future.
- Open-source, so access by students and teachers is facilitated and contributions such as new features and fixes are possible.
- Easily configurable, allowing parameters to be changed for different experiments that may require a wide veriety of environments.

4.2.2 Physical Platform

Robots come in many sizes and shapes. One of the most common configurations is a small differential drive robot, with two wheels and a pod for minimalist mechanics while maintaining high flexibility, for instance to teach robot control topics ranging from simple controllers to Reinforcement Learning.

The educational package is meant to include a small mobile robot, which can be seamlessly swapped with the simulation, so students can experience operating real robots and test the code they develop in both environments. Since these tools are expected to be used with entire classes of students, each group of students should have access to their own robot and thus the following requirements are set:

- Replicable, meaning multiple instances of the robot should be possible to be acquired and/or made without excessive effort or time spent.

- Low-cost, ideally under 300€ per unit, since costs rapidly multiply when acquiring robots for an entire class of students divided in small groups.
- Open-source, so anyone can build a custom instance of the robot and learn robotics, or even contribute to improve the package.
- Mobile, as mobile robots provide a beginner-friendly platform with clear feedback of the experiments to the student.
- Multiple sensors, either real or virtual sensors developed through other sensors such as a camera, to familiarize students with the various types of data they can work with.
- Safe for handling by students, with no sharp edges or exposed wires on the outer area of the robot.

4.2.3 Lesson Plan

The final element of the package is the lesson plan. This plan should cater to students of diverse backgrounds, and at various points in their studies. The requirements set for it are the following:

- Support for ROS1 and ROS2, as this is a key point of the package.
- Contains lessons in Python and C++, as ROS offers resources to develop in both languages and students may prefer either.
- Covers the essentials of ROS, as most of the concepts are common to both versions and central to the package's objective.
- Introduces the student to central concepts of robot development, such as sensors, robot control and AI applied to robotics.
- Satisfies the learning outcomes described in section 4.1.

4.3 Simulator Selection

In order to select the most appropriate robotics simulation to be used in the package, multiple simulation engines were analyzed. Table 4.1 shows a comparison between the simulators explored considering the requirements set previously.

Originally, the STDR simulator was considered as an option, however the project has been abandoned for a few years, and thus it was kept only as an example of what was desired for the package. Due to the lightweight requirement, most three dimensional simulators were discarded, as they require considerable (or at least special or dedicated) computational resources. Flatland was selected as it fulfilled all of the necessary requirements, and was confirmed to be supported through contact with the maintainers.

Table 4.1: Simulator Comparison

Name	Dimensions	Lightweight	Variable Time	ROS-Friendly	Open-source	Supported
Gazebo	3D	No	Not Usable	Yes	Yes	Yes
CoppeliaSim	3D	No	Yes	Yes	No	Yes
BRAX	3D	No	Yes	No	Yes	Yes
MORSE	3D	Somewhat	Unclear	Yes	Yes	Yes
ARGoS	2D/3D	Yes	Unclear	No	Yes	Yes
STDR	2D	Yes	Unclear	Yes	Yes	No
Flatland	2D	Yes	Yes	Yes	Yes	Yes

4.4 Architecture

The proposed package's architecture involves two main elements: the environment, which contains the robot and, in the simulated environment's case, the representation of the world, and the agent or controller, which makes decisions based on sensor data published by the environment side and subsequently communicates the desired actions through Twist messages, one of the most standard ROS message types used in robot control. An abstract diagram is shown in figure 4.1. For each abstract element of this architecture, the concrete components that can implement it are designed to be interchangeable, allowing any combination of environment and controller to function.

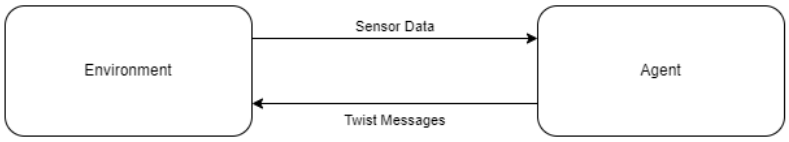
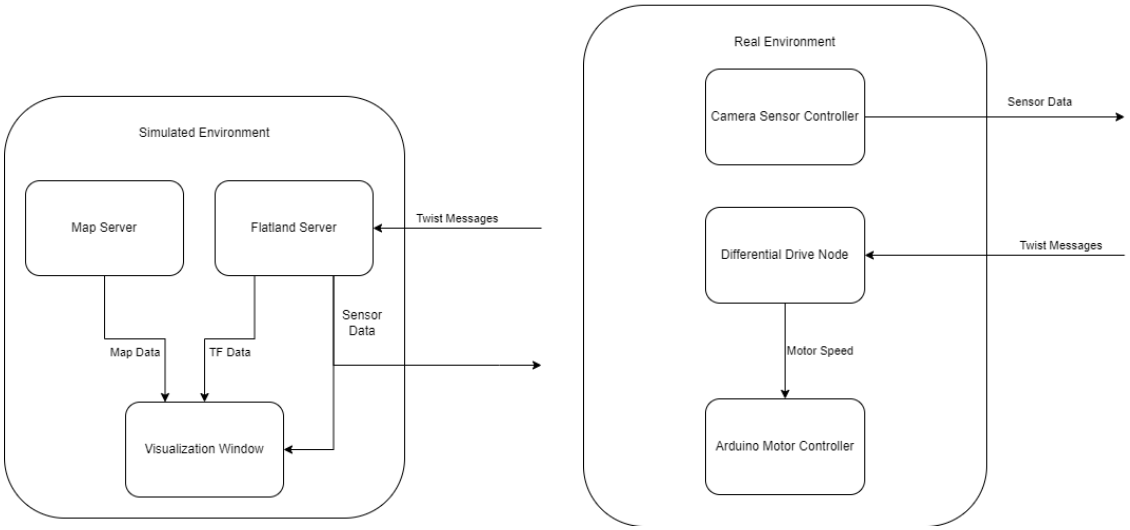


Figure 4.1: Abstract Architecture

The environment element can take one of two forms:

- The simulated environment, containing the Flatland Server, which runs the simulation, receives control commands in the form of Twist messages and publishes sensor data, the Visualization, which can be either Flatland's native visualization or a custom RViz configuration, receives data from the previous element and presents it in a user-friendly manner, and the Map Server, which provides the map representation to RViz, if it is being used for visualization. A diagram of this element is shown in figure 4.2a.
- The real environment, which contains the Camera Controller, publishing both raw data from the camera and sensor data generated by the virtual sensors, the Arduino Motor Controller, which controls the wheels, and the Differential Drive Node, which receives control commands in the form of Twist messages and translates them to the motor controller. A diagram of this element is shown in figure 4.2b.



(a) Simulated Environment Architecture (b) Real Environment Architecture

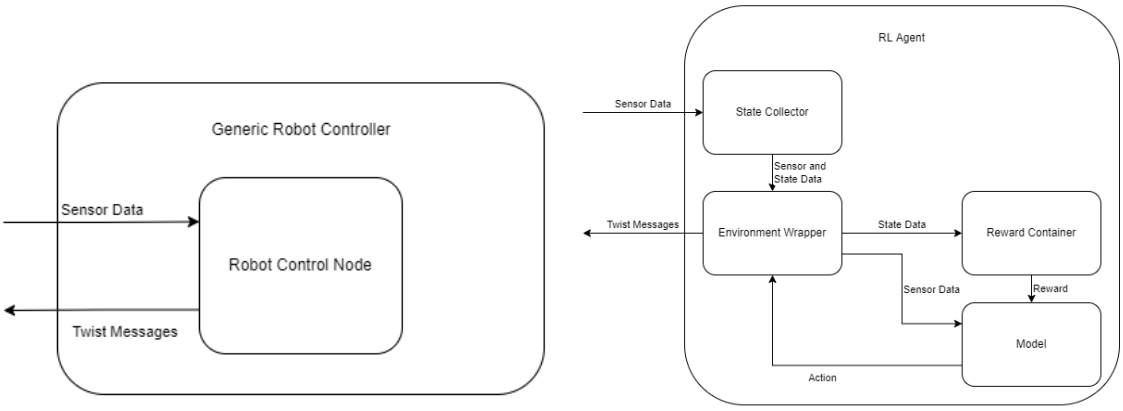
Figure 4.2: Environment Component Architectures

The agent or controller element can take one of two forms:

- A simple robot control node, which receives sensor data from the environment element and publishes Twist messages to control the robot, as shown in figure 4.3a. This element represents any node or group of nodes developed by a student during the lessons, and any sample control nodes provided in the package.
- A Reinforcement Learning agent, consisting of a State Collector which receives sensor and odometry data from the environment, organizes it and passes it to the environment wrapper, a Reward Container which calculates a reward based on the state of the robot and passes it to the model during training, the Model, which decides which action to take based on the sensor data received, and an Environment Wrapper, which contains training and execution parameters, distributes data throughout the agent's components, and communicates actions chosen by the model to the robot via Twist messages. A diagram of this element is shown in figure 4.3b. This architecture is based on the Reinforcement Learning framework developed by Güldenring [18] and its adaptation and usage is further described in chapter 6.

4.4.1 Use Case Example

A sequence diagram showing the expected interaction loop between the package elements is depicted in figure 4.4. Upon launching the system, the elements are designed to run indefinitely, with the environment component regularly publishing sensor data, and the agent subscribing to it and publishing control data in response.



(a) Generic Robot Controller Architecture (b) RL Agent Architecture

Figure 4.3: Agent Component Architecture

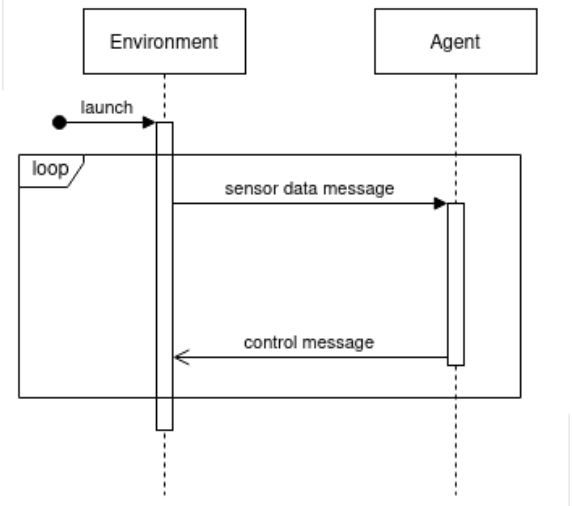


Figure 4.4: Sequence Diagram

Chapter 5

Educational Package Development

The first step in the development of the package is to put together the toolkit, specifically the physical robot and its accompanying simulation, as the lessons will be based on it. This chapter is divided in two main sections; section 5.1 focuses on setting up the Flatland simulator for teaching ROS-based robotics and includes the development of a pair of introductory hands-on tutorials, one for each ROS version. Section 5.2 details the setup of the physical robot used throughout this project, along with the virtual sensors created with the camera. Note that building the physical robotic platform from scratch is not in the scope of this work, as an existing robot was used as a base.

5.1 Simulation Development

The selected simulation environment, Flatland, is a two-dimensional robotics simulator which uses the Box2D physics engine and is built as a group of ROS nodes, facilitating the development of ROS-based robotics. Although the simulator itself required no modifications, representations of the robot, obstacles and the environment had to be developed and a specialized RViz configuration was built to display sensor data within the environment. After this preparatory work, two introductory tutorials were developed, one for each ROS version, with sample code in both Python and C++, effectively an example of one of the first lessons of the idealized lesson plan.

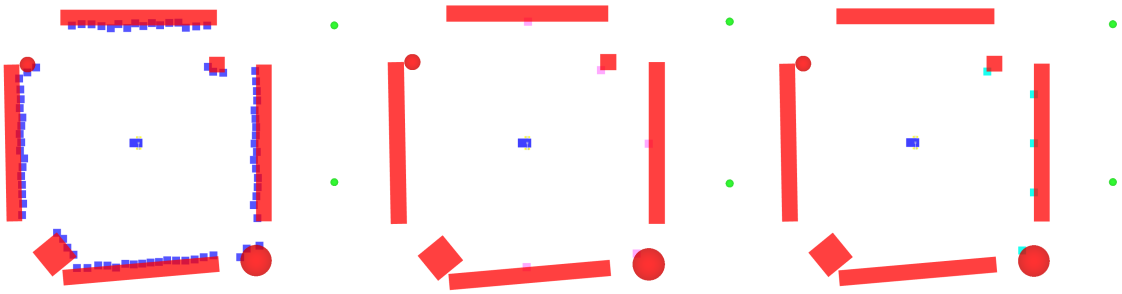
5.1.1 Simulation Assets

Although Flatland includes some sample assets, namely maps and simple robots, the representations of obstacles and the robot used, and the world definition still had to be created. The obstacles were defined as circles and squares, as the simulation is two-dimensional, and two separate representations of the real robot were created, one with detail, defining the robot body and the wheels as separate bodies (fig. 5.1a), and the other defining the entire robot as a single body (fig. 5.1b). This was done due to the proximity between vertices in the first representation causing the simulator

to classify them as degenerate points, which results in the simulation failing to generate the robot representation due to not having enough vertices to define the bodies. Additionally, three different simulated sensors were placed on the robots: a simulated Light Detection And Ranging (LiDAR) with 90 rays equally spread around the robot, a simplified version of the previous LiDAR with 7 rays, and a front mounted LASER sensor with 5 rays. In figure 5.2 the differences between the sensors can be seen, where each colored square represents the point detected by each ray of the sensor. The first two were used in the tutorials in section 5.1.2 and the last was used for the Reinforcement Learning experiments described in chapter 6, as it simulates the most accurate of the virtual sensors developed for the real robot.

(a) Detailed Robot Model (b) Simplified Robot Model

Figure 5.1: Robot Representations



(a) Realistic LiDAR (b) Simplified LiDAR (c) Front Mounted Laser

Figure 5.2: Simulated Sensors

Lastly, some debug tools were developed[1], as these proved useful in the preparation of the simulation environment and in the development of the tutorials. Among these are:

- The RViz configuration, which allows the user to visualize sensor data in the simulation environment, as opposed to the regular Flatland visualization window.

[1] https://github.com/BerserkingIdiot/robot_control_utils_ros1

- Two publishing ROS nodes, one of which allows the user to control the robot via the keyboard and another which saves and publishes trajectory data for user visualization.
- A sensor data collection tool and a visualization tool, which, when combined, allow the user to save data from LaserScan messages to files and subsequently see the values over time it in an animated graph.

5.1.2 Introductory Tutorials

In addition to the previously described assets, two introductions to robotics simulation with Flatland were developed, one for ROS1[2] and one for ROS2[3], which serve as a first exercise for the proposed lesson plan. Each of these consists of a simple guide on how to set up a ROS workspace containing the Flatland simulator, an environment with the assets described in the previous section, and sample code in both C++ and Python, and instructions on how to launch the simulation and control nodes written by the user. This is meant to provide a hands-on first contact with ROS and simulation where the user can learn experiment without prior knowledge of robotics. The tutorials also double as an example of the proposed solution, allowing students and teachers to provide objective feedback and suggestions.

These tutorials are expected to satisfy the following outcomes:

1. Prepare a machine to develop basic ROS packages with Flatland.
2. Analyse the progress of a virtual robot in 2D simulation with visualization tools.
3. Modify the code controlling a virtual robot in 2D simulation.

In order to evaluate the tutorials' pertinence, Google Form surveys were added at the end, so users could provide relevant feedback. Amongst the information collected was the user's prior experience with the field and the tools used, their perception of the field after following the tutorial, and their opinion on the fulfillment of the proposed learning outcomes. All questions using numeric value answers accept integers between 1 and 7. The forms used can be seen in appendix B.

The ROS1 tutorial results presented a sample size of 11, with users of various levels of contact with robotics (M = 3.73, SD = 2.26). This tutorial yielded positive results relating to the expected learning outcomes, with the questions relating to them presenting high mean values (M) and small standard (SD) deviation values, as shown in table 5.1. A visual representation is also shown in figure 5.3. The values were somewhat similar when dividing responses by proficiency in robotics, based on answers to the first question. Notably, the second learning outcome presented the largest discrepancy between beginner and expert users. This is theorized to be due to the second group having used more complete visualization tools in the past, thus having a more defined point of comparison.

[2] https://github.com/BerserkingIdiot/flatland_quick_start_ros1
[3] https://github.com/BerserkingIdiot/flatland_quick_start_ros2

Table 5.1: ROS1 Learning Outcome Results

Proficiency	Count	Value	Learning Outcome		
			1	2	3
Overall	11	Mean	5.73	5.64	5.64
		Standard Deviation	1.05	0.98	0.88
Beginner	6	Mean	5.67	6.00	5.50
		Standard Deviation	1.11	1.00	0.96
Expert	5	Mean	5.80	5.20	5.80
		Standard Deviation	0.98	0.75	0.75

The ROS2 tutorial results presented a smaller sample size than its ROS1 counterpart, with 9 total responses, however it still had a representative distribution (M = 3.67, SD = 2.16). This tutorial yielded positive results in the learning outcomes section, similarly to the other version, which can be seen in table 5.2. A visual representation is also shown in figure 5.4. These similarities with the ROS1 version are consistent with both tutorials being very similar. A notable difference can be seen in the results of the third learning outcome for beginners, however. One possible reason might be due to some added complexities in ROS2 over ROS1, such as the need to compile code written in Python, which the older version does not entail.

Table 5.2: ROS2 Learning Outcome Results

Proficiency	Count	Value	Learning Outcome		
			1	2	3
Overall	9	Mean	5.89	5.89	5.33
		Standard Deviation	0.87	0.87	1.05
Beginner	6	Mean	6.00	6.00	5.17
		Standard Deviation	0.82	0.82	1.07
Expert	3	Mean	5.67	5.67	5.67
		Standard Deviation	0.94	0.94	0.94

A complete transcript of the responses received is included in appendix C.

5.2 Physical Platform Development

5.2.1 Robot Setup

A low-cost physical robot is an essential component of the proposed educational package, and so developing or acquiring a robot that fit the requirements was one of the project's goals. In an effort to reuse resources and save time, a robot developed for a previous project by FEUP students [4] was selected, as the requirements set for the robot at the time were similar to those set in this proposal. The robot can be seen in figure 5.5.

[4] https://github.com/jorgef1299/SERP

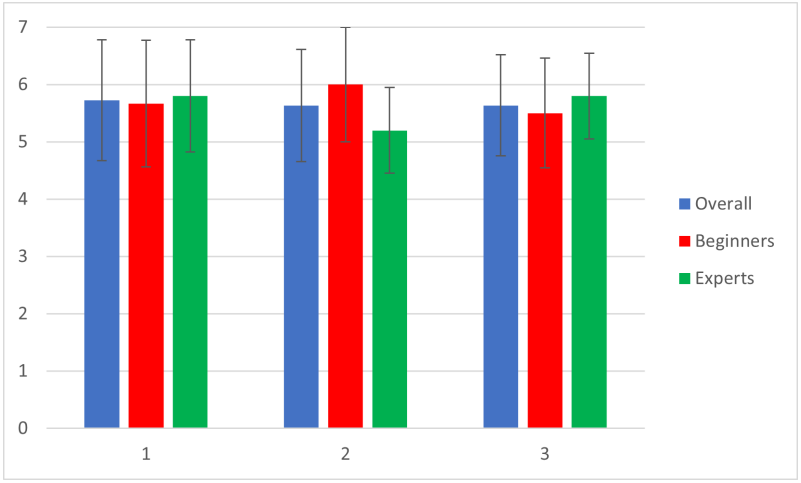


Figure 5.3: ROS1 Learning Outcome Results

The robot in use is a two-wheeled ground robot, containing a wide lens camera mounted above the robot as its sole sensor, an Arduino microcontroller dedicated to controlling the wheel motors, and a Raspberry Pi 4B single board computer serving as the main controller of the system. Originally, the motors were controlled individually and set to only allow the robot to move forward. The Arduino code was modified to allow the motors to move both ways and a differential drive node was developed to allow the control of the robot via Twist messages[5]. The entire system is powered by two 18650 lithium on-board batteries mounted on a so called "UPS" board for the Raspberry Pi [6].

5.2.2 Virtual Sensors

In order to reduce the cost of the proposed robot while keeping the possibility of having multiple different sensors, the robot in use possesses a wide lens camera placed above the robot and aimed vertically down, providing a view all around the body of the robot. However, this introduces a new issue, as image processing is an advanced topic which may not be part of the programs of some of the students that would use this package. The proposed solution to this was to develop virtual sensors based on the camera, providing data analogous to that of a sensor that is simpler to process, such as laser or sonar sensor.

The virtual sensors developed as part of this project function as individual ROS nodes containing a pipeline which processes camera frames and outputs sensor data, as shown in 1. Due to the methods used, the sensors can be used in any uniform surface, independently of color, as obstacles are detected through contrast. For this first iteration of the physical robot, three separate virtual sensors were developed[7]:

[5] https://github.com/BerserkingIdiot/robot_control_utils_ros1

[6] https://www.seeedstudio.com/UPS-With-RTC-Coulometer-For-Raspberry-Pi-4B-3B-3B-p-4639.html

[7] https://github.com/BerserkingIdiot/robot_control_utils_ros1

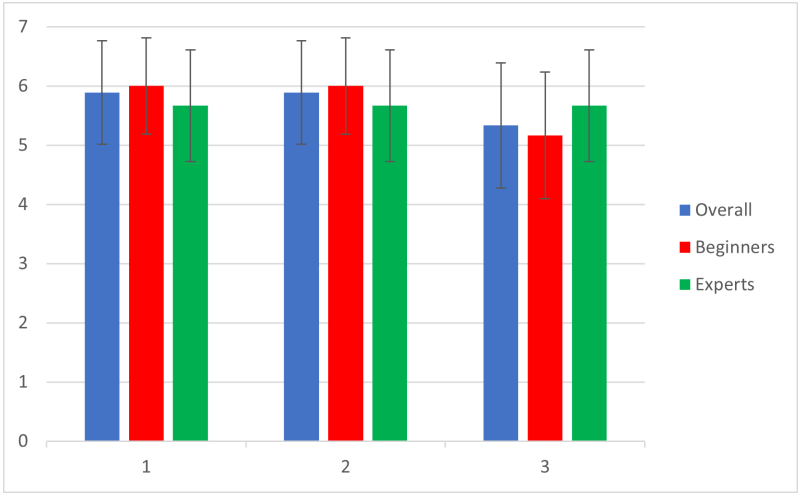


Figure 5.4: ROS2 Learning Outcome Results

- A LiDAR with 90 rays, equally spread around the robot, providing data about obstacles with values ranging from 135 to 220. These values are merely relative, as they describe pixel distance rather than real distance and some rays on the sides of the robot provide false information due to the camera support structure blocking the view.

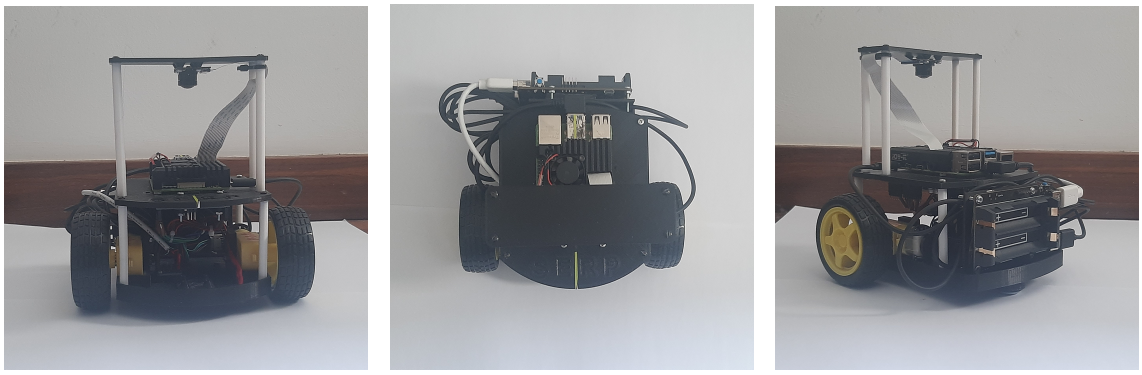
- An object detection sensor, providing a proximity values between 0 and 3 for objects detected on four locations around the robot, these being front, front left, front right, and rear. This sensor is based on previous work from the team that developed the original form of the robot in use [8], and provides user-friendly data that can be easy for a student to understand and use.

- A laser sensor, with 5 rays distributed over a 90º area in front of the robot and a similar setup on the rear, providing approximate values of detected obstacles. Due to the characteristics of the camera lens and its positioning, the equation for each ray was calculated individually through curve fitting, to provide data as accurate as possible up to a range of 1,5 meters, with an estimated maximum range of 2 meters. In table 5.3 the virtual values used to calculate

[8] https://github.com/jorgef1299/SERP

Algorithm 1 Image Processing Pipeline

while ROS running **do**
    $frame \leftarrow get\_camera\_frame$
    $frame \leftarrow grayscale\_conversion(frame)$
    $frame \leftarrow bilateral\_filter(frame)$
    $frame \leftarrow canny\_filter(frame)$
    $points \leftarrow contour\_detection(frame)$
    $sensor\_data \leftarrow sensor\_specific\_processing(points)$
    Publish $frame$
end while



(a) Front of the robot (b) Top of the robot (c) Rear of the robot

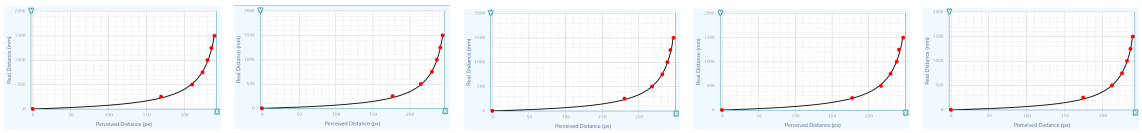
Figure 5.5: SERP robot

the equation for the ray at each angle and the respective resulting equation can be seen, with the corresponding real values in the first line. Additionally, figure 5.6 shows the points and approximate equations for each ray. The images with greater resolution are included in appendix D.

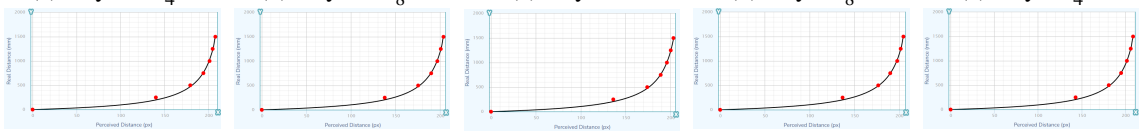
Table 5.3: Virtual laser sensor distance calculation

	0	250	500	750	1000	1250	1500	Resulting Equation
$-\frac{1}{4}\pi$	0	169	210	224	230,5	235,5	239,5	$y = \frac{-118.7824*x}{-258.2981+x}$
$-\frac{1}{8}\pi$	0	176,5	214,5	229,5	236,5	241	244	$y = \frac{-106.463*x}{-261.4368+x}$
0	0	180	218	232	239	243	247	$y = \frac{-108.4156*x}{-264.6905+x}$
$\frac{1}{8}\pi$	0	177,5	217	230	238	241,5	247	$y = \frac{-122.4684*x}{-266.7484+x}$
$\frac{1}{4}\pi$	0	174	212	225,5	232	236,5	239,5	$y = \frac{-104.0271*x}{-256.1551+x}$
$-\frac{3}{4}\pi$	0	140	179	194	201	204,5	207,5	$y = \frac{-119.897*x}{-224.2417+x}$
$-\frac{7}{8}\pi$	0	137,5	175	190	196,5	200,5	203,5	$y = \frac{-124.4037*x}{-220.4973+x}$
π	0	136	174	189	196	200	203	$y = \frac{-127.9308*x}{-220.4707+x}$
$\frac{7}{8}\pi$	0	136,5	176,5	190	197,5	201,5	204,5	$y = \frac{-127.1675*x}{-221.9849+x}$
$\frac{3}{4}\pi$	0	143	181	195	201,5	206	208,5	$y = \frac{-118.8645*x}{-225.261+x}$

For each sensor, a comparison between bright and dark conditions was done and can be seen in figure 5.7. In a future version of the sensors, improvements to mitigate the decay in performance in dark environments should be considered.

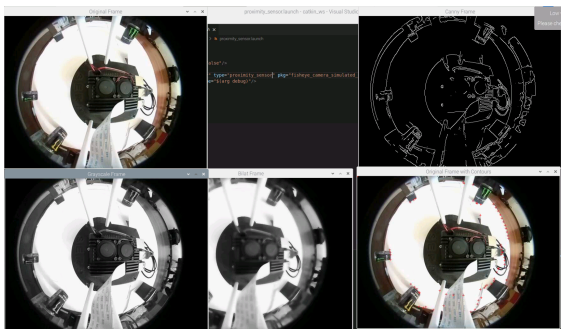


(a) Ray at $-\frac{1}{4}\pi$ (b) Ray at $-\frac{1}{8}\pi$ (c) Ray at 0 (d) Ray at $\frac{1}{8}\pi$ (e) Ray at $\frac{1}{4}\pi$

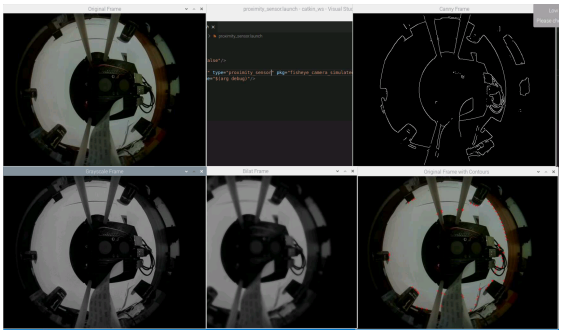


(f) Ray at $-\frac{3}{4}\pi$ (g) Ray at $-\frac{7}{8}\pi$ (h) Ray at π (i) Ray at $\frac{7}{8}\pi$ (j) Ray at $\frac{3}{4}\pi$

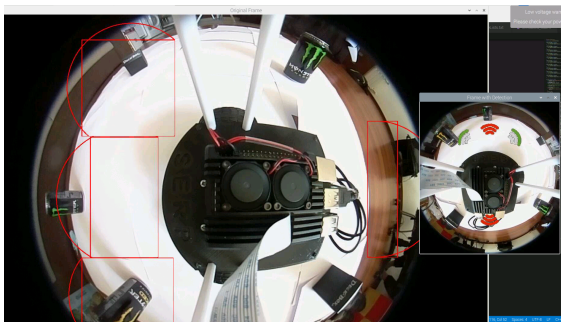
Figure 5.6: Virtual laser sensor distance equations



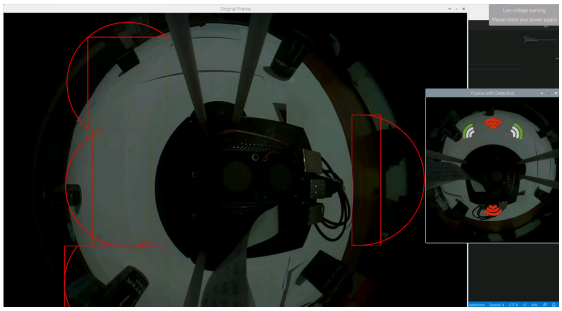
(a) LiDAR in bright conditions



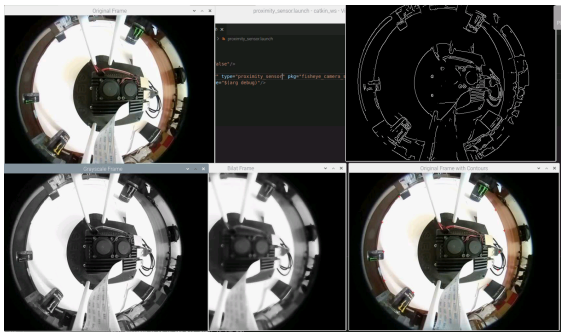
(b) LiDAR in dark conditions



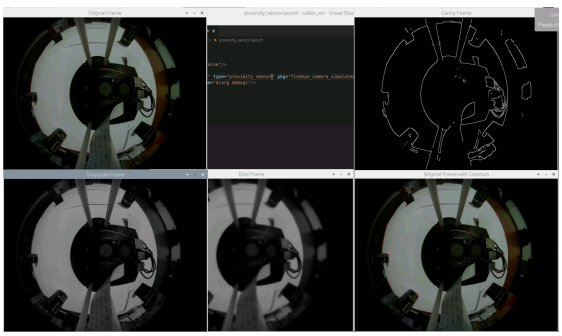
(c) Object detector in bright conditions



(d) Object detector in dark conditions



(e) Laser sensor in bright conditions



(f) Laser sensor in dark conditions

Figure 5.7: Sensor comparison in different lighting conditions

Chapter 6

Reinforcement Learning Lesson

One of the key elements of robotics development is AI, such as RL, which is often used to control robots, and so it is often introduced in robotics courses, even taking a central position in those that focus on intelligent robotics. As such, an educational package for teaching robotics should support the use of AI, and the lesson plan should contain an introduction to the topic. This chapter focuses on the development of a reactive robot control model, based on the work of Güldenring [18], that can be used as an example of how to train an agent with RL and allows students to experiment with parameters.

6.1 Framework Adaptation

The original Deep Reinforcement Learning framework used as a base for this lesson was developed to train an agent meant to control a mobile ground robot equipped with a LiDAR in following a given path made of multiple waypoints in an environment containing both static and moving obstacles. In this form, the agent had access to sensor data from the LiDAR and the positions of the waypoints relative to its own. Additionally, the framework uses the well known Proximal Policy Optimization[34]. Despite being far more complex than is required for the lesson, this framework was chosen due to a few key factors; firstly, it is ROS-based, eliminating the need of adapting an existing RL environment to the ROS architecture, secondly, it uses Flatland as a training environment, tying in to the educational package developed so far, and lastly, the robot used is a differential drive ground robot with a similar size to the one used in the development of the package so far, reducing the amount of adaptations necessary.

Some features of the framework were considered unnecessary or inappropriate and were either removed or modified to satisfy the needs of the lesson. The changes are as follows:

- The integration with PedSim (a tool that simulates pedestrian movement in a two dimensional environment) was removed, as it would add complexity to the system, potentially

requiring more processing power, and dynamic obstacles could prove to be too advanced for a first experience with Reinforcement Learning.

- The waypoint system was modified to only provide a marker of the robot's starting position for the effects of reward calculation, as in the lesson the goal is to generate a simple reactive agent, eliminating the need to provide a path to follow.
- The robot representation in the Flatland simulation was modified to represent the real robot used in the development of the educational package, to enable portability of the agent between the simulation and the physical robot.

One issue introduced by the adaptions is that the reward function used by the model becomes overly simplified. Before the changes, the reward function (shown in equation 6.1) includes four components: a punishment for approaching and colliding with obstacles, a punishment for standing still, and a reward for approaching the nearest waypoint that has not yet been visited and reaching the goal. With the removal of the waypoint system, the third and fourth components become obsolete, simplifying the reward function (as shown in equation 6.2) and potentially introducing faults into the model. This issue is addressed during the experiments described in the following section.

$$reward = obstacle\_punish + still\_punish + wp\_rew + goal\_rew \tag{6.1}$$

$$reward = obstacle\_punish + still\_punish \tag{6.2}$$

$$obstacle\_punish = \begin{cases} obstacle\_punish\_factor, & \text{if } min(scan\_ranges) < robot\_radius \\ 0.0, & \text{otherwise} \end{cases} \tag{6.3}$$

$$still\_punish = \begin{cases} still\_punish\_factor, & \text{if } linear\_vel = 0 \ \ \& \ \ angular\_vel = 0 \\ 0.0, & \text{otherwise} \end{cases} \tag{6.4}$$

6.2 Learning Experiments

In order to produce a model that can serve as an effective first contact with RL, multiple experiments were conducted. Most settings were tested with the default values shown in figure 6.1, as one of the main exercises expected from students is to change these values and compare results. The following experiments focus mainly on improving state collection and reward calculation.

The robot used in these experiments is a representation of the real robot used throughout the project, using differential drive for locomotion, controlled by Twist [1] messages, and a simple front mounted laser sensor with five rays distributed equally over a 90 degree area outputting data via LaserScan [2] messages.

[1] Twist ROS message - see, for example, http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html, visited July 4, 2022

[2] LaserScan ROS message - see, for example, http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html, visited July 4, 2022

```
Starting PPO2 Training of agent: ppo2_simple_reactive
-----------------------------------------------------
gamma                               0.990000
n_steps                             128
ent_coef                            0.005000
learning_rate                       0.000250
vf_coef                             0.500000
max_grad_norm                       0.500000
lam                                 0.950000
nminibatches                        1
noptepochs                          1
cliprange                           0.200000
total_timesteps                     2000000
Policy                              CNN1DPolicy_multi_input
reward_fnc                          19
Normalized state: 0
discrete action space 0
Number of stacks: 3, stack offset: 5
```

Figure 6.1: Default Reinforcement Learning Model Parameters

6.2.1 Original State Collection and Original Reward Function

The agent was trained with the parameters and reward function unchanged past the adaptations to the general framework described in section 6.1. All attempts resulted in the robot moving in circles around its starting position. Despite mostly avoiding obstacles, this behaviour was deemed unsatisfactory and thus improvements were introduced in the next experiment. Additionally, the factors present in the original reward function, namely those shown in equations 6.3 and 6.4.

6.2.2 Original State Collection and Modified Reward Function

The reward function was modified to include a reward for moving away from the starting position and a punishment for moving back towards it (as shown in equations 6.5 and 6.6), and all other parameters were kept unchanged. Upon training, the robot moved away from the starting point as desired, despite still moving in circles in some rare cases. There were still some collisions with obstacles, however. This was assumed to be caused due to insufficient data in the state used by the reward function, rendering the obstacle collision portion of it ineffective. This issue was addressed in the next experiment. The factors introduced in equation 6.6 were tuned during the experiments as well.

$$reward = obstacle\_punish + still\_punish + move\_away\_rew \tag{6.5}$$

$$move\_away\_rew = \begin{cases} dist\_diff * move\_rew\_factor, & \text{if } dist\_diff \geq 0 \\ dist\_diff * move\_punish\_factor, & \text{if } dist\_diff < 0 \end{cases} \tag{6.6}$$

$$dist\_diff = current\_dist\_to\_start - previous\_dist\_to\_start \tag{6.7}$$

6.2.3 Modified State Collection and Modified Reward Function

An additional sensor, a LiDAR that collects data from 90 rays distributed equally over a 360 degree range, was added to the robot to provide more accurate state data to be used in the reward calculation. This data is only passed to the reward function, with the agent still only using the original sensor as an input. Upon testing, the robot behaved mostly as expected, moving away from the starting point and avoiding obstacles throughout its path most of the time. Some cases still resulted in collisions with obstacles, and so further tuning was performed with the factors of the reward function mentioned in the previous experiments.

6.2.4 Results

The final value of the tuned parameters can be seen in table 6.1. All three configurations described in the experiments were trained with these parameter values to provide a clear comparison between their performance.

Table 6.1: Reward function parameter values

Factor	Final Value
still_punish_factor	-0.001
obstacle_punish_factor	-7.0
move_rew_factor	2.5
move_punish_factor	-2.0

Figure 6.2 shows the trajectory, in red, followed by agents trained in each of the configurations throughout similar environments with randomly placed obstacles. More detailed examples can be seen in appendix E.

In the first configuration, the agent continued to move in circles around the starting point, with the reward parameter changes not affecting the results, as seen in figures 6.2a and 6.2b. In this configuration, the agent invariably learned this behavior and converged very quickly.

Notably, in the second configuration the agent behaved as desired, moving away from its starting point in a straight line when no obstacles are present and avoiding them consistently, even going as far as following walls, as seen in figures 6.2c and 6.2d. Despite the agent learning to avoid obstacles quickly, by the end of the allotted training time the agent was still showing improvements, indicating that with more training it could be further optimized.

The third configuration did not improve much from the parameter changes, moving somewhat erratically around the environment, while still avoiding obstacles, as seen in figures 6.2e and 6.2f.

The learning process in this configuration was inconsistent, with the agent seemingly getting stuck in specific cases. It appeared that using different forms of sensor data to provide to the agent and calculate the reward function was detrimental to the RL process.

6.3 Agent Application on Real Robot

As the package promotes the usage of both a simulation and a physical robot, and portability of developed code between the two platforms, a RL agent trained in a simulated environment during a lesson should be applicable to both a simulated robot and a real robot. This section describes the application of the agent generated in section 6.2 to the physical robot in use, and subsequent results and improvements.

6.3.1 Direct Port Experiment

The RL framework was prepared to run with a simple real robot and the input and output ROS topics of the robot were modified to match the framework's. The agent was executed in an environment with an obstacle in front and slightly to the left of the robot. The robot displayed a movement similar to was what observed in the simulation, however it failed to avoid the obstacle. In a second attempt, the robot was placed in an environment without obstacles, to serve as a control in a comparison with the first. No noticeable differences were found between the two experiments.

Upon analysis of the robot's behavior and the RL framework's parameters, several factors were found that could be causing the aforementioned issues:

- Both the virtual sensor used by the real robot and the sensor used in the simulation were assigned a maximum range of 5 meters, despite the precision of the virtual sensor decaying rapidly past 1.5 meters. This could cause the agent to expect data that the sensor cannot provide, hindering calculations.
- The sensor used in the simulation to collect the state data subsequently used for reward calculation was assigned a range of 10 meters, which may be excessive and pollute the reward values with inconsequential information.
- The action space of the robot was too limited, causing the robot to have a large turning radius. This may hinder the agent's ability to avoid obstacles while moving forward.

6.3.2 Improvements and Results

In an effort to obtain improved results, the issues described in the previous section were addressed, by reducing the maximum range of the sensors used by the model to 2 meters, reducing the maximum range of the state collection sensor to 5 meters, and increasing the maximum angular velocity of the agent from 0.5 radians per second to 1.5 radians per second.

Upon training an agent with the new parameters, the simulated testing demonstrated the robot avoiding most obstacles, despite seeming to wander randomly when it detected nothing. The agent was subsequently tested on the real robot and avoided some obstacles, moving mostly as expected. The robot hit obstacles during some experiment, which is assumed to be due to the size of the testing area and lighting issues. Videos[3] and rosbags[4] of the experiments were made available as well.

The virtual sensor has shown a decay in performance in low light conditions, and therefore the agent's decisions could be based on detecting no clear obstacles. Further improvement of the virtual sensors, as well as development of new ones would be recommended.

The reduced size of the testing area, approximately a 2 meters by 3 meters rectangle, may have led the agent to believe that there were obstacles present in every direction at all times. Despite the floor presenting irregularities, prior tests during the development of the virtual sensors indicated that these were being filtered out during the image processing phase. Testing in a larger space would be a valuable next step in the evaluation of the RL model, however no such environment was available at the time of writing.

6.4 Lesson Development and Analysis

Upon having a functional model with consistent results, a tutorial was conceived to guide a user through setting up a ROS workspace with the RL framework, its dependencies, and the Flatland simulator and training an agent[5]. This is meant to provide students with a means to learn how RL functions from a high-level perspective, as well as allow them to alter several parameters of the model and observe the differences in the resulting agent.

This tutorial is expected to satisfy the following outcomes:

1. Prepare a machine to run a reinforcement learning system.
2. Analyse the behavior of a reinforcement learning agent.
3. Modify the learning parameters of a reinforcement learning agent.

In order to evaluate the pertinence of the tutorial, a form was added to the end, for users to provide feedback. All numerical values in the answers are limited to integer values between 1 and 7 and the form used is included in appendix B. The responses for this guide presented a sample size of 9, with an even distribution in terms of experience in robotics (M = 3.89, SD = 2.28), however with the majority having relatively little experience in AI (M = 3.00, SD = 1.83).

The questions relative to the learning outcomes described previously yielded positive results, however somewhat lower than those obtained for the introductions to ROS1 and ROS2, as shown in table 6.2. A visual representation is also shown in figure 6.3. These differences could be due to

[3] https://youtube.com/playlist?list=PL7CIfqtisIW1GHEnlYJhxHGpzSaVv45zY
[4] https://github.com/BerserkingIdiot/robot_control_utils_ros1/tree/main/bags
[5] https://github.com/BerserkingIdiot/drl_local_planner_ros_stable_baselines

this lesson being more advanced, requiring some knowledge of ROS, and is meant to be placed at a later stage in the lesson plan.

The first learning outcome presented the lowest values of the three, indicating that focus should be placed in either simplifying the setup process or making it friendlier and more understandable to users.This value is especially low in the feedback from more experienced users, which may be due to the complexity of the setup compared to the simplicity of installing regular ROS packages.

Lastly, standard deviation values were high, which may have been caused due to some users having issues with dependencies and compatibility during the setup process. A complete transcript of the responses received is included in appendix C.

Table 6.2: RL Learning Outcome Results

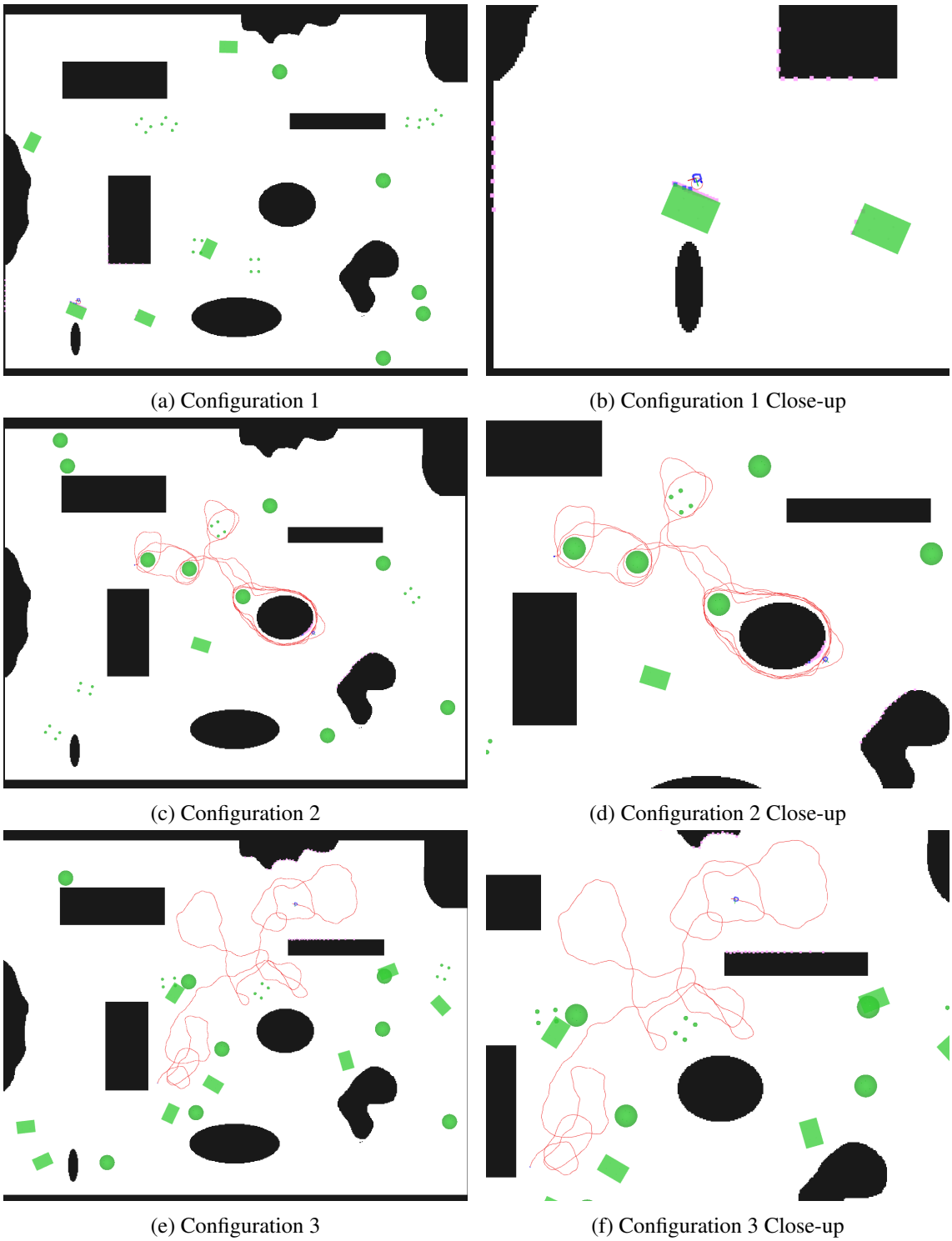
Proficiency	Count	Value	Learning Outcome		
			1	2	3
Overall	9	Mean	4.67	5.22	5.11
		Standard Deviation	1.63	1.55	1.79
Beginner	5	Mean	5.20	5.20	5.00
		Standard Deviation	1.47	1.72	2.10
Expert	4	Mean	4.00	5.25	5.25
		Standard Deviation	1.58	1.30	1.30

6.5 Summary

Concerning the development of a simple reactive agent trained through RL, the adaptations made to the original framework and subsequent tuning proved successful. Although parameters could have been tuned further to optimize the learning process, this would lessen the options for students to experiment, and so the values were kept at a point where they can be both improved and worsened.

Regarding the introductory lesson to RL, results were encouraging, as the proposed learning outcomes were mostly achieved. However, there was also feedback relative to improvements to be made, both to the RL framework and to the lesson itself. Nonetheless, the method has proved itself to be applicable in real lessons.

Applying the agent trained in a simulated environment to the real robot proved fruitful, and this can be used in lessons as an example of the process of robotics development with AI, both by comparing the simulated training environment and the real environment, and by demonstrating the portability of developed code and how to ensure it.



(a) Configuration 1

(b) Configuration 1 Close-up

(c) Configuration 2

(d) Configuration 2 Close-up

(e) Configuration 3

(f) Configuration 3 Close-up

Figure 6.2: Reinforcement Learning Agent Trajectories

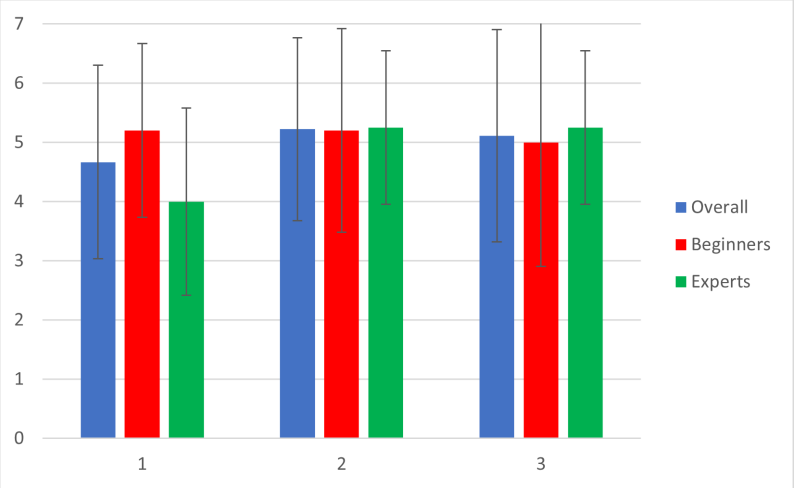


Figure 6.3: RL Learning Outcome Results

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This project set out to design an educational package for teaching ROS-based robotics to students from various backgrounds and with little to no prior knowledge of robotics. As the scope of developing the entire package was too broad, focus was placed on creating a foundation, providing the basis for each component of the package. In its current state, this goal was achieved, as a real robot with accompanying simulation was put together, and introductory lessons to robotics and the application of AI to robotics were constructed. Not only does the work developed serve as a base for future development, it can already be added to existing robotics courses and used by newcomers to robotics.

In response to research question 1, a low-cost, replicable and open-source, both in hardware and software, robot platform for education has proven itself to be feasible. At the end of this project, a functional, simple robot and its respective simulation were achieved, and both the introductory samples and the RL agent could operate within.

Despite not being able to answer research question 2 directly as only a few sample lessons were developed, the feedback received from testing these with users from various backgrounds and with various degrees of proficiency in robotics indicates that the lesson plan designed would be effective in introducing students to robotics. As the lessons are structured for users with no experience in robotics and beginner-level knowledge of programming (common programming languages are used, and alternatives are provided), it is theorized that the educational package in a more complete form could indeed be applied at early stages of university programs.

In an effort to respond to research question 3, this work adopted the approach of providing learners with an existing framework with which they can experiment and explore RL applied to robotics. Despite the framework's shortcomings, some success was achieved, however further research and development on the topic should be conducted, some examples of which are described in section 7.2.

7.2 Future Work

7.2.1 Simulation

In future development stages, more simulation assets could be developed, namely accurate representations of real world environments like classrooms and labs, and additional sensor plugins, to allow the usage of a wider array of sensors to be taught in courses. Additionally, automating the generation of RViz configurations for visualization of the environment or adding representation of sensor data to Flatland's native visualization should be considered, as the development of visualization tools for simulated environments is not part of the educational package's goals.

7.2.2 Physical Robot Platform

Despite satisfying the requirements set for the package, the robot used possessed a few shortcomings that should be improved in future implementations.

The Arduino motor controller could be improved to make better use of the motor encoders and provide more accurate commands to the motors. Moving the conversion of Twist commands to individual motor speeds to this controller should be considered, removing the need for the intermediate differential drive node and simplifying the entire system. Motor assemblies with a lower gear reduction ratio should be considered as well, as the robot was limited to a relatively slow maximum speed.

Although the virtual sensors developed throughout this project performed satisfactorily, there were inaccuracies in low light conditions, as evidenced in the comparison shown in section 5.2.2, and so improvements should be made to provide better reliability in adverse conditions. Additionally, more virtual sensors could be developed, to provide a more complete learning experience, in tandem with the additional sensor plugins suggested for the simulation.

As an alternative to the virtual sensors, more real sensors could be added to the robot. However, this would increase the cost of the package, and so these should be added only as an option, rather than a necessity, and possibly at the discretion of the end user. Nonetheless, the 3D printed bases of the robot could be edited to contain mounting points for additional sensors with little increase to the cost.

Finally, a ROS2 version of the real robot code should be developed, as all of the current work is built in ROS1 only. Although compatibility with ROS2 controllers can still be achieved through the use of the rosbridge package, a full ROS2 port is still recommended.

7.2.3 Reinforcement Learning

The RL approach developed seemed to be successful, however testing environments were considerably limited during the testing of the agent with the real robot. In future development, more tests should be conducted in larger environments, with more obstacles and varying light conditions. The use of different sensors as input, including raw image data from the camera, should be

considered, as it could provide valuable insight to students on the effects of different types of data in the training of an agent.

Despite the framework in use functioning well, the setup procedure has shown itself to be taxing due to legacy dependencies and deprecated functionalities therein, so remaking it with updated dependencies and trimming out unnecessary elements should be considered, both to lighten the load during the training process and to simplify the lesson.

Additionally, a ROS2 version of the RL system should be considered, as the current work is developed purely in ROS1. Similarly to the real robot, compatibility can be achieved through the rosbridge package, but this is not the goal of the project.

7.2.4 Lesson Plan

As the development of the package focused mainly on the tools meant to be used in exercises from the lesson plan, future work should focus on designing a complete lesson plan, exploring all the essential topics of ROS and the differences between both versions, using the introductory tutorials described in section 5.1.2 as a first exercise and the RL lesson described in section 6.4 as an introduction to RL in robotics, closer to the end of the course.

References

[1] H. Levent Akin, Çetin Meriçli, and Tekin Meriçli. Introduction to autonomous mobile robotics using lego mindstorms nxt. *Computer Science Education*, 23:4:368–386, 2013.

[2] Márcia Alves, Armando Sousa, and Ângela Cardoso. Web based robotic simulator for tactode tangible block programming system. *Advances in Intelligent Systems and Computing*, 1092 AISC:490–501, November 2019.

[3] Daniel Amo, Paul Fox, David Fonseca, and César Poyatos. Systematic review on which analytics and learning methodologies are applied in primary and secondary education in the learning of robotics sensors. *Sensors*, 21:153, December 2020.

[4] Francisco Bellas, Alma Mallo, Martin Naya, Daniel Souto, Alvaro Deibe, Abraham Prieto, and Richard J. Duro. Steam approach to autonomous robotics curriculum for high school using the robobo robot. *Advances in Intelligent Systems and Computing*, 1023:77–89, April 2019.

[5] Francisco Bellas, Martin Naya, Gervasio Varela, Luis Llamas, Moises Bautista, Abraham Prieto, and Richard J. Duro. Robobo: The next generation of educational robot. *Advances in Intelligent Systems and Computing*, 694:359–369, November 2017.

[6] Francisco Bellas, Martin Naya, Gervasio Varela, Luis Llamas, Abraham Prieto, Juan Carlos Becerra, Moises Bautista, Andres Faiña, and Richard Duro. The robobo project: Bringing educational robotics closer to real-world applications. *Advances in Intelligent Systems and Computing*, 630:226–237, 2017.

[7] Robert Bogue. Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot*, 43:583–587, October 2016.

[8] João Braun, Lucas A. Fernandes, Thiago Moya, Vitor Oliveira, Thadeu Brito, José Lima, and Paulo Costa. Robot@factory lite: An educational approach for the competition with simulated and real environment. *Advances in Intelligent Systems and Computing*, 1092 AISC:478–489, November 2019.

[9] Maxime Chevalier-Boisvert, Florian Golemo, Yanjun Cao, Bhairav Mehta, Andrea Censi, and Liam Paull. Duckietown environments for openai gym. https://github.com/duckietown/gym-duckietown, 2018. Acessed on 20 of July of 2022.

[10] Avidbots Corp. Welcome to flatland's documentation! — flatland documentation. https://flatland-simulator.readthedocs.io/en/latest/, 2017. Acessed on 20 of July of 2022.

[11] DeepMind. Mujoco — advanced physics simulation. https://mujoco.org/, 2022. Acessed on 20 of July of 2022.

[12] Waveshare Electronics. Alphabot2 - waveshare wiki. https://www.waveshare.com/wiki/AlphaBot2, 2018. Acessed on 20 of July of 2022.

[13] RoboCup Federation. Robocup federation official website. https://www.robocup.org/, 2022. Acessed on 20 of July of 2022.

[14] Duckietown Foundation. Duckiebot operation manual. https://docs.duckietown.org/daffy/opmanual_duckiebot/out/index.html, 2021. Acessed on 20 of July of 2022.

[15] Patrick Goebel. *ROS By Example*, volume 1. Lulu, April 2013.

[16] Patrick Goebel. *ROS By Example*, volume 2. Lulu, April 2014.

[17] The LEGO Group. Lego® mindstorms®| about | official lego® shop us. https://www.lego.com/en-us/themes/mindstorms/about, 2022. Acessed on 20 of July of 2022.

[18] Ronja Güldenring. Applying deep reinforcement learning in the navigation of mobile robots in static and dynamic environments. Master's thesis, Hamburg University, April 2019.

[19] Valentin Haak, Joerg Abke, and Kai Borgeest. Work-in-progress: Development of a lego mindstorms ev3 simulation for programming in c. *Advances in Intelligent Systems and Computing*, 917:667–674, September 2018.

[20] Georgios Karalekas, Stavros Vologiannidis, and John Kalomiros. Europa: A case study for teaching sensors, data acquisition and robotics via a ros-based educational robot. *Sensors*, 20:2469, April 2020.

[21] Cyberbotics Ltd. Webots: robot simulator. https://cyberbotics.com/, 2022. Acessed on 20 of July of 2022.

[22] Jesús López-Belmonte, Adrián Segura-Robles, Antonio-José Moreno-Guerrero, and María-Elena Parra-González. Robotics in education: A scientific mapping of the literature in web of science. *Electronics*, 10:291, January 2021.

[23] Somayya Madakam, Rajesh M. Holmukhe, and Durgesh Kumar Jaiswal. The future digital work force: Robotic process automation (rpa). *JISTEM - Journal of Information Systems and Technology Management*, 16:1–17, January 2019.

[24] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. *Proceedings of the 13th International Conference on Embedded Software, EMSOFT 2016*, October 2016.

[25] NASA. Mars exploration rovers - nasa mars. https://mars.nasa.gov/mer/, 2022. Acessed on 20 of July of 2022.

[26] Wyatt S. Newman. *A Systematic Approach to Learning Robot Programming with ROS*. Chapman and Hall/CRC, September 2017.

[27] Jason M O'Kane. A gentle introduction to ros, 2016.

[28] Ana Rafael, Cássio Santos, Diogo Duque, Sara Fernandes, Armando Sousa, and Luís Paulo Reis. Development of an alphabot2 simulator for rpi camera and infrared sensors. *Advances in Intelligent Systems and Computing*, 1092 AISC:502–514, November 2019.

[29] Open Robotics. Distributions - ros wiki. http://wiki.ros.org/Distributions, 2021. Acessed on 20 of July of 2022.

[30] Open Robotics. Ros/tutorials - ros wiki. https://wiki.ros.org/ROS/Tutorials, 2021. Acessed on 20 of July of 2022.

[31] Open Robotics. Gazebo. http://gazebosim.org/, 2022. Acessed on 20 of July of 2022.

[32] Open Robotics. Project governance — ros 2 documentation: Rolling documentation. https://docs.ros.org/en/rolling/Governance.html, 2022. Acessed on 20 of July of 2022.

[33] Open Robotics. Ros 2 documentation — ros 2 documentation: Galactic documentation. http://docs.ros.org/en/galactic/index.html, 2022. Acessed on 20 of July of 2022.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *Computing Research Repository (CoRR)*, vol. abs/1707.06347, 2017.

[35] SONY. aibo. https://us.aibo.com/, 2021. Acessed on 20 of July of 2022.

[36] George Stavrinos. Ros2 for ros1 users. *Studies in Computational Intelligence*, 895:31–42, 2021.

[37] Jacopo Tani, Liam Paull, Maria T. Zuber, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. Duckietown: An innovative way to teach autonomy. *Advances in Intelligent Systems and Computing*, 560:104–121, November 2016.

[38] Ricardo Tellez. A thousand robots for each student: Using cloud robot simulations to teach robotics. *Advances in Intelligent Systems and Computing*, 457:143–155, 2017.

[39] Jim Torresen. A review of future and ethical perspectives of robotics and ai. *Frontiers in Robotics and AI*, 4, January 2018.

[40] Elisa Tosello, Nicola Castaman, Stefano Michieletto, and Emanuele Menegatti. Teaching robot programming for industry 4.0. *Advances in Intelligent Systems and Computing*, 946 AISC:107–119, October 2018.

[41] Manos Tsardoulias, Chris Zalidis, and Aris Thallas. stdr_simulator - ros wiki. http://wiki.ros.org/stdr_simulator, 2014. Acessed on 20 of July of 2022.

[42] Sokratis Tselegkaridis and Theodosios Sapounidis. Simulators in educational robotics: A review. *Education Sciences*, 11:11, January 2021.

[43] José Varela-Aldás, Oswaldo Miranda-Quintana, Cesar Guevara, Franklin Castillo, and Guillermo Palacios-Navarro. Educational robot using lego mindstorms and mobile device. *Advances in Intelligent Systems and Computing*, 1078:71–82, October 2019.

[44] Julio Vega and José Cañas. Pibot: An open low-cost robotic platform with camera for stem education. *Electronics*, 7:430, December 2018.

[45] Julio Vega and José M. Cañas. Pybokids: An innovative python-based educational framework using real and simulated arduino robots. *Electronics*, 8:899, August 2019.

[46] Sandra Wachter, Brent Mittelstadt, and Luciano Floridi. Transparent, explainable, and accountable ai for robotics. *Science Robotics*, 2, May 2017.

[47] Chris Welch. The duckumentary - the first edition of duckietown (at mit, in 2016) on vimeo. https://vimeo.com/231688769, 2018. Acessed on 20 of July of 2022.

Appendix A

Introductory Tutorials

A.1 ROS1 Introduction

ROS1 Robot Control Crash Course, with Flatland

Installation and Environment Setup

Note: This guide assumes you are using an Ubuntu 20.04 operating system, and can be followed using either a computer running on this OS, WSL, or a Virtual Machine.

ROS Installation

Follow the instructions in the official ROS installation guide. It is recommended to at least install the desktop version, as some tools that will be used here are already contained within.

In this tutorial it is assumed that your ROS installation is placed in /opt/ros/, under the name noetic (which is the most recent Long Term Support distribution at the time of writing).

From this point on, in each bash terminal where ROS commands are used, the setup.bash script must be sourced (as explained in section 1.5 of the installation guide):

```
source /opt/ros/noetic/setup.bash
```

Instead of having to write this command everytime a new terminal is opened, the command can be written onto the .bashrc script, so that it is run automatically when opening a new terminal:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Workspace Setup

Create your own workspace. Pick where you would like your workspace to be placed (usually in the home directory, a.k.a. *~/*) and create a folder with a name of your choice, for example *ros_workspace*, and within it create a folder named *src*. Finally, from within the workspace folder, in this case the ros_workspace folder, run the *catkin_make* command. You can use the following commands in a terminal for this process, assuming you place your workspace in your home directory:

```
mkdir -p ~/ros_workspace/src
cd ~/ros_workspace/
catkin_make
```

Two new folders should have appeared in your workspace, *build/* and *devel/*. Within the *devel/* folder, you'll find multiple shell scripts. One of these, *devel/setup.bash* will be used to allow you to run code from packages within your workspace. After you run *catkin_make*, don't forget to execute the following command:

```
source devel/setup.bash
```

Important Note: Every time you open a new terminal to run ROS code from within your workspace, you must first source ROS's *setup.bash* script and your workspace's own *setup.bash*. The command to source ROS's *setup.bash* is already being automatically run, if you added it to *~/.bashrc*, as indicated in the *ROS Installation* section. In order to avoid having to source the setup.bash of the workspace everytime a new terminal is opened, similarly to what was done for sourcing ROS's setup.bash, the source command can be added to the .bashrc file:

```
echo "source ~/ros_workspace/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Flatland Setup

First, clone the flatland repository into the *src* folder of your workspace and install its dependencies (you may need to install LUA separately if you don't have it already). Then, from within your workspace, build the package with *catkin_make* (and source devel/setup.bash), and launch the default server to test whether everything is working. You can do this process with the following commands:

```
cd src/
git clone https://github.com/avidbots/flatland.git
cd ..
sudo apt-get update
```

```
sudo apt-get install liblua5.1-0-dev
sudo apt-get install ros-noetic-map-server
rosdep install --from-paths src --ignore-src
catkin_make
source devel/setup.bash
roslaunch flatland_server server.launch
```

If all went well, you should see a map of an office and a few models placed around, one of them moving back and forth on a cycle.

First Run

Now that you have ROS and Flatland set up, you can start experimenting with the code from this tutorial. First clone the repository and build it:

```
cd src/
git clone https://github.com/BerserkingIdiot
/flatland_quick_start_ros1.git
cd ..
catkin_make
```

You can launch the example Flatland's native visualization, or RViz, both allowing you to visualize the simulation environment and the models. Flatland's native visualization adapts to simulation environments automatically, and provides some tools to move, spawn and delete models, and pausing the simulation. To launch this version you can use:

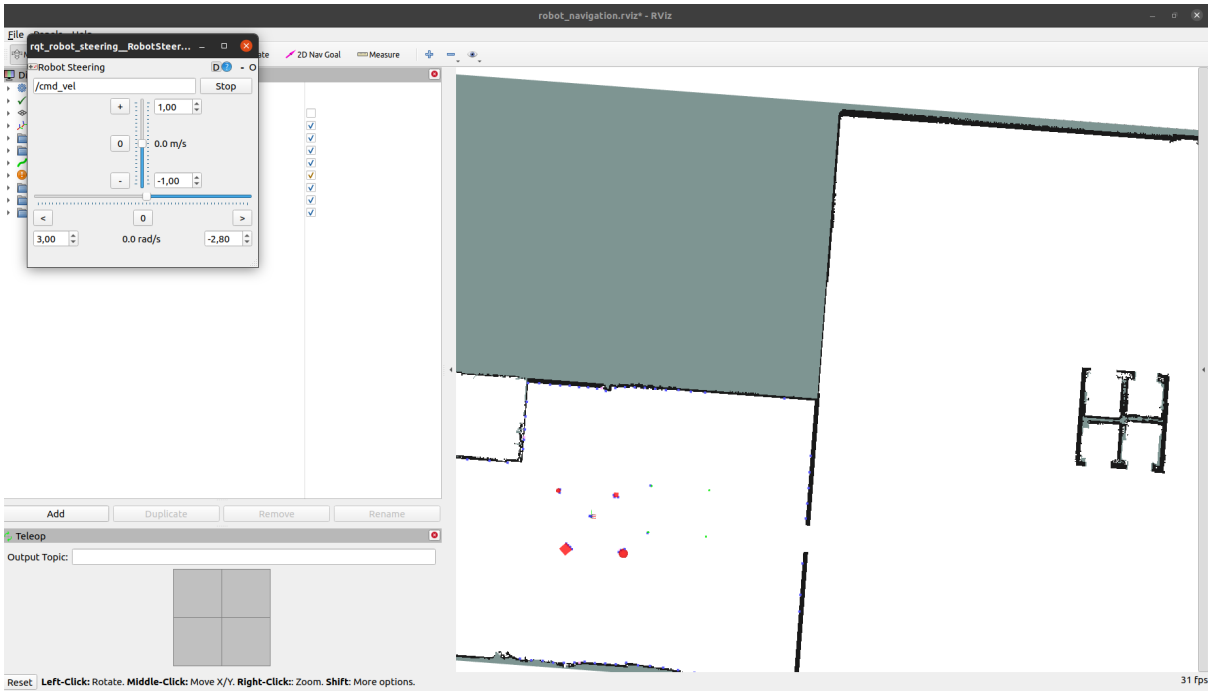
```
roslaunch flatland_quick_start_ros1 flatland_fviz.launch
```

RViz is more complex and requires manual setup for each new environment, but is also more flexible, in this case providing a visualization of the data generated by the LiDAR. To launch this version you can use:

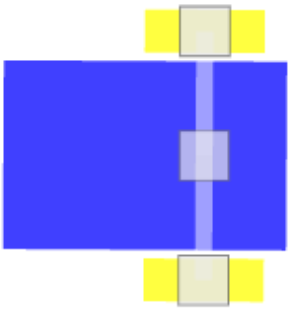
```
roslaunch flatland_quick_start_ros1 flatland_rviz.launch
```

Both of these will launch a Flatland simulation, containing a small differential drive robot equipped with a LiDAR, and a few obstacles. Along with it, *rqt_robot_steering*, a tool that allows you to control the robot's linear and angular velocity, is also launched. Both visualization tools can be manipulated with the mouse, allowing you to drag, rotate and zoom in.

If all went well, you should see something like this (this is the RViz version, however Flatland's native visualization should look similar):



You can zoom in the visualization window with the scroll wheel and move it around by using alt + left click and dragging. The front of the robot starts pointed towards the right and the robot looks like this:



Write your own code

The simulation in this package provides a small differential drive robot equipped with a LiDAR. The differential drive is controlled through Twist messages, and the LiDAR provides data in the form of LaserScan messages. In the *src/* folder of this package, you will find some sample code that you can use to help develop your own robot controller.

Note: the LiDAR scan data is provided in the form of an array of ranges, each value corresponding to the nearest detected obstacle by that ray, or *nan* if nothing is found. The rays are defined counter-clockwise, starting from the rear of the robot, meaning that in the first half of the array are the values for obstacles on the right and in the second half are values for obstacles on the left, with the middle of the array corresponding to the front of the robot. Playing around with the RViz visualization may help understanding how these work.

Three examples are provided in comments on the C++ and Python controllers. The GIFS below provide an example in RViz of the robot moving using each controller example.

Example 1: A robot that moves forward

Example 2: A robot that moves in a circle

Example 3: A robot that moves in circles, and turns around itself when it finds an obstacle in front of it

C++

Within the *src/* folder of this package (not to be confused with the workspace's *src/*), you can find a file named *custom_robot_controller.cpp*. The path to this .cpp file is */ros_workspace/src/flatland_quick_start_ros1/src/custom_robot_controller.cpp*.

You can use this code to write your own code, and experiment with robot control. Don't forget to build the package after you modify the code:

```
cd ~/ros_workspace/src/flatland_quick_start_ros1/src
nano custom_robot_controller.cpp
<edit your code>
cd ~/ros_workspace/
```

```
catkin_make
```

To run the controller you must first make sure to have a simulation running. You can use one of the launch files from the previous section, for example, and then run the controller (with rosrun). For instance,the following commands can be used in two terminals:

In the first terminal (to start the robot and simulator):

```
roslaunch flatland_quick_start_ros1 flatland_rviz.launch
```

In the second terminal (to launch the controller):

```
rosrun flatland_quick_start_ros1 custom_robot_controller
```

Python

Within the *src/* folder of this package (not to be confused with the workspace's *src/*), you can find a file named *custom_robot_controller.py*. The path to this .py file is */ros_workspace/src/flatland_quick_start_ros1/src/custom_robot_controller.py*.

You can use this code to write your own code, and experiment with robot control. Unlike the C++ version, you don't have to build the package every time you modify the code, but before you run it you must make the script executable. You can do this by running this command from your workspace:

```
chmod +x src/flatland_quick_start_ros1/src/custom_robot_controller.py
```

To run the controller you must first make sure to have a simulation running. You can use one of the launch files from the previous section, for example, and then run the controller (with rosrun). For instance,the following commands can be used in two terminals:

In the first terminal (to start the robot and simulator):

```
roslaunch flatland_quick_start_ros1 flatland_rviz.launch
```

In the second terminal (to launch the controller):

```
rosrun flatland_quick_start_ros1 custom_robot_controller.py
```

Next Steps

Topics, Publishers and Subscribers

In this tutorial, you were provided with a ROS node with the communication features already set up, and you might have noticed elements like the LiDAR scan subscriber and the Twist message publisher. These are essential elements of the ROS environment, and you can learn more about them in the official ROS tutorials.

Exploring Flatland

In this package, there are currently 2 different maps to experiment with, found in the *flatland_worlds/* folder. The launch files default to the office test world, but you can use the world_path parameter to choose the world you wish to boot up, for example to boot up the maze world you can run this:

```
roslaunch flatland_quick_start_ros1 flatland_rviz.launch
world_path:="maze"
```

To learn more about the Flatland simulator, read the documentation and tutorials

Feedback

Once you have finished following the tutorial, please fill out the following form: https://forms.gle/Pw1brvNvAmvSGhQr9

A.2 ROS2 Introduction

ROS2 Robot Control Crash Course, with Flatland

Installation and Environment Setup

Note: This guide assumes you are using an Ubuntu 20.04 operating system, and can be followed using either a computer running on this OS, WSL, or a Virtual Machine.

ROS Installation

Follow the instructions in the official ROS installation guide. It is recommended to at least install the desktop version, as some tools that will be used here are already contained within.

In this tutorial it is assumed that your ROS installation is placed in /opt/ros/, under the name foxy (which is the second most recent Long Term Support distribution at the time of writing).

Note: From this point on, it is assumed that every terminal has sourced the *setup.bash* script as indicated in the Environment Setup section of the installation guide. Without this, you will not be able to use ROS or any of its tools. If you moved the installation to */opt/ros/foxy/*, the command to source it becomes:

```
source /opt/ros/foxy/setup.bash
```

Instead of having to write this command everytime a new terminal is opened, the command can be written onto the .bashrc script, so that it is run automatically when opening a new terminal:

```
echo "source /opt/ros/foxy/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Some additional ROS2 tools and the colcon build system will be required for this tutorial to function properly. You can install them with:

```
sudo apt install ros-foxy-rviz2 ros-foxy-navigation2 ros-foxy-nav2-
bringup
sudo apt install python3-colcon-common-extensions
```

```
sudo apt install python3-rosdep
sudo rosdep init
rosdep update
```

Workspace Setup

Create your own workspace. Pick where you would like your workspace to be placed (usually in the home directory, a.k.a. *~/*) and create a folder with a name of your choice, for example *ros_workspace*, and within it create a folder named *src*. Finally, from within the workspace folder, in this case the ros_workspace folder, run the *colcon build* command. You can use the following commands in a terminal for this process, assuming you place your workspace in your home directory:

```
mkdir -p ~/ros_workspace/src
cd ~/ros_workspace/
colcon build
```

Three new folders should have appeared in your workspace, *build/*, *install/* and *log/*. Within the *install/* folder, you'll find multiple shell scripts. One of these, *install/setup.bash* will be used to allow you to run code from packages within your workspace. After you run *colcon build*, don't forget to execute the following command:

```
source install/setup.bash
```

Important Note: Every time you open a new terminal to run ROS code from within your workspace, you must first source ROS's *setup.bash* script and your workspace's own *setup.bash*. The command to source ROS's *setup.bash* is already being automatically run, if you added it to *~/.bashrc*, as indicated in the *ROS Installation* section. In order to avoid having to source the setup.bash of the workspace everytime a new terminal is opened, similarly to what was done for sourcing ROS's setup.bash, the source command can be added to the .bashrc file:

```
echo "source ~/ros_workspace/install/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Flatland Setup

First, clone the flatland repository into the *src* folder of your workspace, check out the ros2-plugins-port-broken branch and install its dependencies (you may have to install LUA separately if you don't already have it). Then, from within your workspace, build the package with *colcon build* (and source install/setup.bash), and launch the default server to test whether everything is working. You can do this process with the following commands:

```
cd src/
git clone https://github.com/avidbots/flatland.git
cd flatland
git checkout origin/ros2-plugins-port-broken
cd ../..
sudo apt-get update
sudo apt-get install liblua5.1-0-dev
rosdep install --from-paths src --ignore-src
colcon build
source install/setup.bash
```

If all went well, you should see some feedback on the console from the build process.

First Run

Now that you have ROS and Flatland set up, you can start experimenting with the code from this tutorial. First clone the repository and build it:

```
cd src/
git clone https://github.com/BerserkingIdiot
/flatland_quick_start_ros2.git
cd ..
colcon build
source install/setup.bash
```

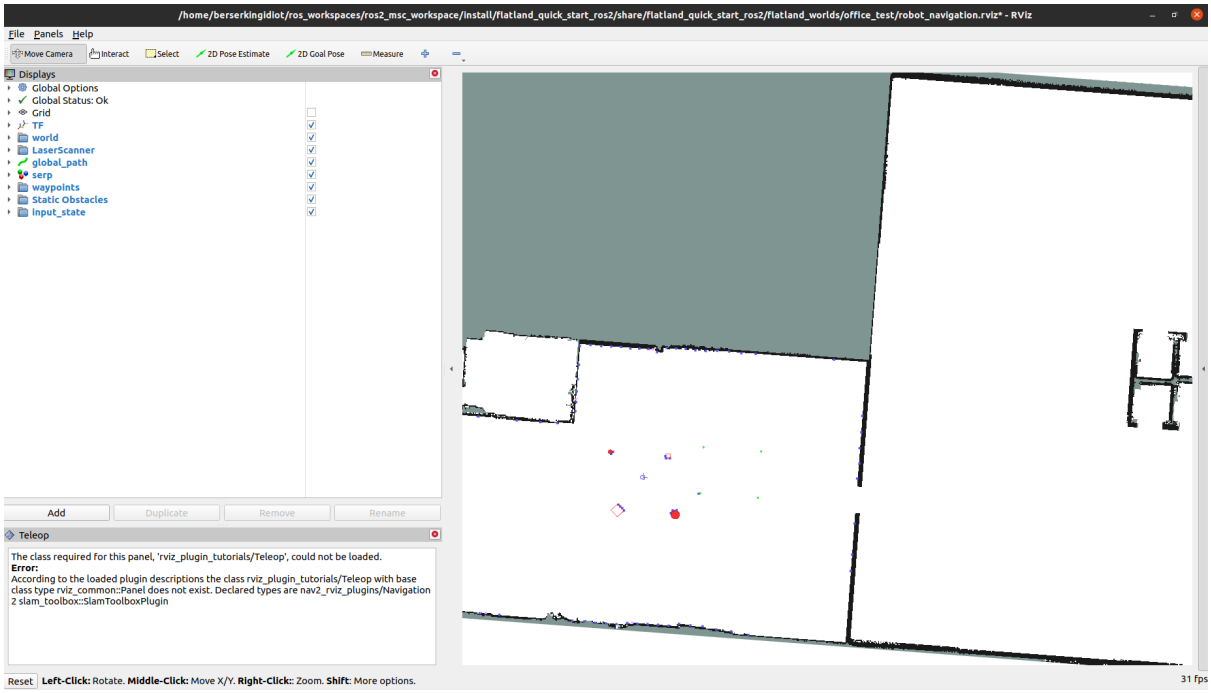
You can launch the example with RViz visualization. RViz is complex and requires manual setup for each new environment, but also flexible, in this case providing a visualization of the data generated by the LiDAR. To launch this you can use:

```
ros2 launch flatland_quick_start_ros2 flatland_rviz.launch.xml
```

Once the simulation is running, in a separate terminal, run:

```
ros2 run nav2_util lifecycle_bringup map_server
```

This will launch a Flatland simulation, containing a small differential drive robot equipped with a LiDAR, and a few obstacles. The RViz visualization can be manipulated with the mouse, allowing you to drag, rotate, zoom and enable and disable certain components of the visualization. The second command allows the map to be visible in RViz by activating the *map_server* node (lifecycles are a new feature in ROS2 that allows for more deterministic code development but we won't delve into it in this tutorial). If all went well, you should see something like this:



You can zoom in the visualization window with the scroll wheel and move it around by using alt + left click and dragging. The front of the robot starts pointed towards the right and the robot looks like this:

Write your own code

The simulation in this package provides a small differential drive robot equipped with a LiDAR. The differential drive is controlled through Twist messages, and the LiDAR provides data in the form of LaserScan messages. In this package, you will find some sample code that you can use to help develop your own robot controller.

Note: the LiDAR scan data is provided in the form of an array of ranges, each value corresponding to the nearest detected obstacle by that ray, or *nan* if nothing is found. The rays are defined counter-clockwise, starting from the rear of the robot, meaning that in the first half of the array are the values for obstacles on the right and in the second half are values for obstacles on the left, with the middle of the array corresponding to the front of the robot. Playing around with the RViz visualization may help understanding how these work.

Three examples are provided in comments on the C++ and Python controllers. The GIFS below provide an example in RViz of the robot moving using each controller example.

Example 1: A robot that moves forward

Example 2: A robot that moves in a circle

Example 3: A robot that moves in circles, and turns around itself when it finds an obstacle in front of it

C++

Within the *src/* folder of this package (not to be confused with the workspace's *src/*), you can find a file named *custom_robot_controller.cpp*. The path to this .cpp file is */ros_workspace/src/flatland_quick_start_ros2/src/custom_robot_controller.cpp*.

You can use this code to write your own code, and experiment with robot control. Don't forget to build the package after you modify the code:

```
cd ~/ros_workspace/src/flatland_quick_start_ros2/src
nano custom_robot_controller.cpp
<edit your code>
cd ~/ros_workspace/
colcon build
```

To run the controller you must first make sure to have a simulation running. You can use one of the launch files from the previous section, for example, and then run the controller (with rosrun). For instance,the following commands can be used in two terminals:

In the first terminal (to start the robot and simulator):

```
ros2 launch flatland_quick_start_ros2 flatland_rviz.launch.xml
```

In the second terminal (to launch the controller):

```
ros2 run flatland_quick_start_ros2 custom_robot_controller
```

Python

Within the *scripts/* folder of this package, you can find a file named *custom_robot_controller.py*. The path to this .py file is */ros_workspace /src/flatland_quick_start_ros2/scripts/custom_robot_controller.py*.

You can use this code to write your own code, and experiment with robot control. Don't forget to build the package after you modify the code:

```
cd ~/ros_workspace/src/flatland_quick_start_ros2/scripts
nano custom_robot_controller.py
<edit your code>
cd ~/ros_workspace/
colcon build
```

To run the controller you must first make sure to have a simulation running. You can use one of the launch files from the previous section, for example, and then run the controller (with rosrun). For instance,the following commands can be used in two terminals:

In the first terminal (to start the robot and simulator):

```
ros2 launch flatland_quick_start_ros2 flatland_rviz.launch.xml
```

In the second terminal (to launch the controller):

```
ros2 run flatland_quick_start_ros2 custom_robot_controller.py
```

Next Steps

Topics, Publishers and Subscribers

In this tutorial, you were provided with a ROS node with the communication features already set up, and you might have noticed elements like the LiDAR scan subscriber and the Twist message publisher. These are essential elements of the ROS environment, and you can learn more about them in the official ROS tutorials.

Exploring Flatland

In this package, there are currently 2 different maps to experiment with, found in the *flatland_worlds/* folder. The launch files default to the office test world, but you can use the world_path parameter to choose the world you wish to boot up, for example to boot up the maze world you can run this:

```
ros2 launch flatland_quick_start_ros2 flatland_rviz.launch.xml
world_path:="maze"
```

To learn more about the Flatland simulator, read the documentation and tutorials

Feedback

Once you have finished following the tutorial, please fill out the following form:
https://forms.gle/Pw1brvNvAmvSGhQr9

A.3 RL Introduction

Introduction

This repository cointains a simplified version of the Reinforcement Learning training environment developed by Ronja Gueldenring (which can be found here).

By following the instructions found in this document, you should be able to train a simple reactive agent to avoid obstacles, and is intended to serve as an introductory lesson to Reinforcement Learning. It is assumed that you have prior experience with ROS and have a workspace set up.

This document is by no means a complete guide on Reinforcement Learning with ROS, but rather a simple first contact, and should be taken as such.

Installation

1. Standard ROS setup (Code has been tested with ROS-noetic on Ubuntu 20.04)
2. Install additional packages

```
sudo apt-get update && sudo apt-get install -y \
libopencv-dev \
liblua5.2-dev \
virtualenv \
screen \
ros-noetic-tf2-geometry-msgs \
ros-noetic-navigation \
ros-noetic-rviz \
python3-rosinstall
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install python3.7-dev -y
sudo apt install python3.7-venv -y
```

3. Setup repository:
    - First, lone this repository in your src-folder of your catkin workspace with:

```
cd <path_to_catkin_ws>/src/
git clone https://github.com/BerserkingIdiot
/drl_local_planner_ros_stable_baselines.git
```

- Then do the following:

```
cd <path_to_catkin_ws>/src/drl_local_planner_ros_stable_baselines
cp .rosinstall ../
cd ..
rosws update
cd <path_to_catkin_ws>
catkin_make -DCMAKE_BUILD_TYPE=Release
```

(please install missing packages)

4. Setup virtual environment to be able to use python3 with ROS

```
 virtualenv <path_to_venv>/venv_p3 --python=python3.7
 source <path_to_venv>/venv_p3/bin/activate
 python -m ensurepip --upgrade
 <path_to_venv>/venv_p3/bin/pip3 install \
     pyyaml \
     rospkg \
     catkin_pkg \
     exception \
     numpy \
     tensorflow=="1.15" \
     pyquaternion \
     mpi4py \
     matplotlib
 <path_to_venv>/venv_p3/bin/pip3 install -r
src/drl_local_planner_ros_stable_baselines/requirements.txt
 <path_to_venv>/venv_p3/bin/pip3 install -e
<path_to_catkin_ws>/src/drl_local_planner_forks/stable-baselines/
```

5. Set system-relevant variables

- Modify all relevant paths in rl_bringup/config/path_config.ini

Training and running your first agent:

In this section, you'll be able to train an agent with the default parameters, and subsequently run your agent to test its capabilities. Notice that even with the exact same parameters, there are slightly different outcomes from separate training runs!

Important Note: All commands from here on are formatted to be run from the root of your workspace folder, and every terminal is expected to have the ROS environment set up (sourced the setup.bash files). For this, do:

```
source /opt/ros/noetic/setup.bash
source devel/setup.bash
```

1. Train agent
    - Open first terminal (roscore):

    ```
    roscore
    ```

    - Open second terminal (simulation):

    ```
    roslaunch rl_bringup setup.launch ns:="sim1"
    rl_params:="rl_params_scan"
    ```

    - Open third terminal (Visualization):

    ```
    roslaunch rl_bringup rviz.launch ns:="sim1"
    ```

    - Open fourth terminal (DRL-agent):

    ```
    source <path_to_venv>/venv_p3/bin/activate
    python src/drl_local_planner_ros_stable_baselines/rl_agent
    /scripts/train_scripts/train_ppo.py
    ```

2. Execute self-trained ppo-agent
    - Open first terminal:

    ```
    roscore
    ```

    - Open second terminal:

    ```
    roslaunch rl_bringup setup.launch ns:="sim1"
    rl_params:="rl_params_scan"
    ```

    - Open third terminal:

    ```
    roslaunch rl_bringup rviz.launch ns:="sim1"
    ```

    - Open fourth terminal:

```
source <path_to_venv>/venv_p3/bin/activate
python src/drl_local_planner_ros_stable_baselines/rl_agent
/scripts/train_scripts/run_ppo.py ppo2_simple_reactive
CNN1DPolicy_multi_input train 1 0 0 static 3
```

Tuning and improving your agent:

By opening the train_ppo.py script (found in rl_agent/scripts/train_scripts/) and scrolling to the end, you can find the default parameters used by the model. Try changing the values of the following and see how it changes the learning process:

- gamma
- n_steps
- ent_coef
- learning_rate
- cliprange
- total_timesteps

Additionally, you can find the reward function being used in the reward_container.py script (found in rl_agent/src/rl_agent/env_utils/), currently in the form of *rew_func_19*. You can change some of the values of the different elements of the reward, such as the punishment for hitting obstacles and the reward for moving away from the starting position, and see how that changes what the agent learns during training.

Feedback

Once you're finished with this introduction, please fill out the following form (select the Reinforcement Learning Introduction option): https://forms.gle/Nfi6TKP4GBzeb3nG9

Appendix B

Feedback Forms

ROS - Flatland Tutorial

*Obrigatório

ROS 1 Quick Start Tutorial

How proficient are you in robotics? *

	1	2	3	4	5	6	7	
No experience	○	○	○	○	○	○	○	Many years of experience

How proficient are you with simulators? *

	1	2	3	4	5	6	7	
No experience	○	○	○	○	○	○	○	Many years of experience

How did this tutorial change your understanding of robotics development? *

	1	2	3	4	5	6	7	
Decreased considerably	○	○	○	○	○	○	○	Increased considerably

How did this tutorial change your interest in the field of robotics? *

	1	2	3	4	5	6	7	
Decreased considerably	○	○	○	○	○	○	○	Increased considerably

How helpful did you find this tutorial in learning about ROS-based robotics? *

	1	2	3	4	5	6	7	
Not helpful at all	○	○	○	○	○	○	○	Extremely helpful

How appropriate do you consider the visualization of the simulation for learning robotics? *

	1	2	3	4	5	6	7	
Completely inappropriate	○	○	○	○	○	○	○	Completely appropriate

How useful did you find this tutorial to fulfill the following goals? *

	1 - Not helpful at all	2	3	4	5	6	7 - Extremely helpful
Prepare a machine to develop basic ROS packages with Flatland	○	○	○	○	○	○	○
Analyse the progress of a virtual robot in 2D simulation with visualization tools	○	○	○	○	○	○	○
Modify the code controlling a virtual robot in 2D simulation	○	○	○	○	○	○	○

Why did you choose this version over ROS 2?

A sua resposta

Please write any additional comments and suggestions below.

A sua resposta

Anterior Enviar Limpar formulário

Nunca envie palavras-passe através dos Google Forms.

ROS - Flatland Tutorial

*Obrigatório

ROS 2 Quick Start Tutorial

How proficient are you in robotics? *

	1	2	3	4	5	6	7	
No experience	○	○	○	○	○	○	○	Many years of experience

How proficient are you with simulators? *

	1	2	3	4	5	6	7	
No experience	○	○	○	○	○	○	○	Many years of experience

How did this tutorial change your understanding of robotics development? *

	1	2	3	4	5	6	7	
Decreased considerably	○	○	○	○	○	○	○	Increased considerably

How did this tutorial change your interest in the field of robotics? *

	1	2	3	4	5	6	7	
Decreased considerably	○	○	○	○	○	○	○	Increased considerably

How helpful did you find this tutorial in learning about ROS-based robotics? *

	1	2	3	4	5	6	7	
Not helpful at all	○	○	○	○	○	○	○	Extremely helpful

How appropriate do you consider the visualization of the simulation for learning robotics? *

	1	2	3	4	5	6	7	
Completely inappropriate	○	○	○	○	○	○	○	Completely appropriate

How useful did you find this tutorial to fulfill the following goals? *

	1 - Not helpful at all	2	3	4	5	6	7 - Extremely helpful
Prepare a machine to develop basic ROS packages with Flatland	○	○	○	○	○	○	○
Analyse the progress of a virtual robot in 2D simulation with visualization tools	○	○	○	○	○	○	○
Modify the code controlling a virtual robot in 2D simulation	○	○	○	○	○	○	○

Did you have any prior experience with ROS 1? *

- ○ Yes
- ○ No

Why did you choose this version over ROS 1?

A sua resposta

Please write any additional comments and suggestions below.

A sua resposta

Anterior Enviar Limpar formulário

ROS - Flatland Tutorial

*Obrigatório

Reinforcement Learning Introduction

How proficient are you in robotics? *

	1	2	3	4	5	6	7	
No experience	○	○	○	○	○	○	○	Many years of experience

How proficient are you with simulators? *

	1	2	3	4	5	6	7	
No experience	○	○	○	○	○	○	○	Many years of experience

How proficient are you with Artifical Intelligence? *

	1	2	3	4	5	6	7	
No experience	○	○	○	○	○	○	○	Many years of experience

How did this tutorial change your understanding of robotics development? *

	1	2	3	4	5	6	7	
Decreased considerably	○	○	○	○	○	○	○	Increased considerably

How did this tutorial change your interest in the field of robotics? *

	1	2	3	4	5	6	7	
Decreased considerably	○	○	○	○	○	○	○	Increased considerably

How did this tutorial change your interest in the field of Artificial Intelligence? *

	1	2	3	4	5	6	7	
Decreased considerably	○	○	○	○	○	○	○	Increased considerably

How helpful did you find this tutorial in learning about Reinforcement Learning? *

	1	2	3	4	5	6	7	
Not helpful at all	○	○	○	○	○	○	○	Extremely helpful

How appropriate do you consider the visualization of the simulation for learning robotics? *

	1	2	3	4	5	6	7	
Completely inappropriate	○	○	○	○	○	○	○	Completely appropriate

How useful did you find this tutorial to fulfill the following goals? *

	1 - Not helpful at all	2	3	4	5	6	7 - Extremely helpful
Prepare a machine to run a reinforcement learning system	○	○	○	○	○	○	○
Analyse the behavior of a reinforcement learning agent	○	○	○	○	○	○	○
Modify the learning parameters of a reinforcement learning agent	○	○	○	○	○	○	○

Please write any additional comments and suggestions below.

A sua resposta

Anterior Enviar Limpar formulário

Appendix C

Form Results

C.1 ROS1 Tutorial

Question key:

1. How proficient are you in robotics?
2. How proficient are you with simulators?
3. How did this tutorial change your understanding of robotics development?
4. How did this tutorial change your interest in the field of robotics?
5. How helpful did you find this tutorial in learning about ROS-based robotics?
6. How appropriate do you consider the visualization of the simulation for learning robotics?
7. How useful did you find this tutorial to fulfill the following goals? [Prepare a machine to develop basic ROS packages with Flatland]
8. How useful did you find this tutorial to fulfill the following goals? [Analyse the progress of a virtual robot in 2D simulation with visualization tools]
9. How useful did you find this tutorial to fulfill the following goals? [Modify the code controlling a virtual robot in 2D simulation]

All questions accept answers as integer values between 1 and 7.
Additional feedback elements:

- Id 1: "A video tutorial would help to make it easier to navigate the instructions, for inexperience people this tutorial would give some confusion on some steps."
- Id 2: "It is not critical, but you could add some links for more information about the tools, commands, etc through the tutorial, for a better understanding of what we are doing, while we are doing it. (Similar suggestion as in ros2)"

Table C.1: ROS1 Tutorial feedback results

Id	1	2	3	4	5	6	7	8	9
1	1	1	2	1	4	3	4	4	5
2	3	2	4	6	5	7	5	6	4
3	7	7	4	4	5	5	5	5	5
4	6	6	4	4	6	6	7	6	6
5	3	3	6	6	6	6	6	6	6
6	1	1	2	5	6	7	5	6	6
7	5	5	5	5	7	7	5	5	5
8	7	6	1	1	5	7	7	6	7
9	5	3	6	5	4	5	5	4	6
10	1	2	4	5	5	7	7	7	5
11	2	4	4	4	5	7	7	7	7

- Id 8: "The tutorial text could be more objective and more formal."
- Id 9: "Can have a better environment to achieve more young age. But i understand that isn't import now. ;-)"
- Id 11: "Better troubleshoot tips and hints for newbies would go a long way"

C.2 ROS2 Tutorial

Question key:

1. How proficient are you in robotics?
2. How proficient are you with simulators?
3. How did this tutorial change your understanding of robotics development?
4. How did this tutorial change your interest in the field of robotics?
5. How helpful did you find this tutorial in learning about ROS-based robotics?
6. How appropriate do you consider the visualization of the simulation for learning robotics?
7. How useful did you find this tutorial to fulfill the following goals? [Prepare a machine to develop basic ROS packages with Flatland]
8. How useful did you find this tutorial to fulfill the following goals? [Analyse the progress of a virtual robot in 2D simulation with visualization tools]
9. How useful did you find this tutorial to fulfill the following goals? [Modify the code controlling a virtual robot in 2D simulation]
10. Did you have any prior experience with ROS 1?

Questions 1 through 9 accept answers as integer values between 1 and 7. Question 10 accepts only "Yes" or "No" answers.

Table C.2: ROS2 Tutorial feedback results

Id	1	2	3	4	5	6	7	8	9	10
1	3	2	4	5	5	7	5	6	4	Yes
2	7	7	4	4	5	5	5	5	5	Yes
3	3	3	6	6	6	6	6	6	6	No
4	1	4	4	1	5	6	6	5	4	No
5	5	5	5	6	6	7	5	5	5	Yes
6	7	6	1	1	5	6	7	7	7	Yes
7	4	5	6	5	5	3	5	5	5	No
8	1	2	4	5	5	7	7	7	5	No
9	2	4	6	4	5	7	7	7	7	No

Additional feedback elements:

- Id 1: "It's nothing critical because you already guide the people doing these tutorials at the end, but you could add some additional links through the tutorial for more information about some commands and tools used. For example, additional information about the ROS file structure and workspace, about the "run" and "launch" commands, about the rviz tool, etc. The perception i got is that these tutorials are for beginners in ROS and flatland, and having a "if you want to see more about this that you will be using" would be a nice to have (not a need to have), mostly for a better understanding of what they are doing."
- Id 4: "as in ros1 a video tutorial would had been a bit better for unexperienced people to use the program"

C.3 RL Tutorial

Question key:

1. How proficient are you in robotics?
2. How proficient are you with simulators?
3. How proficient are you with Artifical Intelligence?
4. How did this tutorial change your understanding of robotics development?
5. How did this tutorial change your interest in the field of robotics?
6. How did this tutorial change your interest in the field of Artificial Intelligence?
7. How helpful did you find this tutorial in learning about Reinforcement Learning?

8. How appropriate do you consider the visualization of the simulation for learning robotics?
9. How useful did you find this tutorial to fulfill the following goals? [Prepare a machine to run a reinforcement learning system]
10. How useful did you find this tutorial to fulfill the following goals? [Analyse the behavior of a reinforcement learning agent]
11. How useful did you find this tutorial to fulfill the following goals? [Modify the learning parameters of a reinforcement learning agent]

All questions accept answers as integer values between 1 and 7.

Table C.3: RL Tutorial feedback results

Id	1	2	3	4	5	6	7	8	9	10	11
1	3	2	2	4	5	6	5	7	6	6	5
2	7	7	5	4	4	5	6	5	5	6	6
3	6	6	3	5	4	5	6	6	2	6	6
4	3	3	3	5	6	6	6	6	6	6	6
5	1	1	1	4	1	1	5	5	4	5	6
6	5	5	7	6	6	7	7	6	3	3	3
7	7	6	3	1	1	4	6	6	6	6	6
8	1	2	2	3	3	3	2	6	3	2	1
9	2	4	1	4	4	4	6	7	7	7	7

Additional feedback elements:

- Id 3: "I got problems trying to compile the workspace with the given instructions"
- Id 6: "I think that the tutorial could be clearer in the preparation of the environment and the training and execution routines."
- Id 8: "It was not easy following the terminal and was unable to execute the self-trained agent, because training always failed. I found problems running the install commands in virtualenv, related to incompatible versions - changed Kera to 1.0.8."
- Id 9: "Better python venv documentation"

Appendix D

Detailed Virtual Sensor Images

D.1 Simulated LiDAR

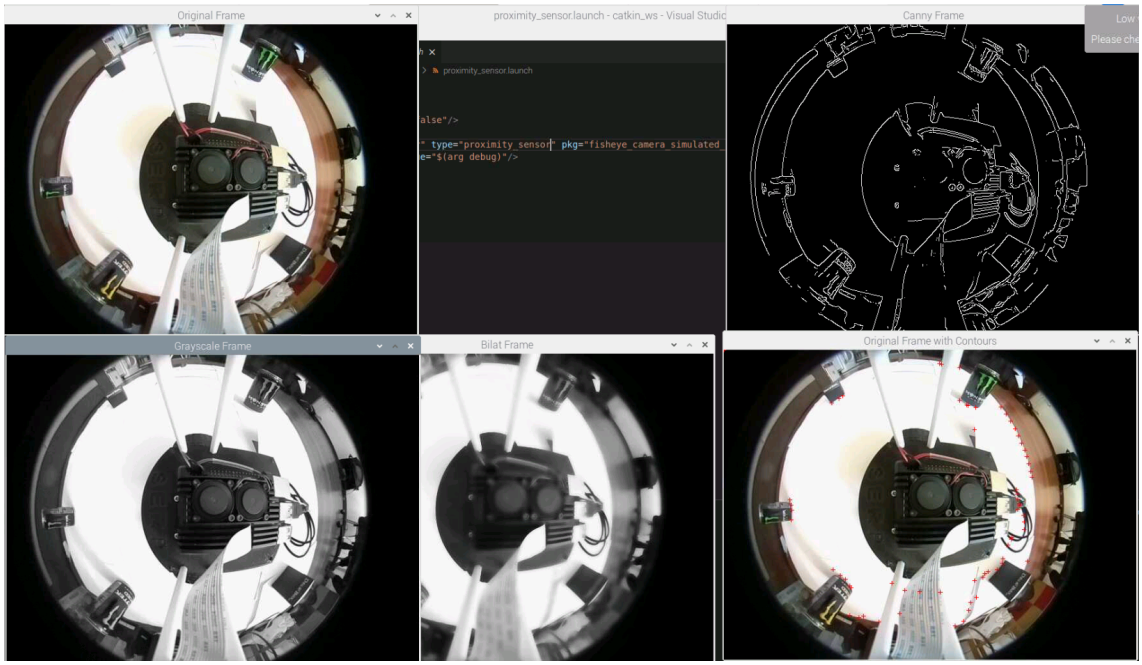


Figure D.1: Simulated LiDAR in well lit conditions

D.2 Object Detection

D.3 Proximity Sensor

D.3.1 Calibration

This section contains the approximate curve for each ray of the proximity sensor.

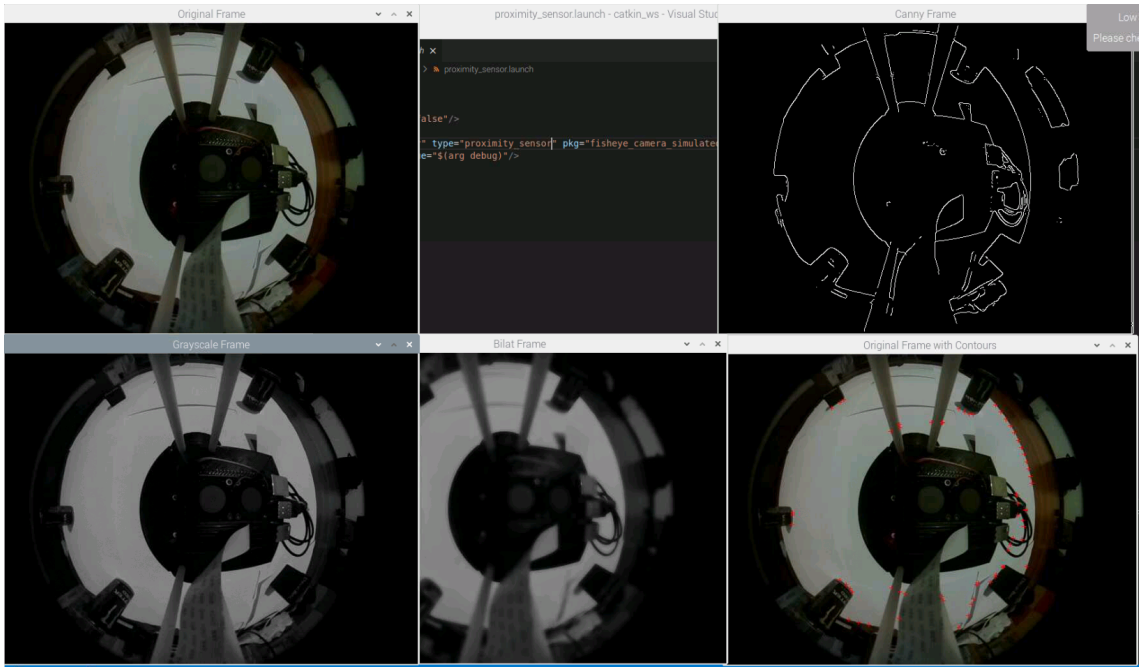


Figure D.2: Simulated LiDAR in low light conditions

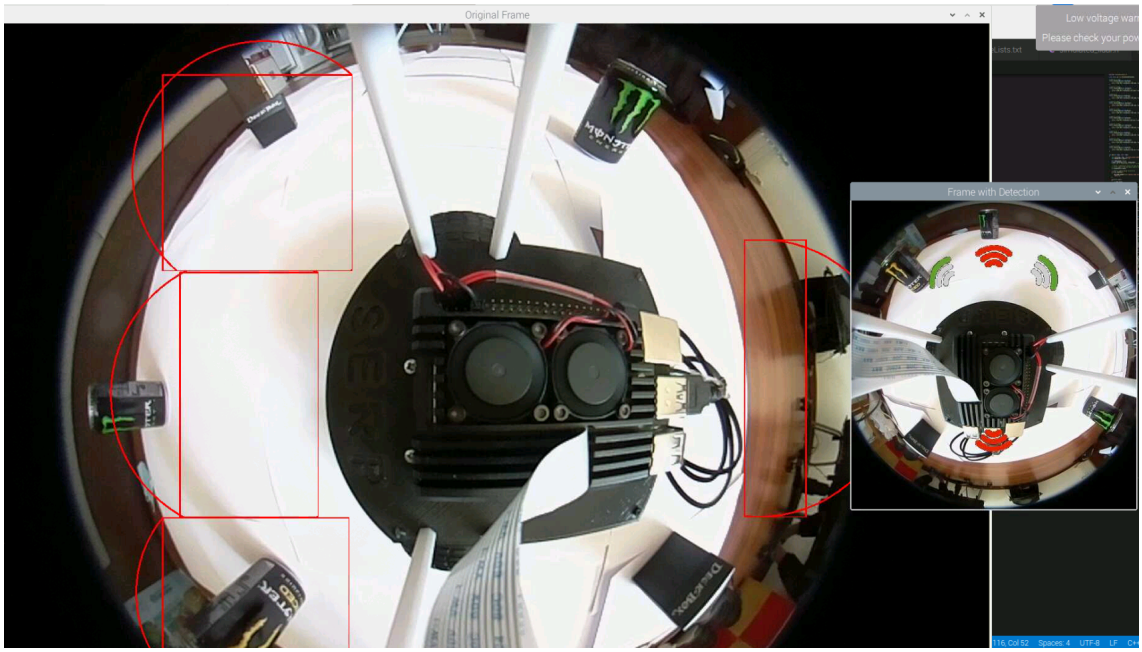


Figure D.3: Object Detection in well lit conditions

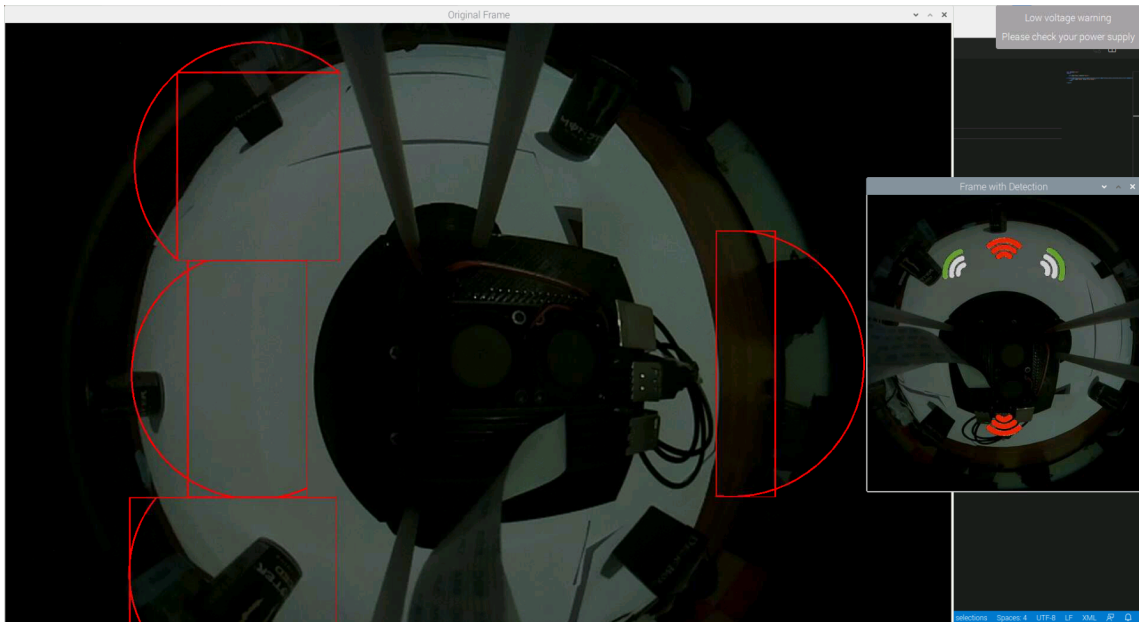


Figure D.4: Object Detection in low light conditions

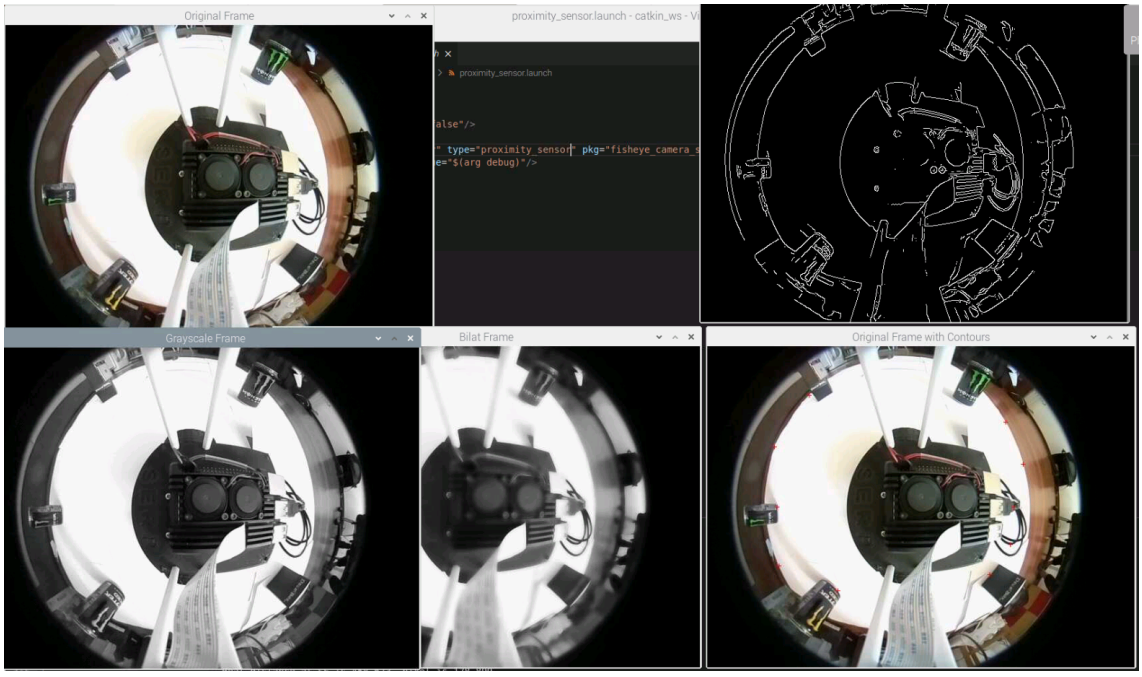


Figure D.5: Proximity Sensor in well lit conditions

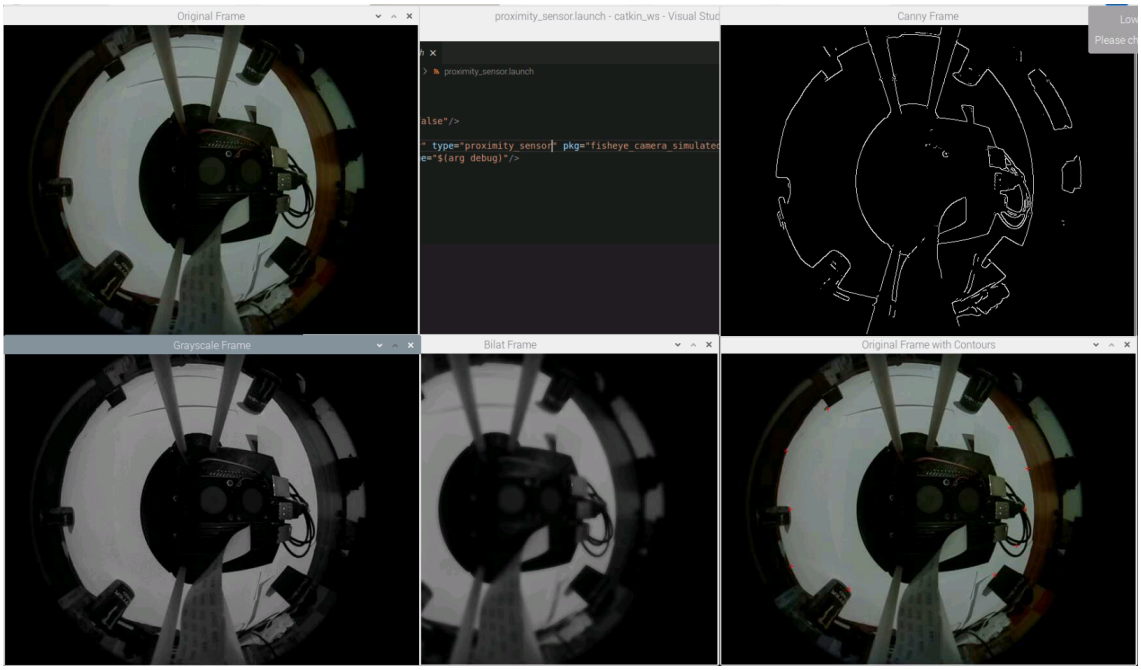


Figure D.6: Proximity Sensor in low light conditions

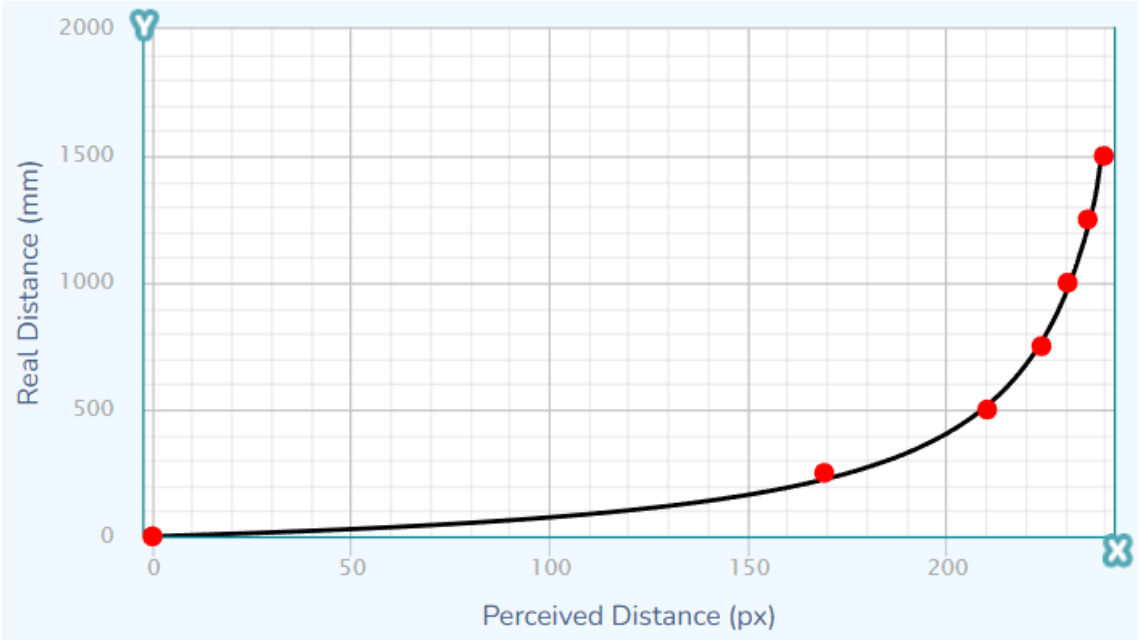


Figure D.7: Ray at $-\frac{1}{4}\pi$

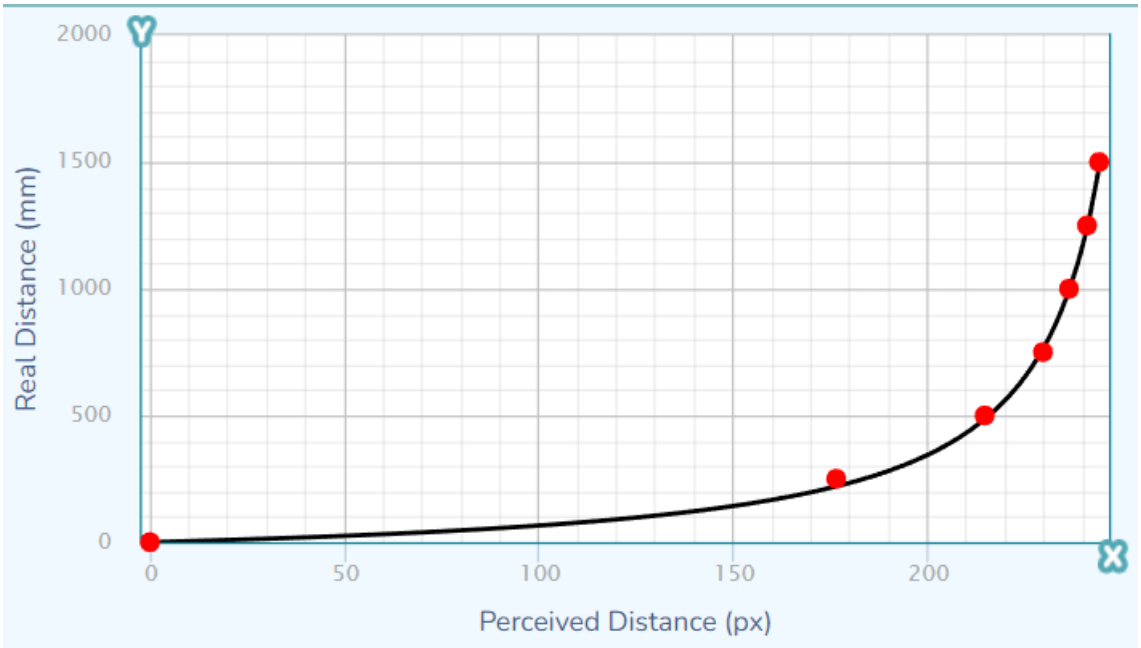


Figure D.8: Ray at $-\frac{1}{8}\pi$

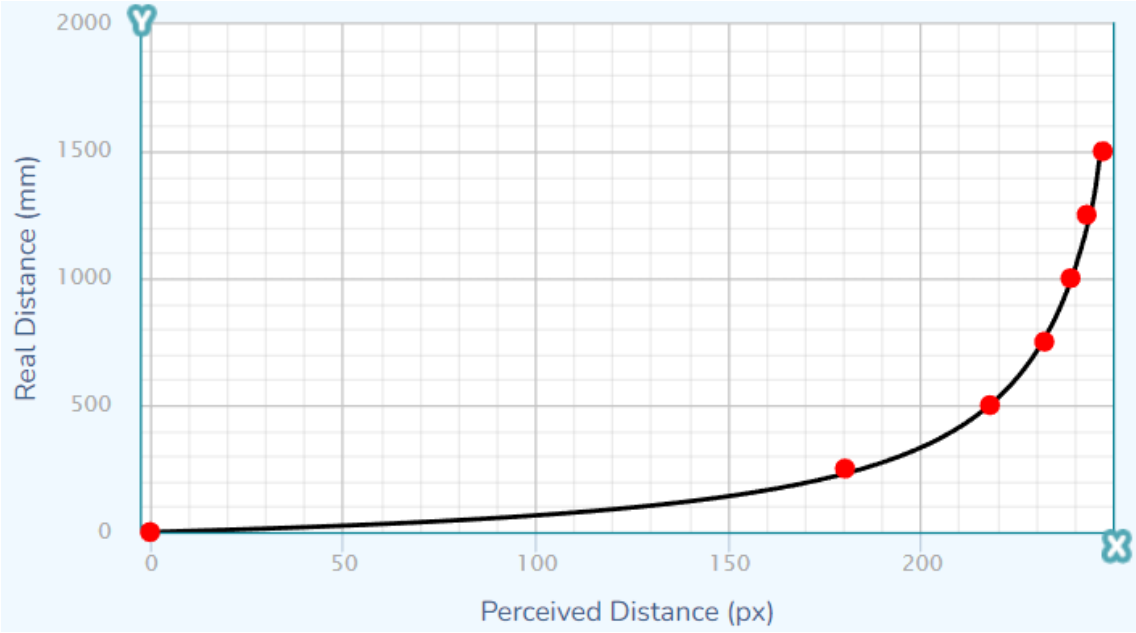


Figure D.9: Ray at 0

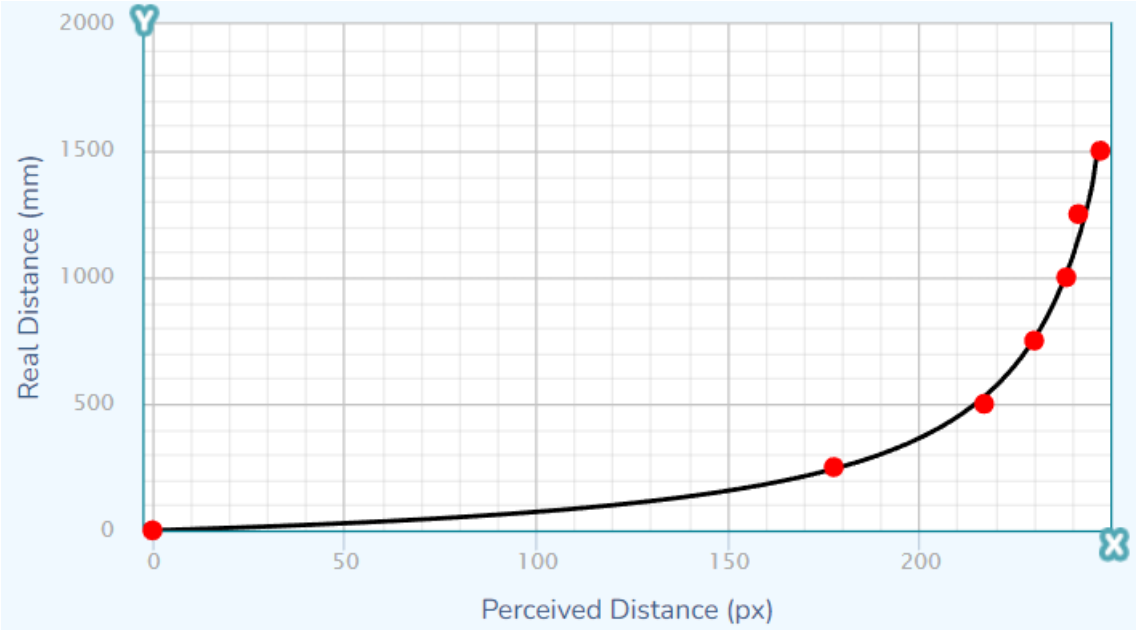


Figure D.10: Ray at $\frac{1}{8}\pi$

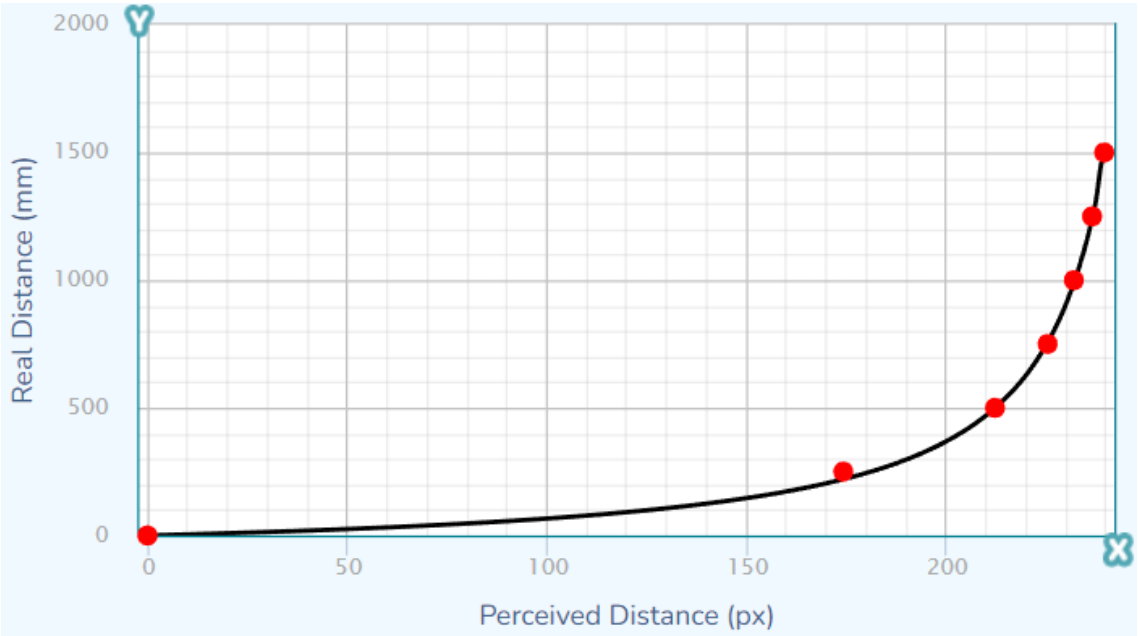


Figure D.11: Ray at $\frac{1}{4}\pi$

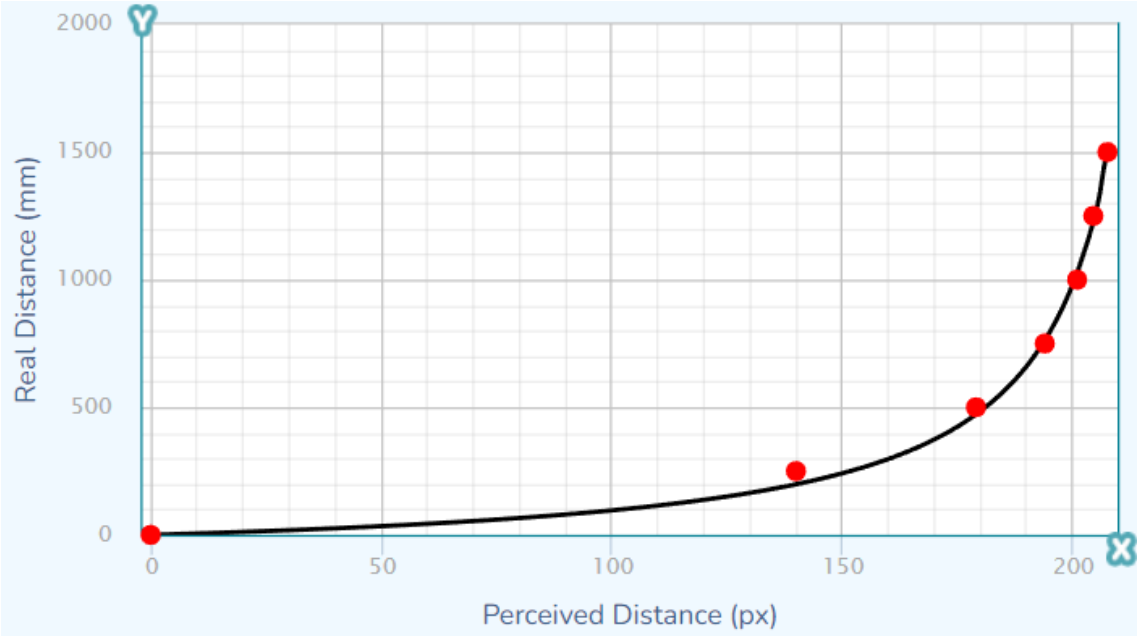


Figure D.12: Ray at $-\frac{3}{4}\pi$

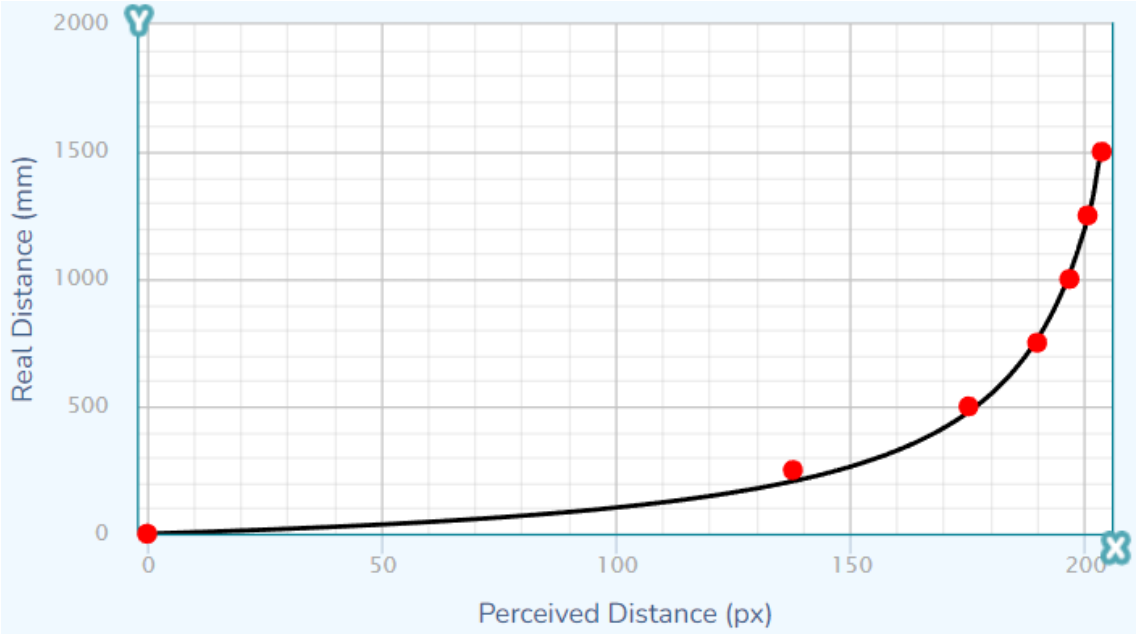


Figure D.13: Ray at $-\frac{7}{8}\pi$

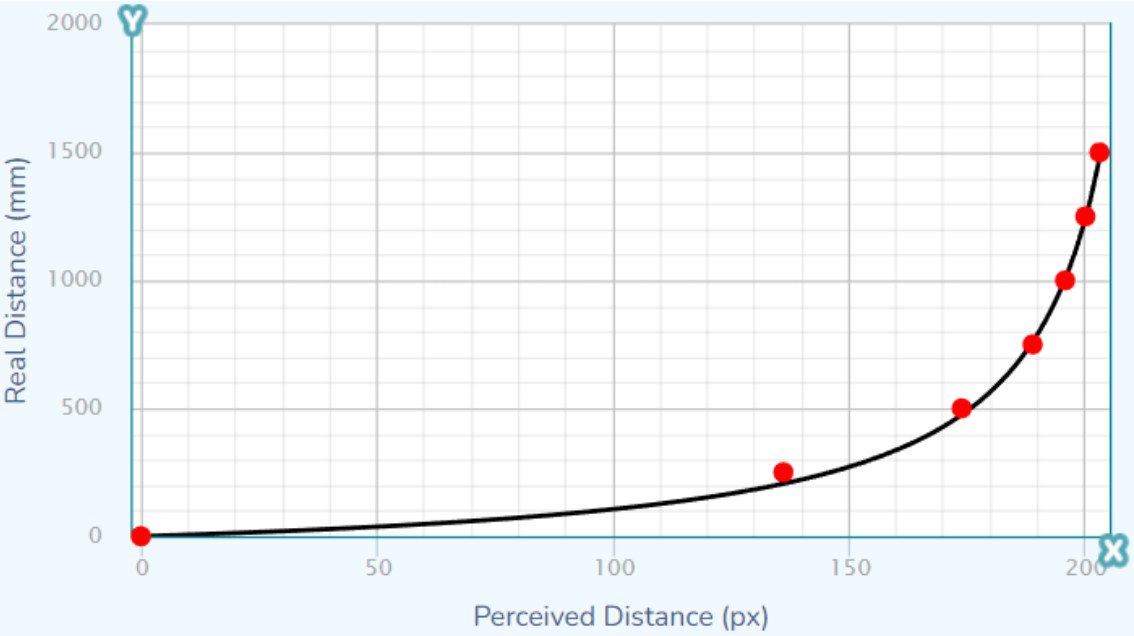


Figure D.14: Ray at π

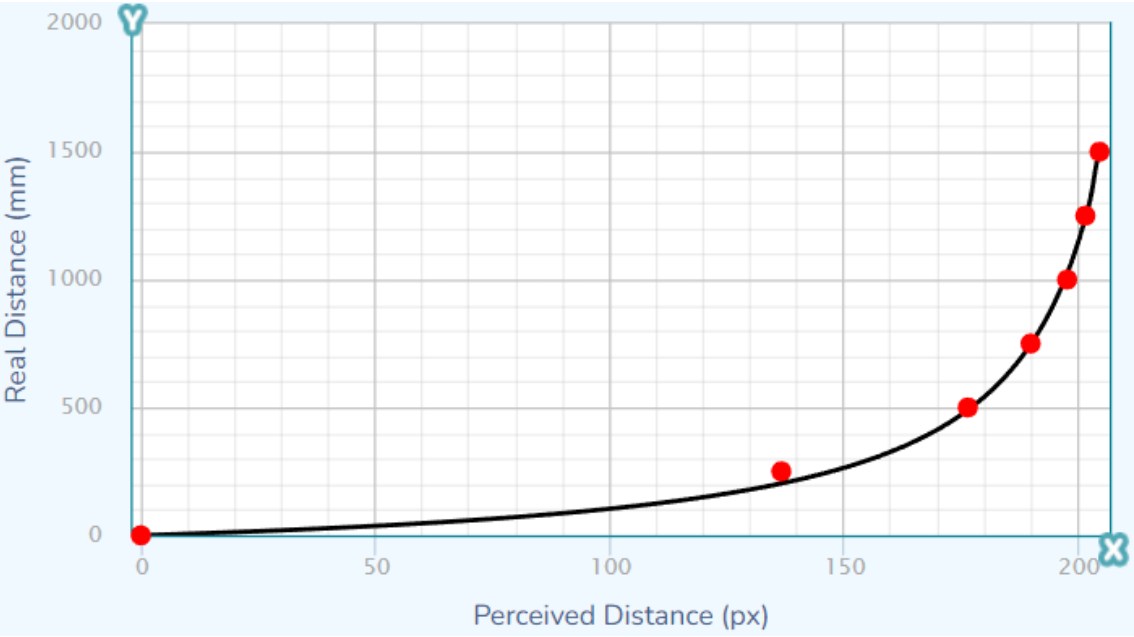


Figure D.15: Ray at $\frac{7}{8}\pi$

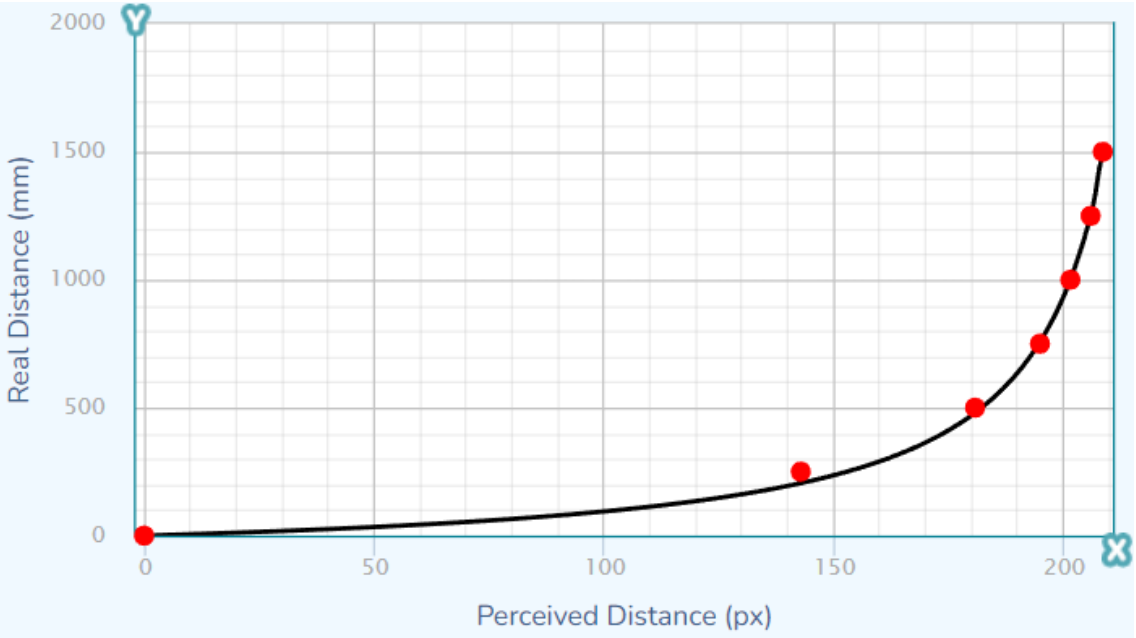


Figure D.16: Ray at $\frac{3}{4}\pi$

Appendix E

Reinforcement Learning Results

Videos of testing on the real robot can be found in urlhttps://youtube.com/playlist?list=PL7CIfqtisIW1GHEnlYJhx
and rosbags containing the same tests can be found in https://github.com/BerserkingIdiot/robot_control_utils_ros1/tree/main/bags.

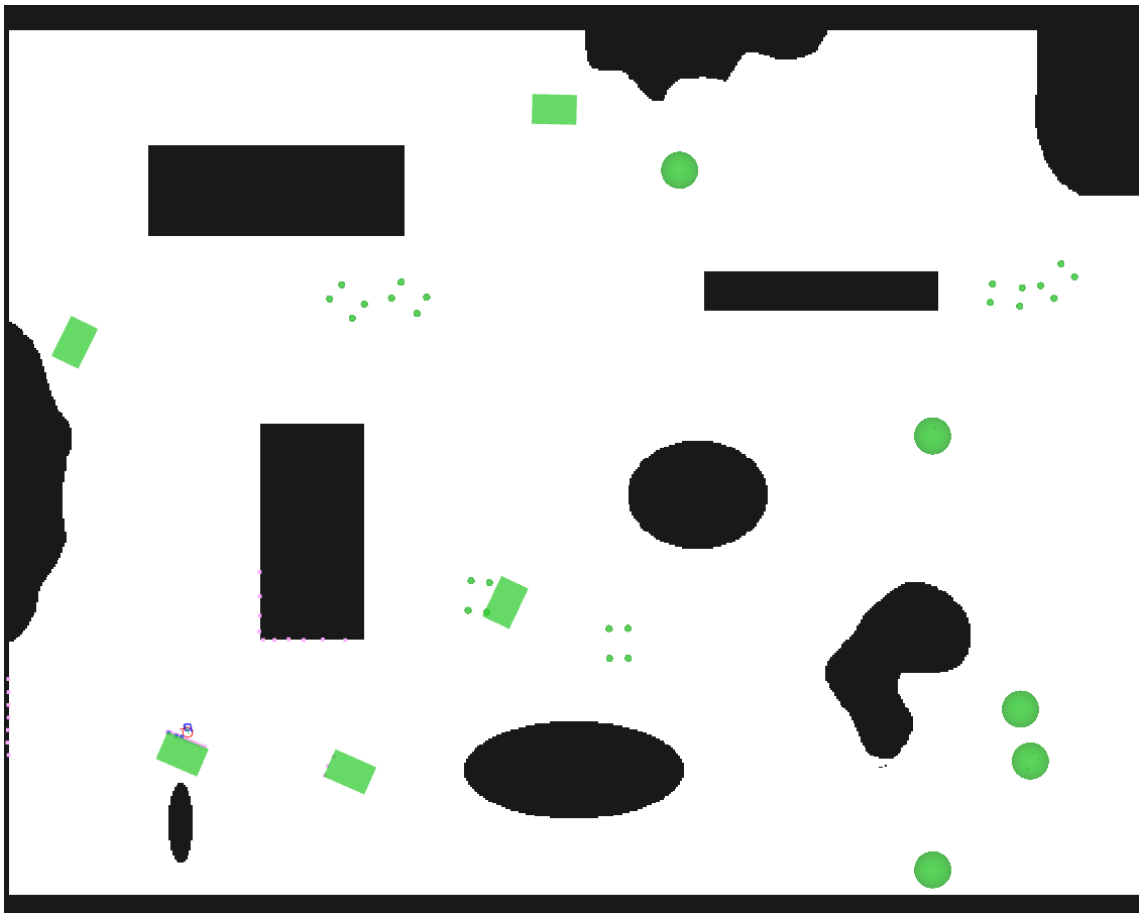


Figure E.1: Test with Configuration 1, Complete Environment

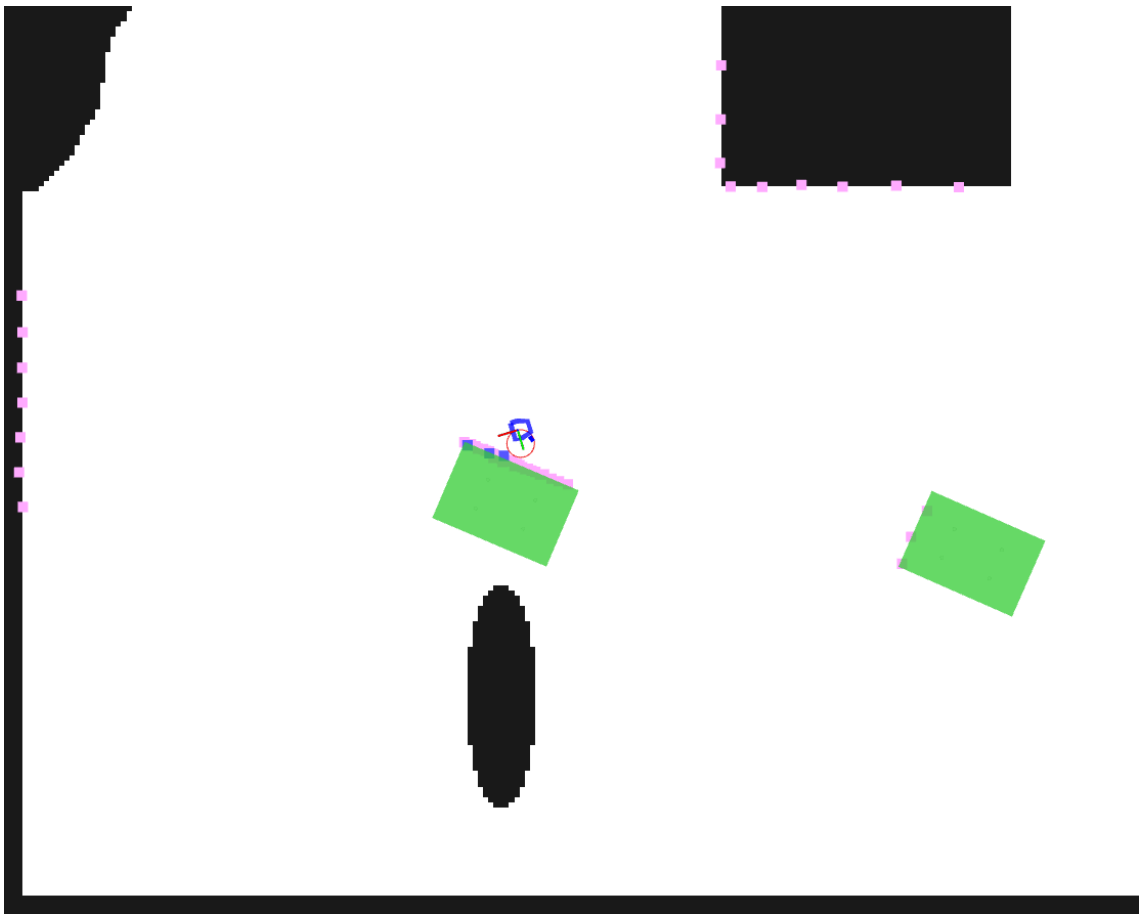


Figure E.2: Test with Configuration 1, Close-up

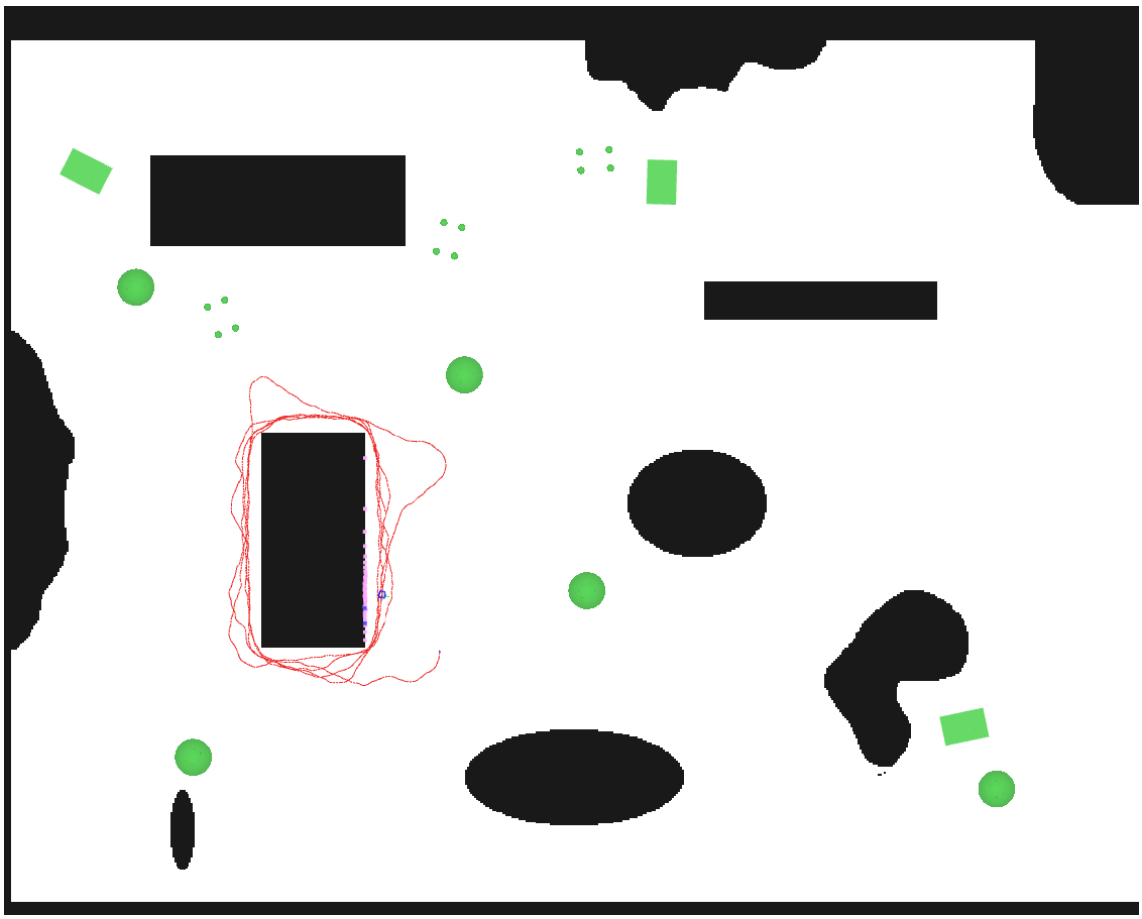


Figure E.3: Test 1 with Configuration 2, Complete Environment

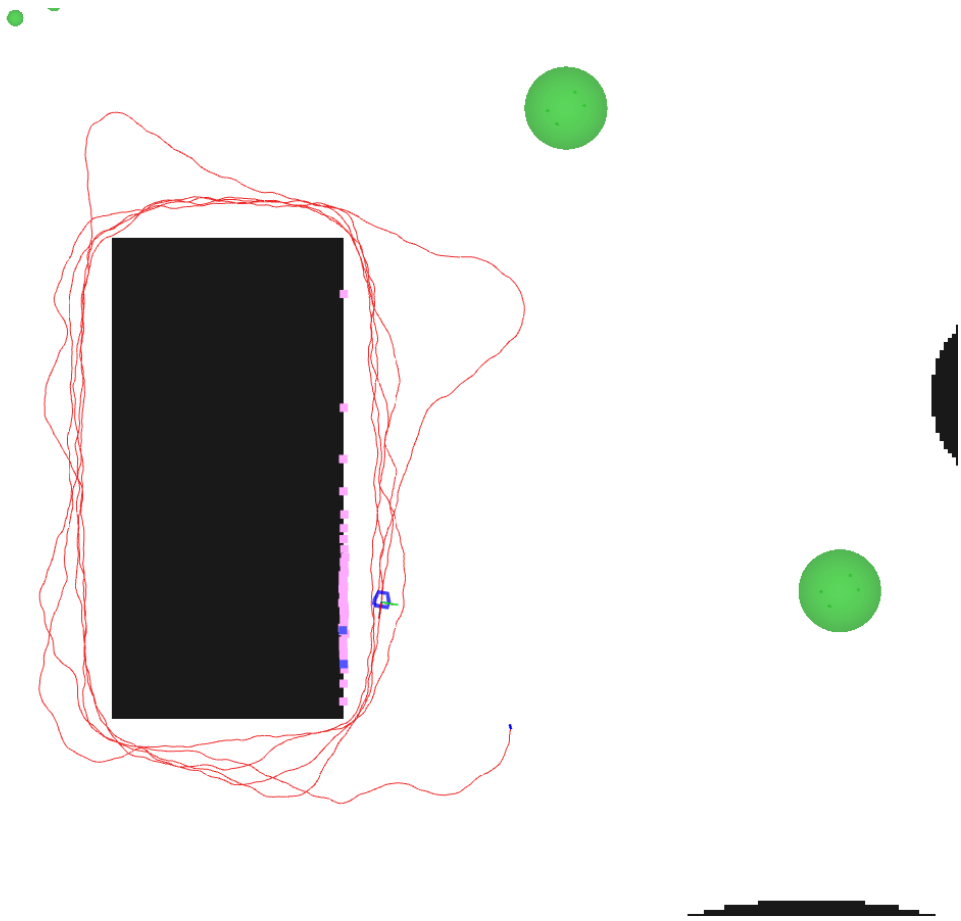


Figure E.4: Test 1 with Configuration 2, Close-up

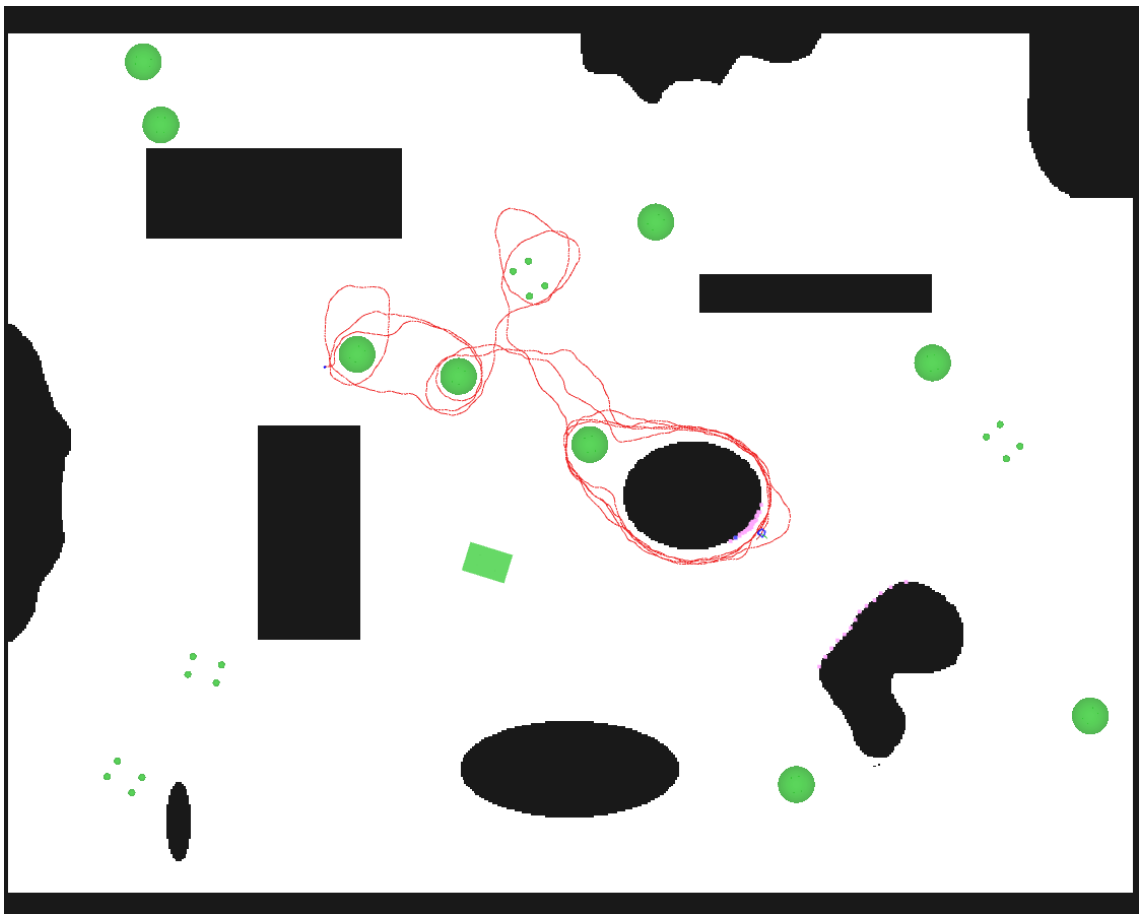


Figure E.5: Test 2 with Configuration 2, Complete Environment

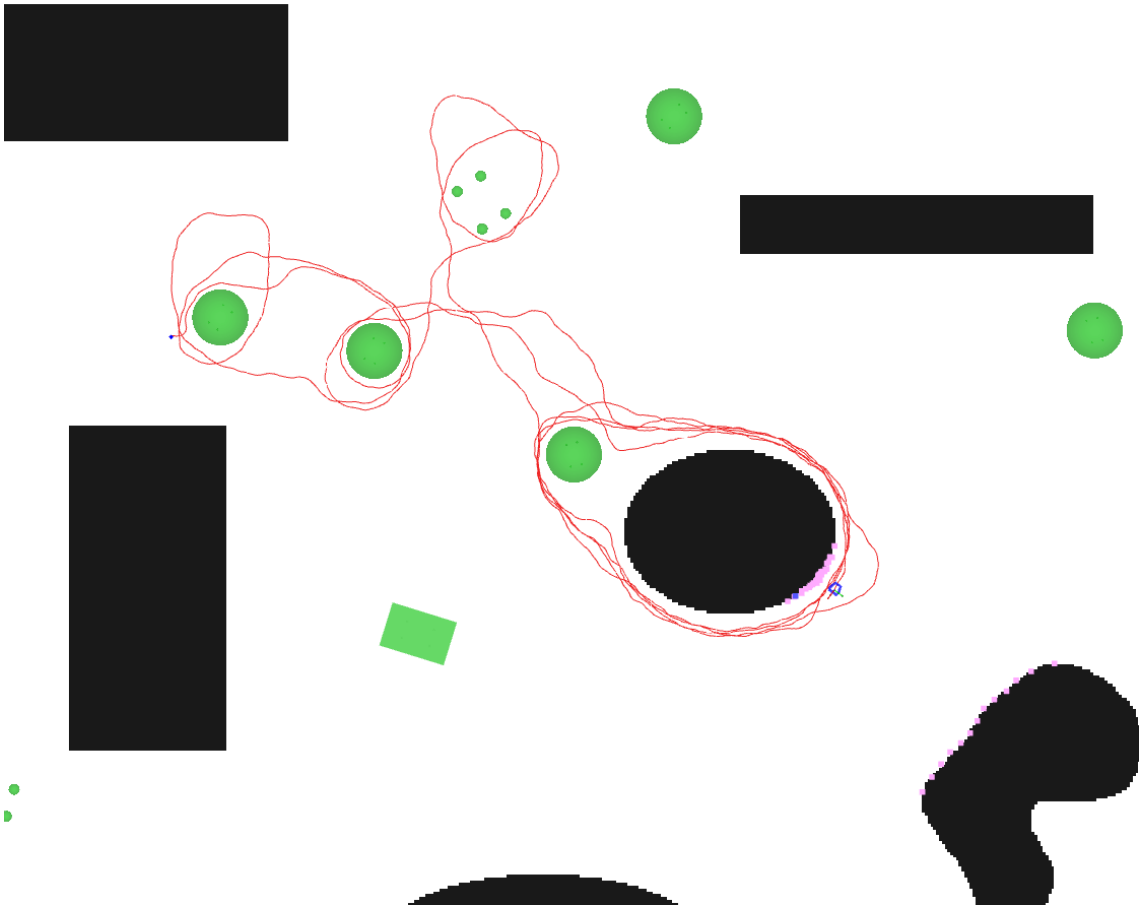


Figure E.6: Test 2 with Configuration 2, Close-up

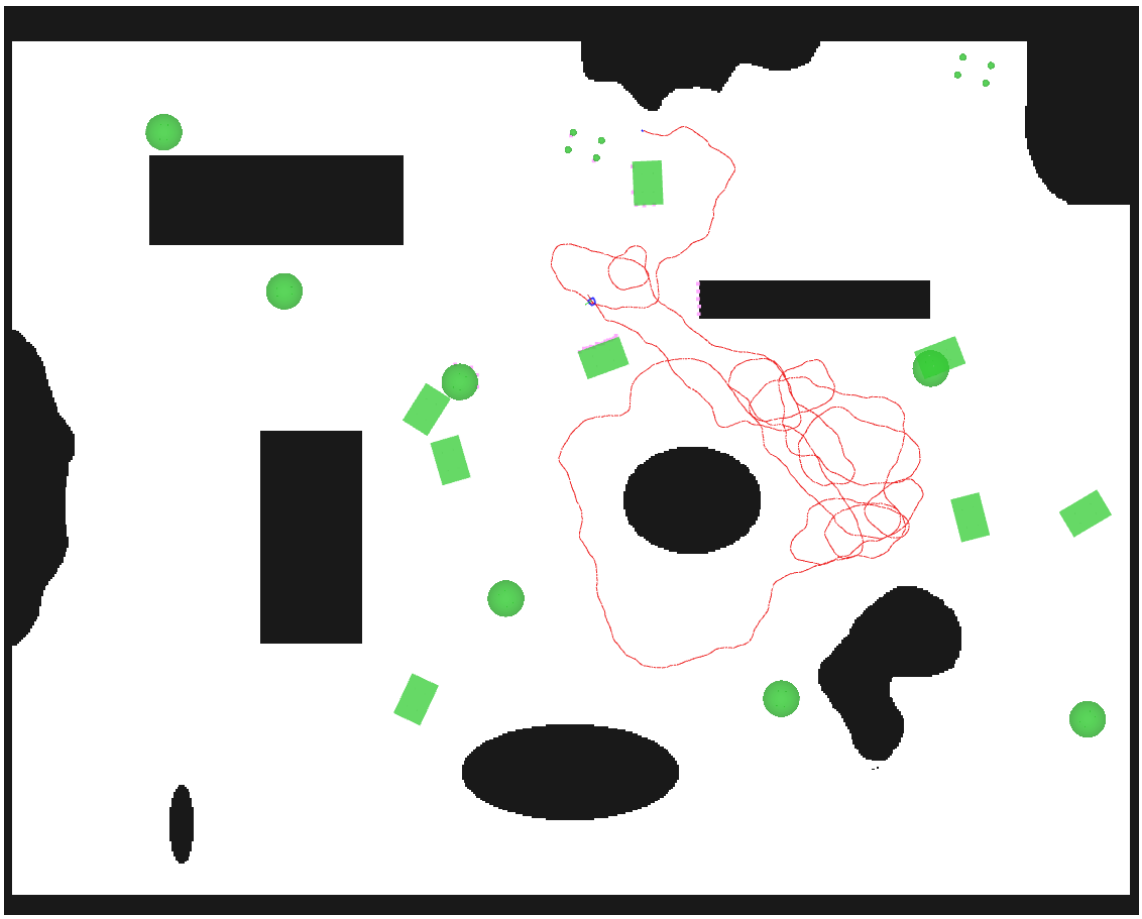


Figure E.7: Test 1 with Configuration 3, Complete Environment

Figure E.8: Test 1 with Configuration 3, Close-up

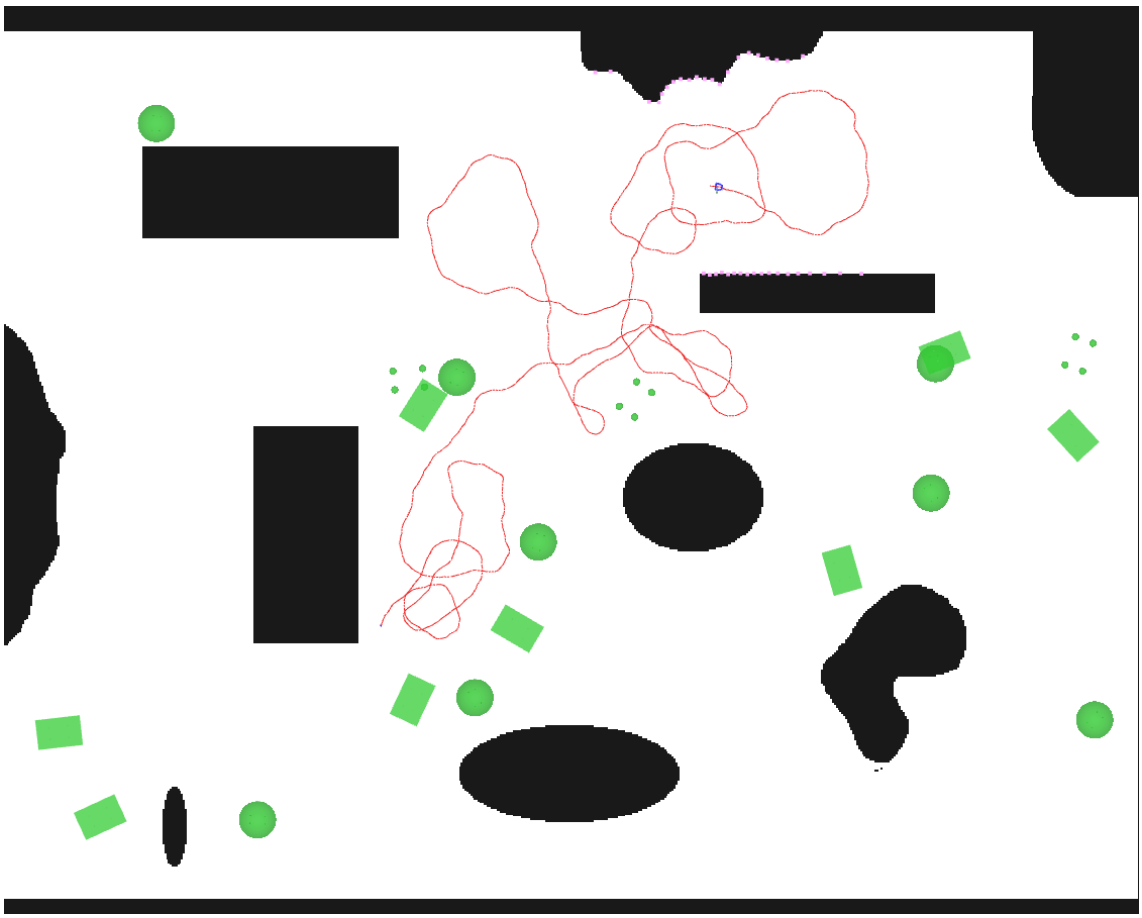


Figure E.9: Test 2 with Configuration 3, Complete Environment

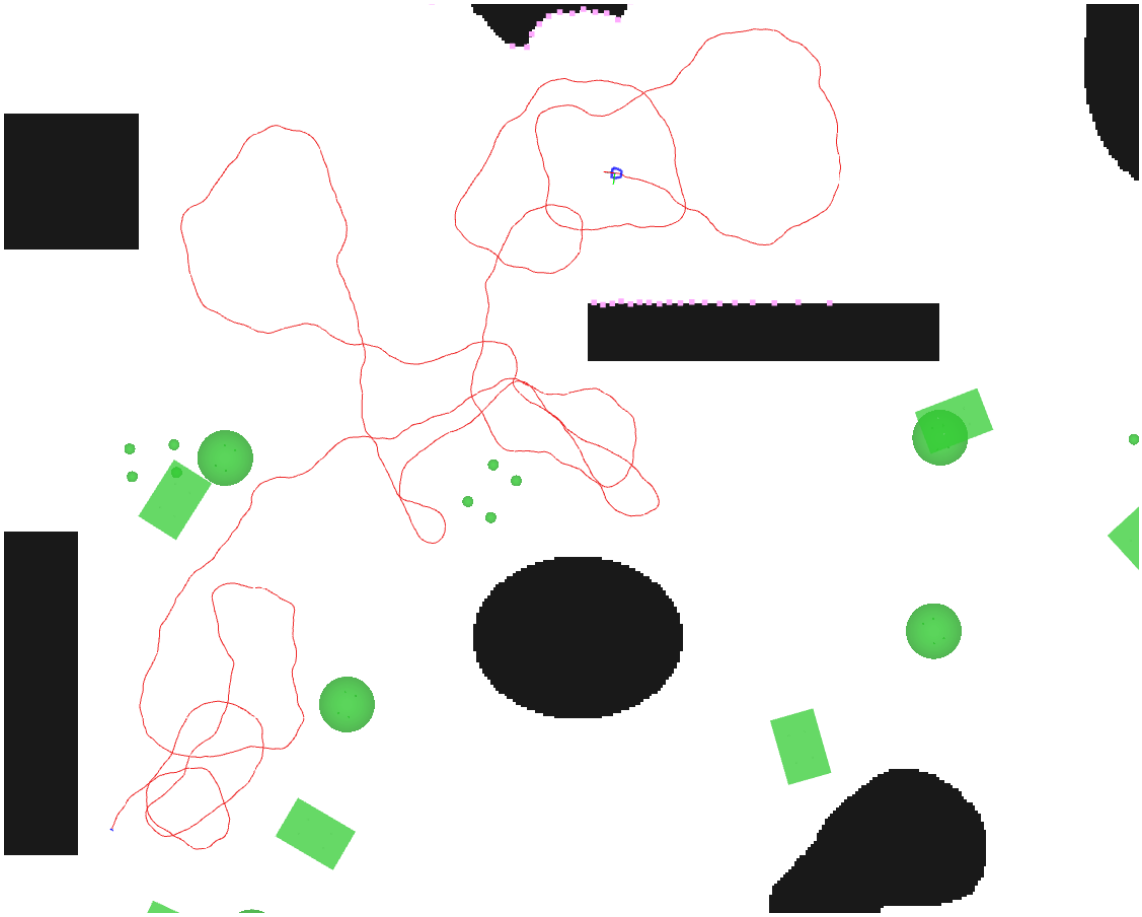


Figure E.10: Test 2 with Configuration 3, Close-up