



UNIVERSIDADE D  
COIMBRA

Rui Jorge Loureiro Simão

DESIGN, 3D MODELING AND SIMULATION  
OF AN AUTONOMOUS BEACH CLEANING  
ROBOT

Dissertation within the ambit of the Master's degree in Electrical and Computer Engineering in the field of Robotics, Control and Artificial Intelligence, oriented by the Prof. Doctor António Paulo Mendes Breda Dias Coimbra and by the Prof. Doctor Manuel Marques Crisóstomo, and presented to the Department of Electrical and Computer Engineering of the Faculty of Science and Technology of the University of Coimbra.

February, 2024







FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
COIMBRA

# **Design, 3D Modeling and Simulation of an Autonomous Beach Cleaning Robot**

**Mestrado em Engenharia  
Eletrotécnica e de Computadores**

**Author:**  
Rui Jorge Loureiro Simão

**President of the Jury:**  
Prof. Doctor Jorge Nuno de Almeida e Sousa Almada Lobo

**Vowels:**  
Prof. Doctor Mahmoud Tavakoli  
Prof. Doctor António Paulo Mendes Breda Dias Coimbra

Coimbra, February 2024



*"In the face of adversity, remember that every storm eventually passes, and with it comes the opportunity for growth and **new beginnings.**"*



# Acknowledgments

I would like to express my deepest gratitude to my parents for their unwavering support throughout this journey. Their constant encouragement, love, and understanding have been instrumental in my success. They have been there for me during both the difficult and easy times, providing a sense of stability and strength. Completing this dissertation is not just an achievement for me, but also for them, as they have been with me every step of the way.

I would also like to extend my heartfelt thanks to my godmother and cousin for their presence and support. Their belief in my abilities and their words of encouragement have been invaluable. Their unwavering faith in me has given me the confidence to persevere and overcome challenges.

To my beloved dog, I want to apologize for the long weeks without seeing me. Your unconditional love and companionship have brought me comfort and joy, even during the most stressful times. Your wagging tail and playful antics always managed to brighten my day.

Lastly, I want to express my appreciation to my friends, especially to Héber. Their friendship, laughter, and humongous support have been a constant source of strength. Their understanding of my commitments and their willingness to listen and offer advice have been invaluable.

I would like to thank my advisors, Professor Doctor António Paulo Mendes Breda Dias Coimbra, and Professor Doctor Manuel Marques Crisóstomo, as well as Mechanical Engineer João Luis Lourenço, for the support given, the availability to help and their teachings.



To all those mentioned above, and to anyone else who has contributed to my journey, I am eternally grateful. Your support has been a guiding light, helping me navigate through the ups and downs of this dissertation. Thank you for being by my side and believing in me.

The author acknowledges Fundação para a Ciência e a Tecnologia (FCT) for the financial support to the project.

# Abstract

This master thesis presents the development of an autonomous beach cleaning robot using ROS (Robot Operating System), image recognition, and a special proposed robot with a robotic arm. The work objective was to create a new autonomous robot that would contribute to making the world cleaner by focusing on beach environments, which pose unique challenges in terms of terrain, robot localization, and environmental perception.

The work involved integrating various tools and technologies, including AI-based image recognition, the utilization of ROS for the development of a virtual robot with a robotic arm and the use of CAD tools to develop a gripper. The development process required learning and understanding these tools.

This work includes the creation of a simulated environment, composed by a beach terrain, litter objects and the robot with all its components like a litter collecting tool, deposit basket and robotic arm. The dissertation proceeds with the usage of image recognition algorithms, as YOLOv7, and multiple datasets, proceeding with the ROS development, integration, and testing.

The ROS framework played a crucial role in the development and integration of different components, including path planning, robotic arm control, video inference, and drive control. Several ROS packages were also developed to manage these specific functionalities.

The results demonstrated the effectiveness of the system developed in a simulated environment.

The work serves as a basis for future advancements in the field of autonomous cleaning robots and encourages further exploration, in particular for beach environments, and implementation in real-world scenarios.

**Keywords:** ROS, image recognition, autonomous robot, beach cleaning tool.

# Resumo

Esta dissertação de mestrado apresenta o desenvolvimento de um robô autônomo de limpeza de praias utilizando ROS (Robot Operating System), reconhecimento de imagem e equipado de um braço robótico. O objetivo do trabalho foi criar um robô que contribuísse para tornar o mundo mais limpo, com foco em ambientes de praia, que apresentam desafios únicos em termos de terreno, localização de robôs e percepção ambiental.

O trabalho envolveu a integração de várias ferramentas e tecnologias, incluindo reconhecimento de imagem baseado em IA e a utilização de ROS para o desenvolvimento de um virtual robô equipado com um braço robótico. O processo de desenvolvimento exigiu a aprendizagem e a compreensão dessas ferramentas, que não haviam sido exploradas extensivamente antes.

Este trabalho inclui a criação de um ambiente simulado em ROS, em terreno de praia, lixo e o robô concebido com os seguintes componentes: balde para depositar o lixo, braço robótico e uma ferramenta de recolha do lixo. A dissertação prossegue com a descrição da utilização de algoritmos de reconhecimento de imagens, como o YOLOv7, e o recurso a vários *datasets*, seguindo com o desenvolvimento, integração e testes em ROS.

A utilização do ROS desempenhou um papel crucial no desenvolvimento e integração de diferentes componentes, incluindo planeamento de trajetória, controlo do braço robótico, inferência de vídeo e controlo de navegação. Foram desenvolvidos vários pacotes ROS para lidar com essas funcionalidades específicas.

Os resultados obtidos demonstraram a viabilidade da utilização do robô proposto.

O trabalho serve como base para avanços futuros no campo de robôs de limpeza autônomos e incentiva mais exploração e implementação em cenários do mundo real.

**Palavras-chave:** ROS, reconhecimento de imagem, robô autônomo, ferramenta de limpeza de praia.

# Contents

Acknowledgments .....	viii
Abstract .....	x
Resumo .....	xii
Contents .....	xiv
List of Figures .....	xvi
List of Tables.....	xvii
List of Acronyms.....	xviii
1. Introduction .....	1
1.1. Context and Motivation .....	1
1.1.1. Simple Considerations About Waste .....	2
1.2. Objectives .....	2
1.3. Outline of the Dissertation.....	3
2. Literature review and State of the Art.....	5
2.1. Trends in Waste Found on Beaches .....	5
2.2. Sediments' Grain Size Classification .....	6
2.3. Current Methods for Litter Collection on Beaches.....	7
2.3.1. Other Environmental Innovations .....	9
3. Tools and Methods .....	13
3.1. Hardware Tools.....	13
3.2. Software and Web-based Tools .....	14
3.2.1. 3D Modeling .....	14
3.2.2. Artificial Intelligence Tools .....	14
3.2.3. Computer Vision tools.....	16
3.2.4. ROS and ROS packages.....	16
4. Work Developed .....	18
4.1. Operational Concept Description .....	18

4.1.1.	Robot End Development .....	23
4.2.	End-effector Tool .....	23
4.3.	Identifying Litter with AI in Real-world Coordinates .....	27
4.3.1.	TACO dataset and TensorFlow .....	27
4.3.2.	YOLO and PyTorch .....	28
4.3.3.	Pixel to World Coordinates .....	31
4.4.	Development and Integration in ROS .....	32
4.4.1.	Environment and Robot Building .....	32
4.4.2.	Developed ROS Packages .....	35
4.5.	ROS Simulation .....	41
5.	Results .....	42
5.1.	Litter Detection .....	42
5.2.	Litter Collection .....	44
5.3.	System Simulation .....	45
6.	Conclusion .....	49
6.1.	Future Work .....	50
	References .....	51
	ANNEX A .....	xviii
	ANNEX B .....	xi
	ANNEX C .....	xix
	ANNEX D .....	xx
	ANNEX E .....	xxiii
	ANNEX F .....	xxiv
	ANNEX G .....	xxvi
	ANNEX H .....	xxvii
	ANNEX I .....	xxxii

# List of Figures

Figure 1 - Total Abundance and Composition graph's categorizing and establishing a Top 10 types of materials [6].....	6
Figure 2 - Industrial tractor with a dragging tool attached sweeping a southern beach of Spain with the intent to smoothing and filter litter of the sand.....	8
Figure 3 - Solarino Beach Cleaner Robot developed by Dronyx with attachment. [8] ....	9
Figure 4 - BeBot, developed by The Searial Cleaners, controlled by an operator performing beach cleaning. [9] .....	9
Figure 5 - BeachBot developed by Project.BB roaming a beach searching for cigarette butts to catch with it frontal tool located bellow the dome. [10] .....	9
Figure 6 - New concept robot to catch litter having a virtual operator controlling a robotic arm through a VR kit. [10].....	9
Figure 7 - Oceans Cleanup's waste trap for ocean solution. [11] .....	10
Figure 8 - Oceans Cleanup's waste collection station for rivers solution. [11] .....	10
Figure 9 - Seabin installed in Clube Naval de Cascais, Portugal [15].....	11
Figure 10 - DustClean (at the edges) and DustCart (in the middle) developed by ROBOTECH srl [17].....	12
Figure 11 - TACO's number of annotations and classes presented in graph form [21].	15
Figure 12 - TACO's large background litter images coverage [21]. .....	15
Figure 13 - Example of a Rqt_graph output of an ROS execution with Node A subscribing to the same topic Node B is publishing.....	17
Figure 14 - 3D render of litter catching tool design A. ....	24
Figure 15 - 3D render of litter catching tool design B. ....	24
Figure 16 - 3D render of litter catching tool design C v1. ....	25
Figure 17 - 3D render of litter catching tool design C v2. ....	25
Figure 18 - One piece of the end-effector tool mid printing process. ....	27
Figure 19 - Results from testing the TACO dataset after patching the code. ....	28
Figure 20 - Model training log using YOLOv7. ....	30
Figure 21 - YOLOv7 inputs and outputs illustration. ....	31
Figure 22 - Rviz screenshot of video inference and the litter's position. Above is the video inference and bellow a debugging window with the robot description and transforms. ....	32
Figure 23 - Litter affected by the buoyancy plugin on Gazebo with vertical orientation instead of horizontal.....	33
Figure 24 - Gazebo world with clouds and waves for realism. Robot ready to catch the scattered trash.....	33



Figure 25 - Packages relation diagram. ....	35
Figure 26 - Detection of objects 1 and 2 in the upper row, and object 3 and 4 last row. ....	42
Figure 27 – Detection zone with confidence of 35%, 50% and 85% for object 4. ....	43
Figure 28 – Detection zone with confidence of 35%, 50% and 85% for object 2. ....	43
Figure 29 - Tool design C v2 collecting objects 1 and 2 in the upper row, and object 3 and 4 lower row. ....	44
Figure 30 – Boxbot defined sweeping trajectory and trash. ....	45
Figure 31 – Boxbot detour from the defined trajectory to catch object 1 resuming the pathing to the defined trajectory. ....	45
Figure 32 - Boxbot collecting and depositing the object 1. ....	46
Figure 33 - Rviz image of yoloV7 with no detection returned identified by the red rectangle. ....	47
Figure 34 - Rviz visualization of Boxbot detecting and planning parking for the collection of object 2. ....	47
Figure 35 - Boxbot trying to collect object 4. ....	48

## List of Tables

Table 1 - Percentage distribution of trash composition observed in the APA study [6].	5
Table 2 - International scale for granularity classification [7].	7
Table 3 - Initial iteration for the concept's physical characteristics, aligned with the stakeholders' expectations.	19
Table 4 - Description of modes and their operations.	22
Table 5 - Pros and cons of design C related to B.	26
Table 6 - List of ROS computation processes used in the "boxbot_path_planner" node.	36
Table 7 - List of ROS computation processes used in the "litter_detect" node.	38
Table 8 - List of ROS computation processes used in the "arm_controll_server" node.	39

# List of Acronyms

AI – Artificial Intelligence

APA – Agência Portuguesa do Ambiente (Environmental Portuguese Agency)

API - Application Programming Interface

CAD – Computer-Aided Design

CPU – Central Processing Unit

CUDA – Compute Unified Device Architecture

cuDNN – NVIDIA CUDA Deep Neural Network

ETA – Estimated Time of Arrival

FoV – Field of View

GPS – Global Positioning System

GPU – Graphics Processing Unit

GT – Ground Truth

IMU – Inertial Measurement Unit

LiDAR – Light Detection and Ranging

mAP – Mean Average Precision

OCD – Operational Concept Description

PID – Proportional Integral Derivative

RAM – Random Access Memory

ROS – Robot Operating System

TACO – Trash Annotations in Context

YOLO – You Only Look Once

# 1. Introduction

## 1.1. Context and Motivation

In recent years, there has been a growing awareness of the detrimental effects of pollution, with human activities being identified as the primary cause. In response, efforts have been made by humanity to address the issue of pollution through various means such as legislation, technological advancements, and public demonstrations. These undertakings aim to ensure a better future for our survival and the well-being of all life on Earth. However, despite these efforts, the accumulation of waste in various environments remains a recurring problem that captures public attention through media outlets and social networks.

Among the most concerning environments affected by pollution are the oceans. As beaches serve as a natural boundary between land and sea, they become a significant site for the collection and accumulation of waste. Thus, they function both as a repository for waste coming from the seas and as a collection point for litter from land.

Portugal, being a coastal country with a population predominantly residing near the coast, is particularly vulnerable to this issue. Additionally, the ocean holds substantial economic value and is a source of wealth for the country. Consequently, it is expected that these coastal areas, including seas, beaches, and dunes, accumulate a significant amount of trash.

Trash collection on beaches primarily occurs through the efforts of public service companies hired by City Halls, as well as through volunteering initiatives. These methods aim to reduce litter on beaches, but their effectiveness is limited. Contractors may also employ heavy machinery, but this approach is both expensive and renders the beach inaccessible during the cleaning operation. Consequently, the frequency of beach cleaning through these means is low, leading to the accumulation of waste over time.

### 1.1.1. Simple Considerations About Waste

In the English language, waste is defined by Oxford English Dictionary as materials that are no longer needed and are discarded. Waste can originate from various sources, including domestic, industrial, healthcare, or technological activities. Additionally, other terms such as garbage, trash, rubbish, refuse, or litter are commonly used to refer to waste. According to Encyclopedia Britannica [1], these terms can be categorized as follows:

- Garbage primarily refers to decomposable food waste.
- Rubbish is mostly dry material such as glass, paper, cloth, or wood.
- Refuse consists of garbage and rubbish.
- Trash is rubbish that includes bulky items such as appliances, furniture, etc.

In this dissertation, the focus will be primarily on the term "litter," which refers to small pieces of rubbish left on the ground in public places [2]. It is important to note that when any of the other terms mentioned earlier are used, they will be understood within the scope of this definition.

## 1.2. Objectives

Given the advancement of robotics in dull, dirty, and dumb tasks, this field of study presents an ideal opportunity, for the deployment of robots. Consequently, this thesis aims to achieve the objective of develop and test, in a simulated environment, a robot that autonomously cleans a beach. To do so, it will be necessary:

1. Development:
  - 1.1. Operational Concept Description (OCD) [3], description of the intended users, system, uses, how the system is to be used.
  - 1.2. Develop in ROS a virtual robot with a robotic arm.
  - 1.3. Develop a deposit basket to secure litter.

- 1.4. Develop a gripper to collect selected types of litter and attach it to the arm.
  - 1.5. Integrate artificial intelligence (AI) to identify such objects.
  - 1.6. Develop and integrate navigation algorithms to perform the beach cleaning task.
2. Testing:
    - 2.1. Demonstrate the functionality of the robot in a simulated environment using ROS (Robot Operating System).

While the development of such a robot would not entirely solve the problem of beach litter, it would contribute to its mitigation by introducing a new tool that complements existing cleaning measures.

### 1.3. Outline of the Dissertation

This dissertation is structured into five chapters, each serving a specific purpose in addressing the research topic.

#### Chapter 1: Introduction

In this chapter, the dissertation's theme is introduced, and the existing problem that motivated the research is presented.

#### Chapter 2: Literature review and State of the Art

This chapter focuses on the state-of-the-art research and is subdivided into three sub-chapters. The first sub-chapter presents a study about litter sizes and types, which is essential for achieving the end goal of the research. The second sub-chapter explores additional relevant studies about sediments' grain size classification. The last sub-chapter provides an overview of existing methods and robots developed by other entrepreneurs that are involved in cleaning actions.

### Chapter 3: Tools and Methods

The third chapter is divided into three sub-chapters. The first sub-chapter focuses on hardware tools, while the second sub-chapter discusses software tools, including 3D CAD tools, AI tools, computer vision tools, and ROS.

### Chapter 4: Work Development

This chapter is divided into three sub-chapters, each explaining the research process and showcasing the results. The first sub-chapter details the development of a litter-catching tool. The next sub-chapter outlines the procedures used to identify litter deposits in the environment and transform pixel coordinates into real-world coordinates. The final sub-chapter describes the development of an autonomous moving robot equipped with the robotic arm in a simulated environment using ROS.

### Chapter 5: Conclusion

The fifth and final chapter provides the conclusion of the dissertation, reporting on the achievement of the main objectives and their degree of success. The chapter concludes with suggestions for future work that can be pursued in this research area.

## 2. Literature review and State of the Art

### 2.1. Trends in Waste Found on Beaches

Annually, the world generates a staggering 2.01 billion tons of municipal solid waste, with at least 33% of this waste not being managed in an environmentally safe manner [4].

In the context of Portugal, the country produced an average of 1.40 kg of waste per capita per day in 2020. Out of this, 48% was deposited in landfills, while 38% was prepared for reuse and recycling [5]. In a study conducted by the APA – the Portuguese environmental agency – in 2022, a trash monitoring program was implemented on fourteen beaches, resulting in 55 campaigns. Table 1 presents the findings from this study, revealing that 88% of the encountered trash was of plastic origin [6].

*Table 1 - Percentage distribution of trash composition observed in the APA study [6].*

<b>Plastics</b>	<b>88%</b>
<b>Sanitary items</b>	<b>6,0%</b>
<b>Cardboard and paper</b>	<b>1,8%</b>
<b>Metal</b>	<b>1,3%</b>
<b>Clothing</b>	<b>0,6%</b>
<b>Wood</b>	<b>0,6%</b>
<b>Medical items</b>	<b>0,5%</b>
<b>Clay and ceramics</b>	<b>0,4%</b>
<b>Glass</b>	<b>0,3%</b>
<b>Rubber</b>	<b>0,2%</b>

Understanding the materials encountered and their sizes is crucial for the dimensional planning of the catching tool of the robot. The APA study provides valuable insights in this regard, with Figure 1 presenting the top 10 materials encountered, depicted in graph form.

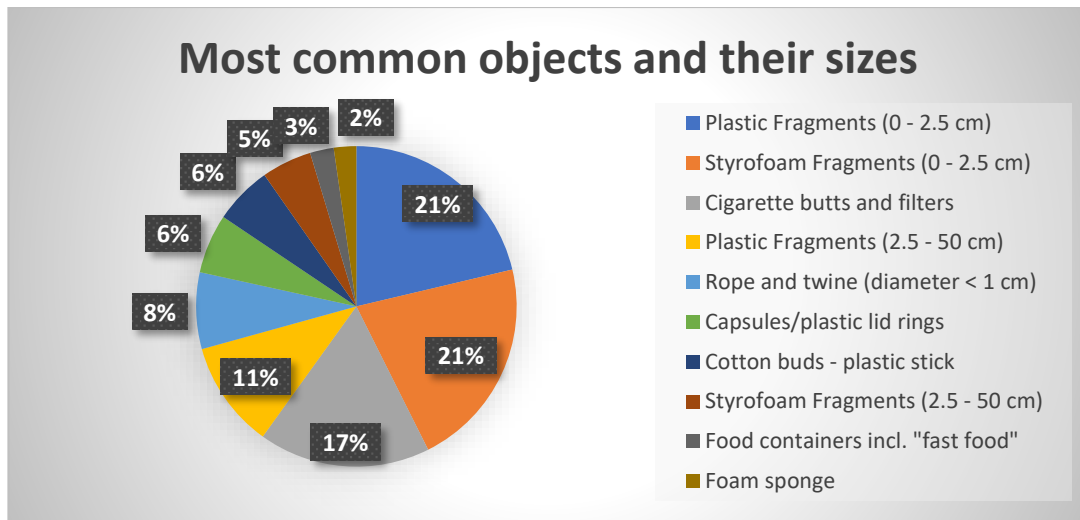


Figure 1 - Total Abundance and Composition graph's categorizing and establishing a Top 10 types of materials [6].

## 2.2. Sediments' Grain Size Classification

A review of existing literature was conducted to determine the typical level of granularity found on a beach. Understanding the size of litter and the granularity of the sand is essential for assessing the potential overlap between them. Knowing this trait was also important to design the catching tool. In Table 2 are presented the findings and it is concluded that beach sand can have sizes between 0.063 and 2 millimeters.

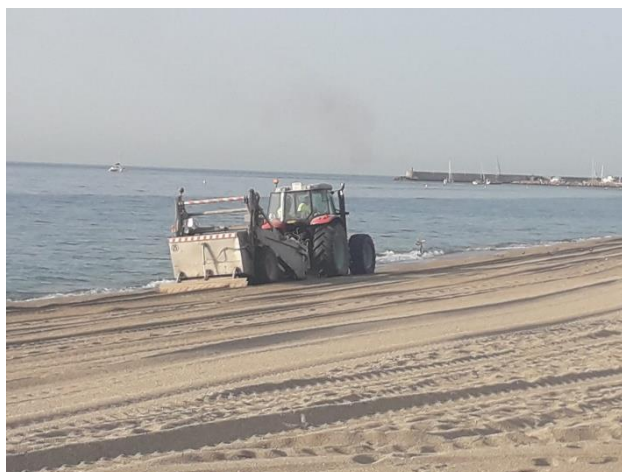


Table 2 - International scale for granularity classification [7].

<b>Classification</b>	<b>Grain Diameter (mm)</b>
Medium gravel	$6,3 < d \leq 20$
Fine gravel	$2,0 < d \leq 6,3$
Coarse sand	$0,63 < d \leq 2,0$
Medium sand	$0,2 < d \leq 0,63$
Fine sand	$0,063 < d \leq 0,2$
Coarse silt	$0,02 < d \leq 0,063$
Medium silt	$0,0063 < d \leq 0,02$
Fine silt	$0,002 < d \leq 0,0063$
Clay	$d \leq 0,002$

### 2.3. Current Methods for Litter Collection on Beaches

Breakthroughs and innovations in the field of litter collection on beaches have been relatively scarce. As mentioned earlier, the most employed methods involve volunteering actions and the utilization of contractors who employ large tractors equipped with dragging tools. This last referred tool serve to filtering out trash, which is then collected in a deposit basket, as depicted in Figure 2, while simultaneously smoothing the sand.



*Figure 2 - Industrial tractor with a dragging tool attached sweeping a southern beach of Spain with the intent to smoothing and filter litter of the sand.*

Though the scarcity of innovation and breakthroughs on this field, there are some emerging projects that involve the use of robots for beach cleaning. Among these projects, three developments stand out: the Solarino Beach Cleaner Robot, BeBot, and BeachBot.

Solarino, in Figure 3, is a “remote control Roomba” developed by Dronyx [8], an Italian venture company. Resembling a miniaturized version of the tractor depicted in Figure 2, Solarino, to collect litter, utilizes a filtering method through its dragging tool and a locomotion mechanism using caterpillar tracks. It should be noted that Dronyx is no longer active, so further details about the robot's characteristics are not available.

Another notable development is BeBot, in Figure 4, by The Searial Cleaners [9]. BeBot shares a similar appearance and functionality with Solarino. It is introduced as a 100% electric robot capable of performing multiple tasks, such as screening sand, raking seaweed, leveling beach areas, and lifting and carrying loads. The robot targets solid waste commonly found on beaches. Like the robot developed by Dronyx, BeBot is not autonomous.



Figure 3 - Solarino Beach Cleaner Robot developed by Dronyx with attachment. [8]



Figure 4 - BeBot, developed by The Searial Cleaners, controlled by an operator performing beach cleaning. [9]

Lastly, there is BeachBot, developed by Project.BB [10]. This project is currently a work in progress, with ongoing efforts to enhance the robot's robustness and versatility. Unlike the previous two robots, BeachBot, shown in Figure 5, is designed to operate autonomously. It relies on artificial intelligence (AI) for image recognition to map litter, and the developers claim success in identifying and picking up cigarette butts. The robot is specifically equipped with a tool designed for collecting cigarette butts. Project.BB is developing a new concept as they are developing a virtual reality (VR) mode of operation with an innovative designed robot, shown in Figure 6.



Figure 5 - BeachBot developed by Project.BB roaming a beach searching for cigarette butts to catch with its frontal tool located below the dome. [10]



Figure 6 - New concept robot to catch litter having a virtual operator controlling a robotic arm through a VR kit. [10]

### 2.3.1. Other Environmental Innovations

During the state-of-the-art research, it became evident that significant efforts to combat waste were focused on the sea and oceans. This is the environment where

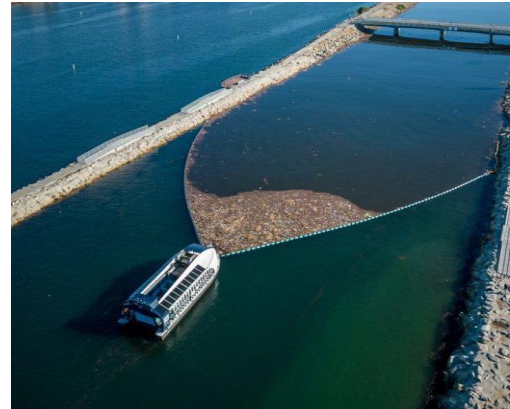
---

numerous innovations are being developed due to the pressing concerns about marine life. Here are some initiatives in the fight against littering in that field:

1. The Ocean Cleanup [11] is a prominent organization that has developed 600-meter-wide ocean floater systems, as shown in Figure 7. These systems utilize ocean currents to capture and collect plastic and microplastics up to three meters deep. The Ocean Cleanup has also designed river litter collecting stations, shown in Figure 8, strategically located downstream of rivers, reducing the delivery of waste to the oceans. They function as a stationary station using a conveyor belt to collect the downstream trash and storing them in deposit crates. Later, those crates are transported to a waste management center.
2. The Jellyfishbot, design by IADYS [12], is a small buoyant robot specifically designed for low-movement waters within a designated area, such as harbors. It operates autonomously, but without a specific trajectory, to collect floating litter.



*Figure 7 - Oceans Cleanup's waste trap for ocean solution. [11]*



*Figure 8 - Oceans Cleanup's waste collection station for rivers solution. [11]*

3. The Seabin Project, from Seabin [13, 14], which has undergone rebranding, and Collec'Thor by The Searial Cleaners [9], are stationary "vacuums" designed for marinas, harbors, and shorelines. They operate by filtering floating waste in these areas, the way it is presented in Figure 9.



*Figure 9 - Seabin installed in Clube Naval de Cascais, Portugal [15].*

In urban environments, some innovations have emerged to aid in waste collection:

4. DustClean and DustCart, depicted in Figure 10, are two autonomous robots developed by ROBOTECH srl [16, 17]. DustClean is responsible for sweeping the streets, while DustCart transports household waste outside for collection by public services. These robots are equipped with artificial vision systems and ozone and carbon monoxide sensors, to issue alerts if the air quality drops too low.



Figure 10 - DustClean (at the edges) and DustCart (in the middle) developed by ROBOTECH srl [17].

5. Corvid Cleaning [18], a Swedish enterprise, has introduced an innovative approach by utilizing animals, specifically crows, to clean up litter in exchange for a food reward.

## 3. Tools and Methods

Throughout the development of this dissertation, a range of tools, both software and hardware, were utilized to achieve the desired objective. This chapter outlines the tools employed in pursuing the goals of this dissertation.

### 3.1. Hardware Tools

As this dissertation is based on a development of a robot using a simulated environment, the use of a computer served as the primary platform for executing most of the software tools employed in this research. It is important to highlight the computer's components as they may serve as a reference for future development and replication. The computer's components were as follows:

- CPU – AMD Ryzen™ 5 3600 with stock cooler
- RAM – Team Group Kit 16GB (2 x 8GB) DDR4 3600MHz Delta RGB
- GPU – Gigabyte GeForce RTX™ 3060 Gaming OC 12Gb

In addition to the use of a computer, the Beeverycreative B2x300 3D printer was utilized for 3D printing the final design of the litter collection tool. Black PLA filament, extrusion thickness of 1,75 mm and the 20% linear infill was the configuration used to print.



### 3.2. Software and Web-based Tools

#### 3.2.1. 3D Modeling

The catching tool and deposit basket, which will be outlined by an Operational Concept Description, were designed using a 3D CAD tool, predominantly Autodesk's Fusion 360. Fusion 360 is a cloud-based software that primarily relies on CPU resources for modeling and rendering tasks. This CPU-bound nature occasionally posed challenges during development, as alterations to the model and their processing times are sometimes long. To ensure the integration with the ROS simulation, an add-on script was employed to export the model to the URDF file format [19]. Also, SOLIDWORKS with the add-on `sw_urdf_exporter` was used to replace the previous script when it could not perform. SOLIDWORKS is, also, a 3D CAD tool with a wide range of applications such as design/engineering, manufacturing, and data management.

In addition to Fusion 360 and SOLIDWORKS, Blender was experimented for 3D modeling purposes. It was employed to create a sand terrain with an applied texture, although this asset was ultimately excluded from the final simulation because during tests the robot slipped.

#### 3.2.2. Artificial Intelligence Tools

For image recognition tasks, AI was utilized. Image recognition is “a sub-category of computer vision technology that deals with recognizing patterns and regularities in the image data, and later classifying them into categories by interpreting image pixel patterns” [20]. State-of-the-art deep learning models can achieve a high level of accuracy, up to 95%, in tasks such as classification and detection, employing techniques like bounding box annotation or semantic segmentation.

To train the model, the TACO GitHub repository was briefly experimented. TACO, as described by its creators, “is an open image dataset of waste in the wild. It contains photos of litter taken under diverse environments, from tropical beaches to London



streets. These images are manually labeled and segmented according to a hierarchical taxonomy to train and evaluate object detection algorithms” [21].

TACO adopts the MS COCO (Microsoft Common Objects in Context) annotation format [22], providing essential information such as image licenses, object class, bounding box position, raw image data including pixel size, and crucial object annotations listed in JSON format. The TACO dataset contains over 4,500 annotations. The dataset provides a diverse range of images depicting litter, as shown in Figure 11 and Figure 12.

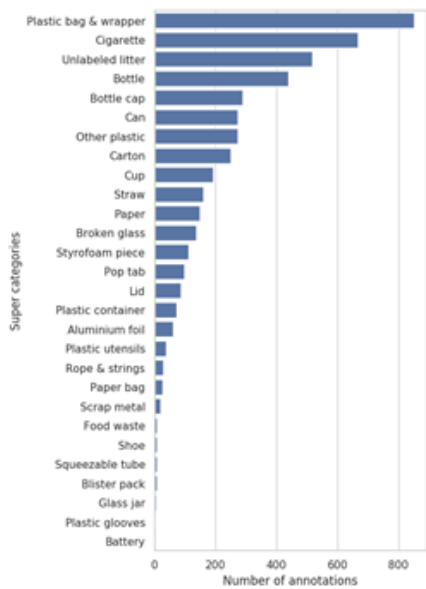


Figure 11 - TACO's number of annotations and classes presented in graph form [21].

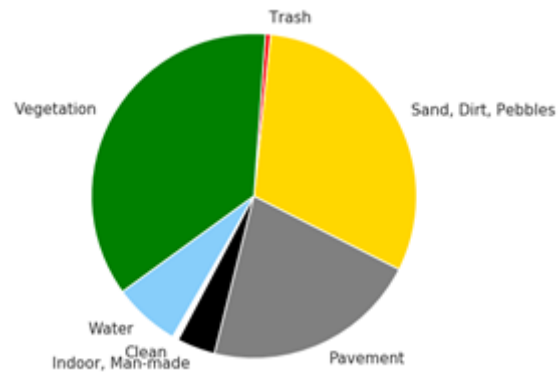


Figure 12 - TACO's large background litter images coverage [21].

Additionally, other software tools were employed for the model training, namely YOLO and PyTorch. YOLO, which stands for "You Only Look Once," is a well-known real-time object detection algorithm. While there are currently eight versions of YOLO available, YOLOv7 [23] was used in this dissertation as it was already in use at the time, prior to the release of YOLOv8 on January 10th, 2023 [24].

To access and explore datasets, a website called Roboflow was utilized. Roboflow hosts public datasets created by users and offers features such as image uploading, annotation capabilities, and the ability to combine existing datasets with newly created

ones. Roboflow does not allow the direct download of pre-trained models but allows dataset downloads with a wide selection of annotation formats. Additionally, the website has developed hosted APIs for server-side prediction. With this service it is possible to send the live feed from the camera and get as response the detection.

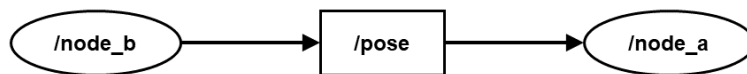
### 3.2.3. Computer Vision tools

To convert pixel coordinates into a 3D pose, a virtual Intel® RealSense™ depth camera D435 with a stereo solution and Computer Vision library was used. On the “IntelRealSense/librealsense: Intel® RealSense™ SDK library”, available on GitHub, there is a function called ‘rs2\_deproject\_pixel\_to\_point’ which returns 3D coordinates related to the camera’s referential system from the x and y pixel coordinates of image data, the same pixel coordinates of the depth data, and the intrinsic camera characteristics.

### 3.2.4. ROS and ROS packages

Robot Operating System (ROS) is an open-source middleware designed for robot development. Although it is not an actual operating system (OS), it provides services that resemble one, offering hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management [25]. ROS divides itself into user-created packages, which contain nodes that communicate with each other through topics (asynchronous messages), services (synchronous messages), and/or actions. Nodes are the executables within ROS packages.

In the context of ROS, nodes can have two roles in message-passing through topics: they can function as publishers and/or subscribers. Subscribers receive a message while publishers send them. For instance, Node A may require information calculated by another node and thus subscribes to a topic named “/pose”. Node B performs the necessary calculations and publishes the results to that topic.



*Figure 13 - Example of a Rqt\_graph output of an ROS execution with Node A subscribing to the same topic Node B is publishing.*

A node can also provide services and actions which have similar functionalities. Both receive requests from other nodes and respond accordingly. However, they differ in execution time. Services are expected to provide an immediate response, while actions may take longer to complete since they handle more time-consuming tasks and offer feedback options to the requesting node.

In this thesis, it was utilized the latest version of ROS 1, ROS Noetic, in a Linux distribution (Ubuntu 20.04 LTS) that was installed within a virtual machine using VMware Workstation 16 Player software.

ROS includes several essential tools. Of particular importance, for this development, are Gazebo, Rviz, and Rqt. Rviz and Rqt are graphical debugging tools that aid in visualizing the system's state. Figure 13 depicts one of the Rqt's many outputs. This figure particular output, "rtq\_graph", is for node linkage graphical debug related to the above-mentioned example. Gazebo, on the other hand, is a simulation engine that embeds real-life physics, enabling the creation and spawn of models of various objects, terrains, and living beings. Gazebo, also, provides features such as the development of plugins for enhanced environment simulation. To create a Gazebo-readable world file, the "SDFormat (Simulation Description Format), sometimes abbreviated as SDF, which is an XML format that describes objects and environments for robot simulators, visualization, and control" [26], is utilized. Additionally, the XACRO file format, also in XML format, is used to do a description of robots.

Multiple open-source packages were utilized in this project, which are enumerated in ANNEX A.

Lastly, it is important to note that to develop new packages in ROS, programming in C++ or Python is required.

## 4. Work Developed

This chapter delves into the development of the robot, as conveyed in the Objectives. A broad range of engineering areas were employed to develop a prototype and transmit the general idea of the system. Within this chapter is described the work developed with the creation of an Operational Concept Description and a gripper for the robotic arm end-effector, the training an AI model to detected litter, and the development of an environment and packages for the ROS simulation.

### 4.1. Operational Concept Description

In this section, an OCD, a system-centric description of the intended users, uses, how the system is intended to be used [3], will be outlined based on the expectations of the stakeholders. The stakeholders include a collective of beach users, the entity that owns the robot, the operators, and the developer company, as they are all interested in the system development. Each stakeholder has specific requirements for the robot, which are as follows:

- The robot is expected to function as a maintenance tool to prevent litter accumulation on the beach, assuming that there is not a significant amount of litter and that it is scattered.
- The robot must possess the ability to identify litter.
- The robot must be capable of catching litter.
- The robot must operate autonomously.
- The robot's size must not exceed an area of 1 m<sup>2</sup> and a height of 65 cm, to avoid becoming hazardous to beach users.
- The robot must be able to operate safely in the presence of beach users.

- The robot must clean 90% in quantity of the scattered litter present within an area up to 250x250 meters, in 7 hours.
- The robot's operation must be simple for the operator.
- Trash/litter max weight of 0.5 kg.

Based on these requirements, a concept of operation, i.e., a description on how to use, was developed. This involved defining certain physical characteristics, as outlined in Table 3.

*Table 3 - Initial iteration for the concept's physical characteristics, aligned with the stakeholders' expectations.*

<b>Mobile Platform</b>	<b>Robotic arm</b>
<ul style="list-style-type: none"><li>– Rectangular shape with length between 100-150 cm, width of 50-100 cm and height not superior to 60 cm (dependent on arm reach).</li><li>– Elevation above the ground of 15-30 cm (dependent on arm reach).</li><li>– A deposit basket with a right trapezoidal prism shape, where the small base has <math>\frac{1}{2}</math> of the mobile platform's length and its width.</li><li>– LIDAR or RADAR sensor with a range of 10 to 15 m for detection of entities moving around the robot.</li><li>– GPS with a compass and IMU for accurate robot localization.</li><li>– LED strips as indicators set in the front and rear of the platform for status and presence warnings.</li><li>– Two emergency buttons, one on each side.</li></ul>	<ul style="list-style-type: none"><li>– 3 or more rotational joints (dependent on mounting configuration).</li><li>– Mounted on the mobile platform's front, <math>\frac{1}{4}</math> of the full length and <math>\frac{1}{2}</math> of the full width.</li><li>– Work area that allows the robotic arm to reach the ground and deposit litter in the deposit basket.</li><li>– End-effector range further than 15-30 cm of the mobile platform's area.</li><li>– A payload superior to 1 kg for tool plus trash.</li><li>– Attachable tool to catch litter.</li><li>– An RGB-D camera for litter detection.</li></ul>

## Work Developed

---

### Mobile Platform

- Embedded interactive touchscreen for user interface and remote access via Wi-Fi.
- Mobility by tracks or 4 wheels with suitable terrain tires and differential drive configuration.

### Robotic arm

Next, is presented a sequence of proceedings that describes how a normal operation would occur:

1. The operator transports the robot to the designated operation area.
  - a. The robot should only operate in situations with a clear view of the surrounding environment, avoiding conditions such as fog or sunlight directed to the camera hampering the camera's visibility.
2. The operator powers on the robot.
  - a. The robot starts with its robotic arm in a retracted position.
  - b. The robot has 4 modes which an operator can select and switch between: IDLE, TELEOPERATE, SWEEP, and MAP.
    - i. IDLE mode, the robot stops with the brakes on.
    - ii. TELEOPERATE mode, the robot can be remote controlled by an operator and driven using a RC controller.
    - iii. MAP mode is a mean of operation, in which is given to the robot a set of coordinate points of litter or litter clusters, called a map of litter, for the robot to move to.
    - iv. SWEEP mode is another mean of operation, where the robot performs a linear sweep of a configurable number of meters from the starting position and passes. The sweep can be performed in an area up to 250 meters in length and width. The passes are defined by width divided by number of passes.

- c. Initially, the LED strip indicators on the robot emit a flashing blue light, indicating that the robot is in IDLE mode.
      - d. When in TELEOPERATE mode, the LED strip indicators change color according to the movement:
        - i. White lights, signal the front of the robot.
        - ii. Red lights, signal the rear of the robot.
        - iii. A sliding-blinker orange light, on the end of each LED strip, signals change of direction in which is the robot turning.
  3. The operator switches to TELEOPERATE mode to drive the robot to the desired area.
  4. The operator positions the robot at a starting point, selects the desired mode of operation using the embedded touchscreen.
    - a. If there is no map of litter available, the operator selects SWEEP mode.
      - i. If in SWEEP mode, the robot should be placed, by the operator, parallel to the shoreline.
    - b. If a map of litter is available, the operator selects MAP mode.
      - i. In MAP mode, the operator provides a list of coordinate points of previous identified litter into the robot's on-board PC prior to the operation, using a USB configured to inject the map or Ethernet connection with a file transfer protocol.
  5. The operator supervises the robot's operation and can stop the operation using the remote controller, interacting with the embedded touchscreen or by using one of the emergency buttons.
    - a. The robot exhibits a behavior of attraction towards identified litter while avoiding obstacles detected by the LIDAR or RADAR, replanning the path if possible.
  6. Depending on the selected mode, SWEEP or MAP, Table 4 describes their actions:
-

## Work Developed

---

Table 4 - Description of modes and their operations

<b>MAP</b>	<b>SWEEP</b>
<ul style="list-style-type: none"><li>• The robot moves to the closest coordinate point marked as litter.</li><li>• Upon arrival, the robot stops and scans the area with the camera to identify trash.<ul style="list-style-type: none"><li>○ It extends the robotic arm vertically up to 85% of its maximum reachability and rotates looking for the existence of litter in a radius of up to 4 meters.</li></ul></li><li>• In case of litter identification, the robot uses the arm to catch the identified litter object.<ul style="list-style-type: none"><li>○ The robot places the collected trash in the deposit basket.</li></ul></li><li>• In the absence of garbage, the robot proceeds to retract the arm and moves to the next closest point in the list.</li></ul>	<ul style="list-style-type: none"><li>• The robot starts sweeping with a range up ideally 5 meters of itself.<ul style="list-style-type: none"><li>○ The robotic arm extends vertically to 30-45% of its maximum reachability, pointing the camera towards the front of the robot and the ground, forming an angle of <math>-30^{\circ}</math> horizontal.</li></ul></li><li>• The robot stops sweeping if it identifies litter within the sweeping line and moves towards it.<ul style="list-style-type: none"><li>○ The robotic arm tracks the trash keeping it in the center of the camera's frame.</li></ul></li><li>• The robot parks adjacent to the trash to pick it up with the robotic arm.</li><li>• The robot places the collected trash in the deposit basket.</li><li>• The robot returns to the path line it followed previously and keeps sweeping.</li></ul>

7. The robot concludes the operation under the following conditions:
    - a. The operator interrupts the operation.
    - b. In MAP mode, the robot has traversed all the specified coordinates.
    - c. In SWEEP mode, the robot has completed the sweep of the entire allowed area.
  8. Upon completing the operation, the robot displays a notification on the touchscreen and changes its LED indicators to a flashing green light.
-



#### 4.1.1. Robot End Development

During the development, there were encountered some obstacles that impeded the realization of the complete OCD's outline, so some aspects were changed or even upgraded:

- During TELEOPERATE mode, the operator can choose to do litter detection. This adds the possibility of driving the robot to a place that was not identified or that was missed.
- The RGB-D camera was moved from the robotic arm attachment to the platform's front. The decision was made after exploring possibility of acquiring a collaborative arm and realizing that more affordable options had limited payload characteristics.
- The camera was set with a FoV of 90°, 640x480 image resolution, due to processing capacity.
- Instead of returning directly to the path line, the robot will trace a path to a waypoint at the end of the linear sweep. This means that after catching a litter that is further form the initial path, the robot would not move parallel to the shoreline.

## 4.2. End-effector Tool

Dragging tools, as mentioned in the State-of-the-Art chapter, were not suitable for this robot due to the eventual requirements needed to deal with the forces related to such tool, and their inefficiency in scenarios with spatially scattered litter. So, it was chosen to use a robotic arm due to the pick-and-place potential and a tool to attach to it for capturing litter was considered.

A claw mechanism was explored as a means of collecting litter. This type of mechanism causes evaluation on topics such as determining the appropriate positions

## Work Developed

---

and orientations for catching objects, devising a strategy to avoid collecting sand, and defining the claw's characteristics.

Several designs were created and experimented in the simulation environment. There were designed 3 mechanically different tools. The 3D renders of the first two designs are presented below:

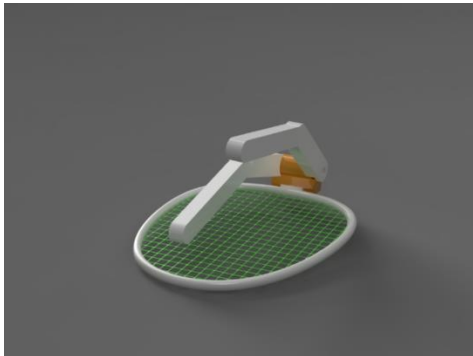


Figure 14 - 3D render of litter catching tool design A.

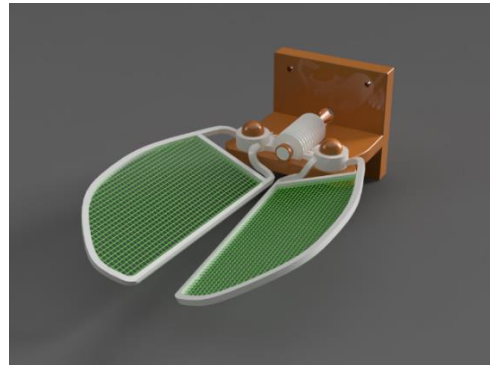


Figure 15 - 3D render of litter catching tool design B.

Figure 14 presents a design with the dimensions to hold objects of 10 cm in size, a mesh net with a spacing of 1.87 mm to allow sand to flow through while collecting trash, and innovative finger-like actuators for gripping and securing the collected litter against the tool's net. The design assumes that the tool's lower frame would penetrate the sand at an entry angle, lifting the litter above the ground and filtering out the sand. The two "fingers" would provide additional gripping options for specific objects.

During the design, it was realized that it would be complex to operate as would have multiple actuators control and be litter orientation sensitive. Next, Figure 15 exhibits the subsequent iteration of the previous tool design. This iteration eliminates the two finger-like actuators while retaining the mesh net. This design proposes a new approach to collecting objects using a single actuator. The tool functions as follows:

1. The tool opens its scoops.
2. The tool is positioned at the surface level or pierced into the sand at a low angle, with the object to be collected held between the open scoops.

3. The scoops are closed, capturing the object, and allowing excess sand to filter out.

This design offers several advantages. As mentioned before, it requires only one actuator, making it lighter and less complex than the previous design. Additionally, it was designed a version that allowed the mounting of an RGB-D camera on top of the tool presented in ANNEX B.

Nonetheless, there are some drawbacks to consider on the design of Figure 15. Firstly, the design requires the robotic arm to have a workspace capable of reaching ground level. Secondly, scoops format may pose challenges in maneuverability and, hypothetically, allowing the captured material to escape through the sides.

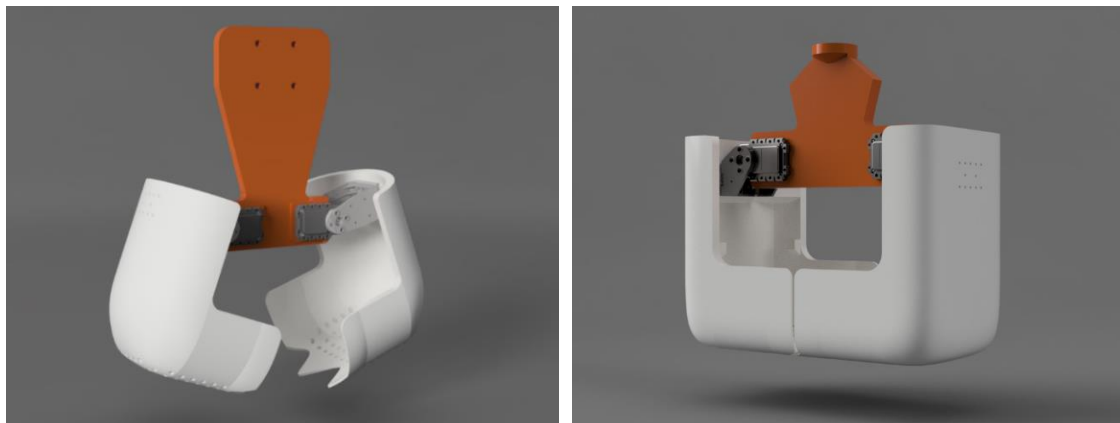


Figure 16 - 3D render of litter catching tool design C v1. Figure 17 - 3D render of litter catching tool design C v2.

Finally, with some guidance of Mechanical Engineer João Lourenço, the designs in Figure 16 and Figure 17 were achieved. These present a whole new style of operation. Unlike the two last designs (in Figure 14 and Figure 15), these ones operate perpendicularly to the ground. They are composed of a linking piece and two scoops. The first was made with the purpose to connect the robotic arm and support the servo actuators. The latter was designed with two large, 4-millimeter potholed scoops, with 2-3 millimetric thick lowered teethed, to better secure, caught, and transport the trash collected, and at the same time let sand flow through. Version 2 of this design has more define lines to aid the printing process. The orientation to attach to the robotic arm was,

---

## Work Developed

---

also, changed to take advantage of full rotation of the end-effector joint in the UR5 robotic arm from Universal Robots, consequently improving the workspace.

To help comparing any of the versions of design C (Figure 16 and Figure 17) with design B (Figure 15), Table 5 highlights the respective advantages and disadvantages:

*Table 5 - Pros and cons of design C related to B.*

<b>Pros</b>	Better litter secureness cause by the shell format.
	No need to make a piercing movement, i.e., have an entry angle related to the ground and bury part of the tool in the sand.
	Less manipulation complexity as the axis of operation matches the arm's end-effector.
	Augmented robotic arm workspace due to the length of the piece connecting the end-effector to the servos.
<b>Cons</b>	More weight, as scoops are more massive.
	Two actuators, one for each scoop.

The tool is actuated by two servo motors, with a bracket attached to the scoops using M2 screws and nuts. To assemble the tool, the scoops are slid in from the top. To be able to print, the tool was divided into five pieces to fit the printer's heated bed. The scoops themselves were split into two pieces.

In ANNEX B there is design drawings with dimensions of all the tool designs developed and the deposit basket.

When printing, it was requested assistance for the process to the Mechanical Engineer João Lourenço. The outcome is presented in Figure 18. The imperfection seen on the images lies on misconfiguration between the extrusion nozzle and its multiplier parameter that states how much material is deposited.

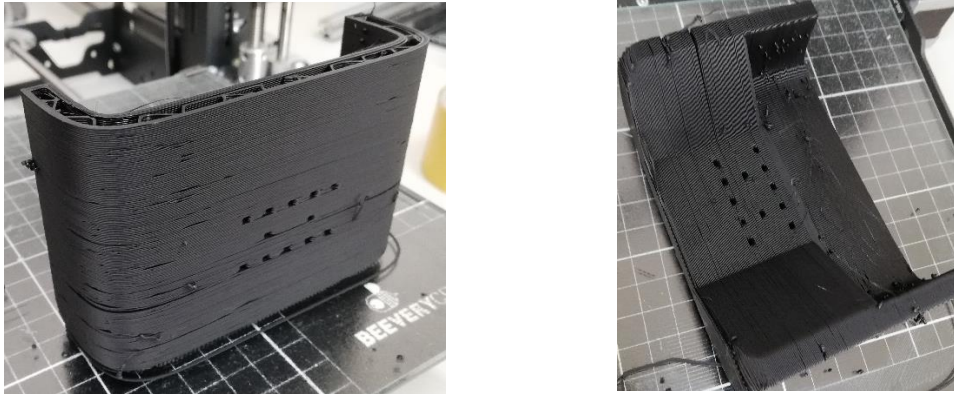


Figure 18 - One piece of the end-effector tool mid printing process.

### 4.3. Identifying Litter with AI in Real-world Coordinates

During the development process, the use of AI for litter identification was deemed the most suitable approach. Image recognition, utilizing cameras and AI computer vision algorithms, is proved to be effective in achieving high accuracy. Training a model was considered, but the availability of a large dataset for accurate training and finding one suitable, that is, with mostly sand backgrounds with trash pictures, was not much.

#### 4.3.1. TACO dataset and TensorFlow

To use the TACO dataset for model training, one must download it from the GitHub repository and follow the instructions, which will subdivide the dataset images into batches of train, validation, and test sets. Dependencies were installed, including TensorFlow, a machine learning framework, and the necessary GPU support. Some compatibility issues were encountered during the usage of TACO and TensorFlow with the code *detect.py* available on the repository.

The experimentation of training a model with TACO and TensorFlow, resulted in a training of approximately 30 hours for 100 epochs of 1500 images with a mix of bounding boxes comprising between an area of 32x32 pixels and 96x96 pixels. The

## Work Developed

---

results on the inference on 10 test set images were unsatisfactory, as shown in Figure 19.

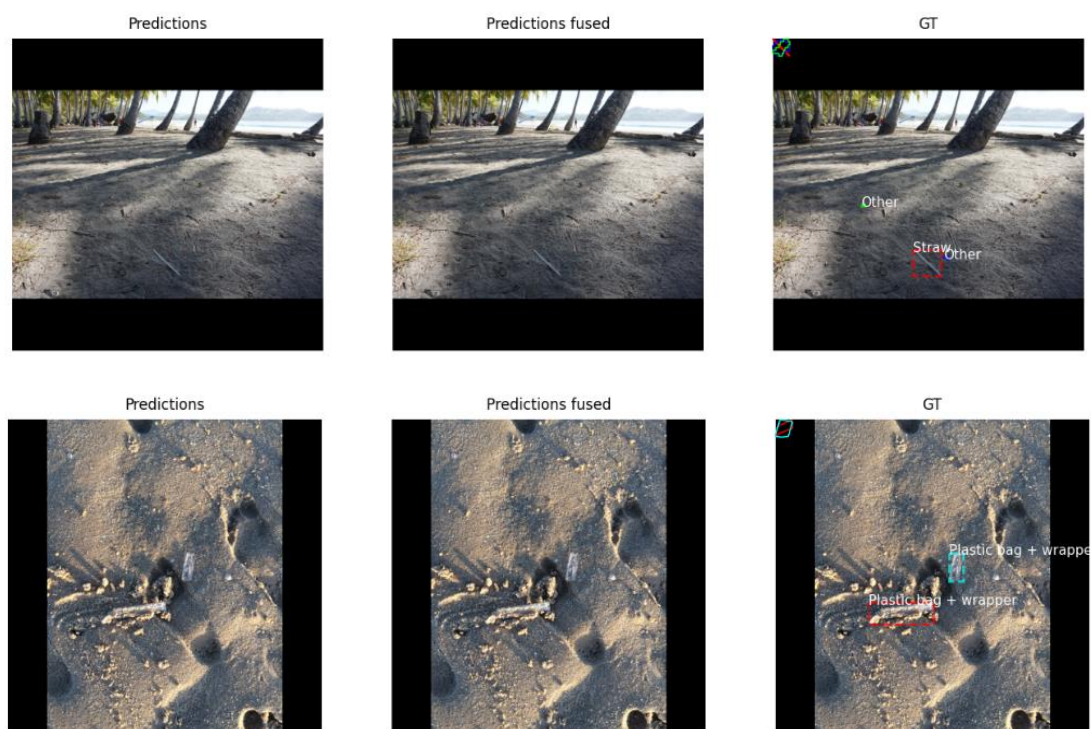


Figure 19 - Results from testing the TACO dataset after patching the code.

There were no predictions and the ground truth (GT) labeled images show displaced outlines in the upper left corner. This outline should be delimiting the object to detect. It became evident that using the TACO dataset with the provided *detect.py* code was not the approach to train a model due to version mismatches between the software and newer hardware.

### 4.3.2. YOLO and PyTorch

YOLOv7 and PyTorch, are widely popular for image recognition tasks. Using the website Roboflow, a new dataset was found. Additionally, Roboflow offered a Hosted API (Remote Server) for inference integration, which was used in the first stages of this

thesis. The Hosted API allowed for testing and understanding the process of video inference.

The dataset utilized for training the model, available on Roboflow [27], consists of approximately 2400 annotated images with a single class "trash". This binary classification resulted in a less precise model when encountering new objects, as it would classify them as trash despite potentially low confidence.

As the development progressed, it became apparent that relying on the hosted video inference API was a not the best solution due to two main reasons:

1. It would require the robot to have a constant internet connection. To address this, a SIM card router was considered, allowing the robot to connect to public Wi-Fi networks if available or use mobile data. This solution limited the robot's usage to areas with good cellular signal or access to public Wi-Fi networks.
2. The video inference using the hosted API introduced significant latency of about 10 seconds between the live feed and the results. It is unclear if Roboflow API is to blame because they affirm "not to worry about the edge device's hardware capabilities as they automatically scale the API up and down and do load balancing". Plus, the usage of the API in a virtual machine which, if not well tuned can throttle the efficiency of a PC's resources usage, may favor this issue.

To overcome these limitations, it was downloaded the same dataset used for the model in the API and it was trained a model locally, instead of online where didn't have the capability of downloading it for usage anywhere else than Roboflow. This approach eliminated the need for an internet connection to perform video inference.

Training the model locally required some tools and libraries already used with the TACO and TensorFlow experiment, like Conda, CUDA toolkit and cuDNN, with the addition of the PyTorch module with GPU support. The training process took approximately 2-3 hours to complete for 215 epochs.

## Work Developed

---

Figure 20 displays the end results of the model's training, with a particular focus on the Mean Average Precision (mAP), which is a metric used to evaluate the model's performance. The figure shows the value of mAP 0.95 to be 0.46 which indicate that the model's performance is not the best, as the closer to 1 the better.

```
wandb: Run history:
wandb:   metrics/mAP_0.5
wandb: metrics/mAP_0.5:0.95
wandb:   metrics/precision
wandb:   metrics/recall
wandb:   train/box_loss
wandb:   train/cls_loss
wandb:   train/obj_loss
wandb:   val/box_loss
wandb:   val/cls_loss
wandb:   val/obj_loss
wandb:   x/lr0
wandb:   x/lr1
wandb:   x/lr2
wandb: Run summary:
wandb:   metrics/mAP_0.5 0.67771
wandb: metrics/mAP_0.5:0.95 0.46783
wandb:   metrics/precision 0.75371
wandb:   metrics/recall 0.60816
wandb:   train/box_loss 0.02504
wandb:   train/cls_loss 0.0
wandb:   train/obj_loss 0.01021
wandb:   val/box_loss 0.0449
wandb:   val/cls_loss 0.0
wandb:   val/obj_loss 0.02998
wandb:   x/lr0 0.0001
wandb:   x/lr1 0.0001
wandb:   x/lr2 0.0001
```

Figure 20 - Model training log using YOLOv7.

Furthermore, using this model in the simulation for litter detection, it was concluded that the position, orientation, and characteristics of the camera, such as the field of view (FoV), had an impact on the model's prediction confidence. The dataset's images suggested that the model performed best when the camera has an aerial view, as most of the dataset's images were captured from this perspective. And, since the camera was mounted on the front side of the robot with a low pitch angle, the results for litter detection were not consistent. The light effects, like the shadow casting, over the litter pieces also contributed for such result.

Additionally, even when performing video inference locally, there was still a latency of approximately 1-3 seconds between the live feed and the processed feed. This latency could be attributed to several factors, including the computational load of running the inference on the virtual machine with no GPU passthrough.



### 4.3.3. Pixel to World Coordinates

Once the trash is identified and its bounding box's center pixel coordinates are obtained, the technique of transposing pixel coordinates into real-world coordinates, described in the Computer Vision Tools section was used. Figure 21 illustrates the inputs and outputs of an example using the YOLOv7 detection algorithm pre-train with a model called yolov2.tiny that could detect and classify persons:

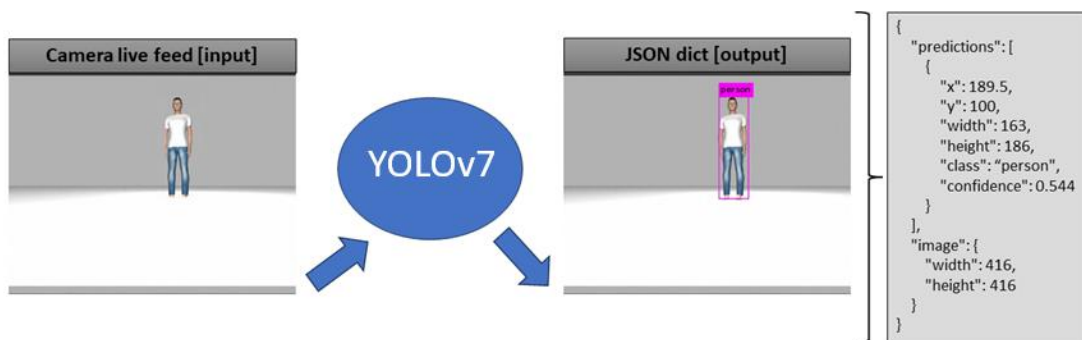


Figure 21 – YOLOv7 inputs and outputs illustration.

In the class “prediction”, “X” and “Y” mark the center pixel coordinates of the bounding box and “width” and “height” states its sizes. The other parameters tell of the class predicted and level of confidence.

Then, to transform those coordinates into 3D coordinates, it was used the Intel RealSense D435 RGB-D camera gazebo plugin for ROS camera emulation, the library provided by the camera producer, and a script developed in Python that utilizes the ‘rs2\_deproject\_pixel\_to\_point’ function to obtain 3D coordinates.

By combining the image recognition capabilities of the trained model with the ability to convert pixel coordinates to real-world coordinates, the system could accurately identify and locate litter objects in the simulated environment, as shown in Figure 22.

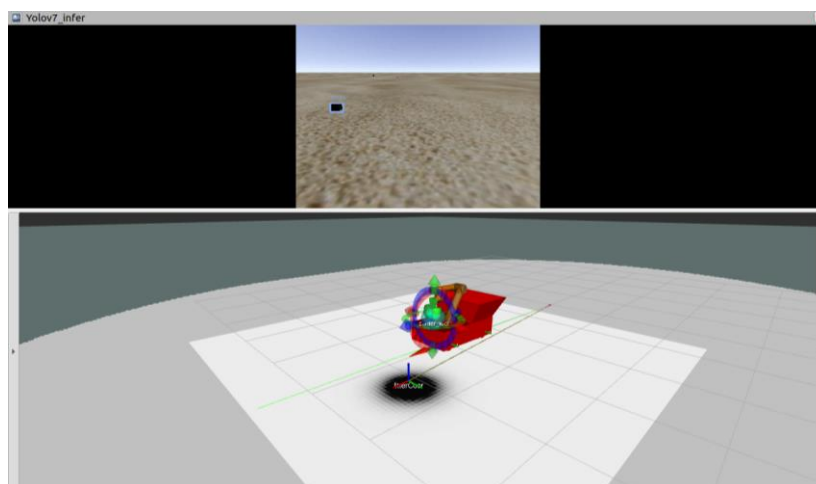


Figure 22 – Rviz screenshot of video inference and the litter's position. Above is the video inference and below a debugging window with the robot description and transforms.

## 4.4. Development and Integration in ROS

ROS plays a vital role in the development and integration of this project. It serves as the framework for various key components, including path planning, robotic arm control, video inference, drive control, and other necessary processes for the operation. Additionally, ROS facilitates the creation and execution of a simulated environment for testing and validation purposes.

### 4.4.1. Environment and Robot Building

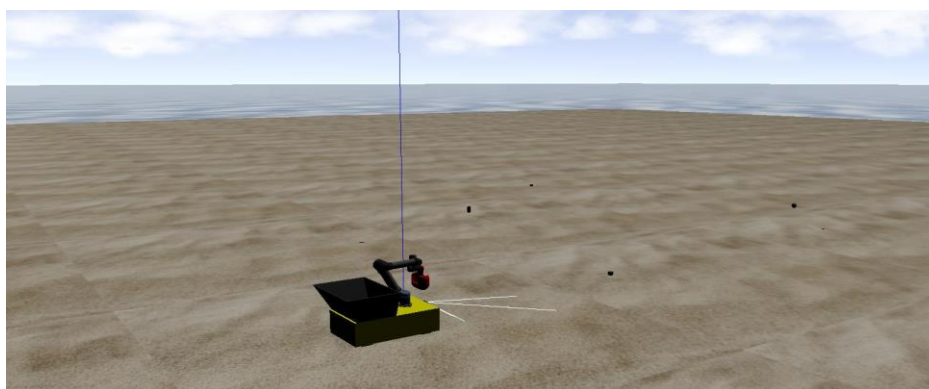
The development of the environment and the robot was assisted by Gazebo. To achieve an approximation of a real-life beach, the model editor in Gazebo was utilized. Objects representing litter on the beach, shown in ANNEX C, which were used for image recognition, were created, and textured using this editor. These objects were placed on top of an invisible step, which was just a collision box, with 8 cm height. To cover the objects elevation, a visual layer of the sand floor was placed. The reason for this elevation was that the ground, in the simulation, is solid and does not allow the tool to interact/collide with it as it would in a real sand environment, causing glitches.

Another approach was explored to address this limitation. The attempt involved making the objects buoyant using a buoyancy plugin and applying that same visual layer of sand. The idea was to have the objects floating in the visual layer as the tool would collect the object with no collisions with the rigid floor. However, this approach did not have the expected effect, as some objects ended up with incorrect orientations as they would if the object were settled in the floor when affected by the plugin, as shown in Figure 23.



*Figure 23 - Litter affected by the buoyancy plugin on Gazebo with vertical orientation instead of horizontal.*

To enhance the visual realism of the simulation, a plugin called “asv\_wave\_sim” was employed. This plugin introduced a sea and mimicked its movement. Figure 24 showcases the outcome of the environment development, highlighting the visual representation achieved in Gazebo.



*Figure 24 - Gazebo world with clouds and waves for realism. Robot ready to catch the scattered trash.*

For the sand beach in the simulated environment, initially a hilly terrain with a slope by the seashore was used, presented in ANNEX E. Due to irregular physics

---

## Work Developed

---

interactions and some issues with the simulation configuration, ended up in certain physics parameters being turned off and it was decided to use a regular horizontal plane as ground. This provided a more stable and consistent environment and results for the robot's operation in Gazebo.

The script in SDF format included in ANNEX D provides a detailed explanation of the parameters used to define the world in Gazebo. It specifies the characteristics of the terrain and other environmental elements.

The robot itself was described using the XACRO format, which contains the necessary descriptions for the robot's base, deposit basket, camera, and robotic arm, in accordance with the OCD. These components are readable by Gazebo, allowing the robot to be spawned and interact with the simulated environment. The description file includes information about joint and link relations, visuals, and physics properties of the robot.

The robot's base has dimensions of 100 x 60 x 25 cm and is equipped with four wheels measuring 10 cm in width and 12.5 cm in radius. The deposit basket, positioned on top and  $\frac{1}{4}$  of the back length of the base, has approximate dimensions of 40.5 x 54 x 17.9 cm and can hold approximately 14.6 liters of trash. An Intel RealSense D435 camera is mounted on the front of the robot's base, while a UR5 robotic arm is positioned on the top and  $\frac{1}{4}$  front of the base. Detailed specifications of these components can be found in ANNEX F.

The file descriptors for the UR5 robotic arm and Intel RealSense camera were obtained from their respective GitHub repositories. Additionally, a Gazebo plugin for the camera was provided by PAL Robotics S.L.'s GitHub repository.

The placement of the camera on the robot was constrained by the workspace of the robotic arm. Due to limited options, it was positioned in the front, which dampened the efficiency of image recognition.

It is important to note that the dimensions and shapes chosen for the robot and its components are not directly related to specific physical characteristics. They were selected to be reasonable within the simulated environment and aligned with the OCD.

For example, the rectangular box shape of the robot was chosen to facilitate the simulation process. And, thus, the robot was baptized as “Boxbot”.

#### 4.4.2. Developed ROS Packages

In this section, it will be enumerated and described the functionality of the ROS packages developed and how they interact with one another (Figure 25) to accomplish the goal. In the figure, green arrows are the hardware inputs and the red arrows are the outputs. The black arrows mark the message trading with the specified topic. ANNEX G presents an overview of nodes and topics that run within ROS. It is important to address that only the SWEEP mode, planned in the OCD, was developed.

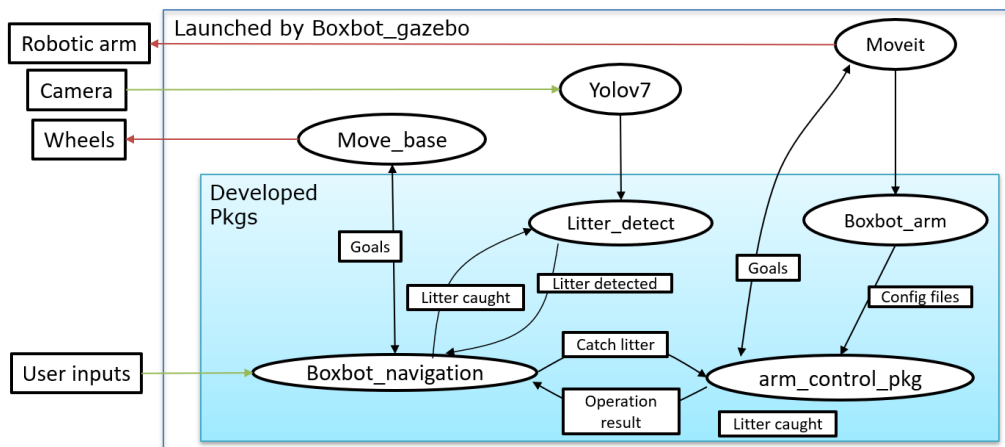


Figure 25 - Packages relation diagram.

##### 1) Boxbot\_gazebo:

The "Boxbot\_gazebo" package serves as a beacon package, i.e., responsible for launching all the necessary nodes and calling other pre-existing launch files from different packages. This approach helps streamline the simulation process by saving time and consolidating all the information in a single launch file for simplicity.

## Work Developed

### 2) Boxbot\_navigation:

This package only contains one node named “boxbot\_path\_planner”. This node is responsible for path planning and managing other actions of the Boxbot, explained below. Table 6 lists the node’s ROS topics and ROS params:

Table 6 - List of ROS computation processes used in the “boxbot\_path\_planner” node.

<b>Subscribed</b>	Signal bool of publish pose of litter	Odometry	Boxbot’s velocity	Boxbot’s mode
Topics	/litter/pose	/odometry/filtered	/cmd_vel	/boxbot/state
<b>Published</b>	State bool of the litter to inform that was it cached	Boxbot’s mode	Signal bool of caching	
Topics	/litter/status	/boxbot/state	/boxbot/catch	
<b>Parameter</b>	Maximum range of path for accepting litter pose	Number of divisions within the sweep area	Fixed referential frame	Area to sweep (length, width) Parking distance

Initially, the Boxbot's mode is selected through external input streamlined to the topic listed as “/boxbot/state” in Table 6, allowing the user to choose from various modes such as starting the sweep, stopping movement, going to the initial pose, entering teleop mode, or setting the robotic arm to a contracted configuration.

In sweep mode, the planner generates linear routes, through waypoints and that avoids obstacles, based on the specified parameters of area to sweep and the desired number of divisions. This divisions sets the width and number of traversals in the area to sweep. By sending a request of the waypoint goal to the move\_base node, which is responsible for navigation. The route is planned, and motion starts. When a signal indicating the pose of a detected litter object is received through the topic “/litter/pose”, the system verifies if the object's position is within the maximum range parameter of the planned path, defined by the operator, and checks the current mode of the Boxbot. If the object is within range and the Boxbot is in SWEEP mode, a new goal is published

to move towards the position of the litter object and park at a define parking distance, by the configurable parameter, of it. On arrival, the system then publishes to the topic *"/boxbot/catch"* to stop litter detection and requests, with a string to the *arm\_controll\_server* node, the desired pose for the end-effector to catch the litter, waiting for the result. Upon successful completion, the node publishes to the topic *"/litter/status"* that the litter has been caught and Boxbot returns to perform the sweep.

In the case of the teleop mode, Boxbot does not follow a route and it tele operated by an operator. This mode the operator can chose to do litter detection or not. Upon detection, the node checks if the robot is within the parking zone using odometry data and verifies if Boxbot has come to a stop based on its velocity. If these conditions are met, the node initiates the *arm\_controll\_server* sequence to actuate the robotic arm and catch the litter detected.

The latency issue associated with the video inference in the *yolov7\_ros* node, mentioned in chapter 4.3.2, has a ripple effect on the execution of the Boxbot system, impacting the path planner node. During the SWEEP mode, when a detection is received, the Boxbot's next operation is to measure the depth of the center pixel. However, due to the time it takes for the detection, there was an offset between the received image and the live feed depth data image, which results in a displacement in the location of the objects. Consequently, the Boxbot gets this goal incorrectly.

To address this problem, a validation step was introduced. When a detection is received, the Boxbot moves backward to get the object in the FoV of the camera. It then pauses for a few seconds to allow the video inference and the live feed to align properly. After this synchronization, the Boxbot resends the goal to the correct location for the detected object. Upon reaching the destination, to validate the accuracy of the litter positioning, Boxbot performs an additional location goal.

By incorporating this validation step, it is mitigated the issue caused by the latency in the video inference, ensuring that the objects are correctly detected and localized.

## Work Developed

---

### 3) litter\_detect:

The “litter\_detect” package includes various early test scripts that were created during the development, like the hosted API video inference from Roboflow to detect litter, and the robotic arm movement controlling through the moveit library. As the development progressed, a node with the same name as the package became responsible for managing the detected litter objects. Table 7 lists the node’s ROS topics and ROS params.

Table 7 - List of ROS computation processes used in the “litter\_detect” node.

<b>Subscribed</b>	Detection results from yolov7_ros	Depth camera data	Litter’s state bool	Signal bool of caching
Topics	/yolov7/yolov7	/camera/depth /image_raw	/litter/status	/boxbot/catch
<b>Published</b>	Pose of litter	Signal bool for publish pose of litter		
Topics	/tf	/litter/pose		
<b>Parameter</b>	Confidence threshold	Fixed referential system		

The “litter\_detect” node performs a sequence of several tasks. Firstly, when the catching flag is set to false, received via the topic “/boxbot/catch”, and results of the detected trash objects, received via the topic “/yolov7/yolov7”, are above the confidence threshold, the node proceeds to convert the detected objects bounding box center pixel coordinates into a corresponding 3D world position, by using the ‘rs2\_deproject\_pixel\_to\_point’ function and then make a transformation to the fixed referential system, in this case the transform named “map”. This conversion relies on the depth camera data and follows the method described in 3.2.3. Then it stores the correspondent litter’s 3D world pose into a FIFO (First-In-First-Out) list. Secondly, while there is any element on the list, the node publishes the litter’s pose via topic “/tf.” It also publishes a signal through the topic “/litter/pose” to notify other nodes about the publication of the litter's pose. The node publishes another element of the list when it



is confirmed that the current litter has been collected, signaled via the topic *"/litter/status"*.

#### 4) **arm\_controll\_pkg:**

This package is constituted by a node named *"arm\_controll\_server"*. This node provides ROS action and is a wrapper around the ROS actions provided by the *moveit* node, meaning that works as a driver between the *moveit* controller package and the requested actions of the *"boxbot\_path\_planner"* node. Table 8 lists the node's ROS topics and ROS params:

*Table 8 - List of ROS computation processes used in the " arm\_controll\_server" node.*

<b>Request</b>	Named type of operation [ <i>litterCoor</i> , <i>home</i> , <i>collected</i> , <i>up</i> , <i>deposit</i> , <i>closed</i> , <i>open</i> ]		
<b>Response</b>	Operation status [ <i>success</i> / <i>failure</i> ]		
<b>Parameters</b>	End-effector pose offset	Fixed referential frame	Z-axis offset to litter

This node receives a request goal of the type of operation to perform. The possible types are *"litterCoor"*, *"home"*, *"collected"*, *"up"*, *"deposit"*, *"closed"*, *"open"*.

If the input is *"litterCoor"*, the scoops are opened and the litter's pose is set as a goal for the end-effector, considering the offset of the end-effector and the approach Z offset parameters. The end-effector pose offset parameters is needed because despite the knowledge of the distance to the litter, its depth is unknown. The usage of this offset is to ensure to catch the litter by its center of mass. The Z-axis offset is to set a waypoint, for the end-effector to move to, above the pose of the litter identified, in order to ensure a vertical entry angle for catching the litter. It is requested that goal to the *moveit* node and a movement plan is created and executed if possible. Right after, the actual pose of the litter is requested for the end-effector to move to and thus, planned and executed.

---

## Work Developed

---

Once the end-effector reaches the requested pose, the scoops are closed, and the robotic arm moves through a series of preset waypoints of joint angles to keep the tool pointed down. Throughout this sequence of events the operation status is constantly reported back and monitored.

The other types of operation for request, are named joints configurations saved in a configuration file which have their respective joint angles, shown in ANNEX H . The last two operations requests, i.e., *“open”* and *“close”*, are associated with the scoops and the others with the arm. The "arm\_control\_server" node manages these types of requests by evaluating to which group, arm or scoops, is directed to then requesting the goal to the moveit node. The moveit node has access to the configuration file and sets joint angles according to the named joint requested and is called every time that is needed to perform a requested operation.

### 5) **boxbot\_arm:**

This package holds the launch files for the execution of the Moveit controller, along with the configuration files obtained through the XACRO files from the "Boxbot\_description" package, allowing seamless integration with the Moveit controller.

The *“boxbot\_arm”* package was created utilizing the "moveit\_setup\_assistant" package. Moveit Setup Assistant simplifies the development process of configuring robotic arms and other joint-controlled robots. By providing a descriptor file of a robot, such as a URDF or XACRO, it generates all the necessary configuration files for the Moveit controller to operate. With Moveit Setup Assistant, it becomes possible to define joint limits, link chains, configure collision checking between links, set predefined joint positions, choose the type of controller (position, velocity, or effort), and configure PID gains, as well as select the planner algorithm for inverse kinematics calculations.

## 6) **Boxbot\_description:**

The "Boxbot\_description" package contains the descriptor files for various components of the Boxbot system. These descriptor files define the properties of the Boxbot, the UR5 robotic arm, the Intel® RealSense™ D435 RGB-D camera, the deposit basket, and the tool attachment for the UR5. Within these descriptor files, properties such as material colors, collision physics, and shapes are defined for each link. The shapes can be represented by either a mesh file (exported CAD files) or simple geometric shapes. Additionally, the package includes Gazebo plugins for various functionalities. These plugins include the IMU (Inertial Measurement Unit), GPS, LiDAR, ROS controller, and libsense for the RGB-D camera plugins.

## 4.5. ROS Simulation

To run the ROS simulation environment, in addition to the installation of the mentioned packages and dependencies listed in ANNEX A, there is the need to compile using the command `catkin_make install` in the ROS workspace directory.

Then one should execute the command `roslaunch boxbot_gazebo demo_gazebo.launch` in a terminal and additionally, in a new terminal window, run the bash script with the command `./gz_setup`. The script sends commands to gazebo to resume the simulation, once it starts paused, and set the gravity to zero. This is necessary to make time for the controller\_manager to initialize the motor drivers. After a few seconds, the script reestablishes the gravity to prevent the arm joints from flailing. When the terminal that executed the roslaunch command pauses printing initialization messages, one must execute another command to activate the SWEEP mode in Boxbot. It should be introduced in the terminal: `rostopic pub /boxbot/state std_msgs/UInt8 "data: 0"`. Using "data: 1" puts Boxbot in standby, interrupting the current task. Using "data: 2" makes Boxbot move to the initial point. Using "data: 3" set Boxbot to TELEOP mode and using "data: 4" forces the robotic arm to the preset position of collected.

## 5. Results

This chapter displays the results obtained concerning the litter detection, litter collection, and overall system simulation. To obtain the results for those subjects, it was used the objects presented in ANNEX C and for that concern the objects are going to be referred, throughout this chapter, as they are labeled in the annex.

### 5.1. Litter Detection

This section shows test's results to the capability of detecting litter. These tests were conducted with Boxbot stopped, an ambient yellow light, the camera with a FoV of 90°, 640x480 image resolution, and with one object at a time. Figure 26 shows such results with a detection confidence threshold of 35%.

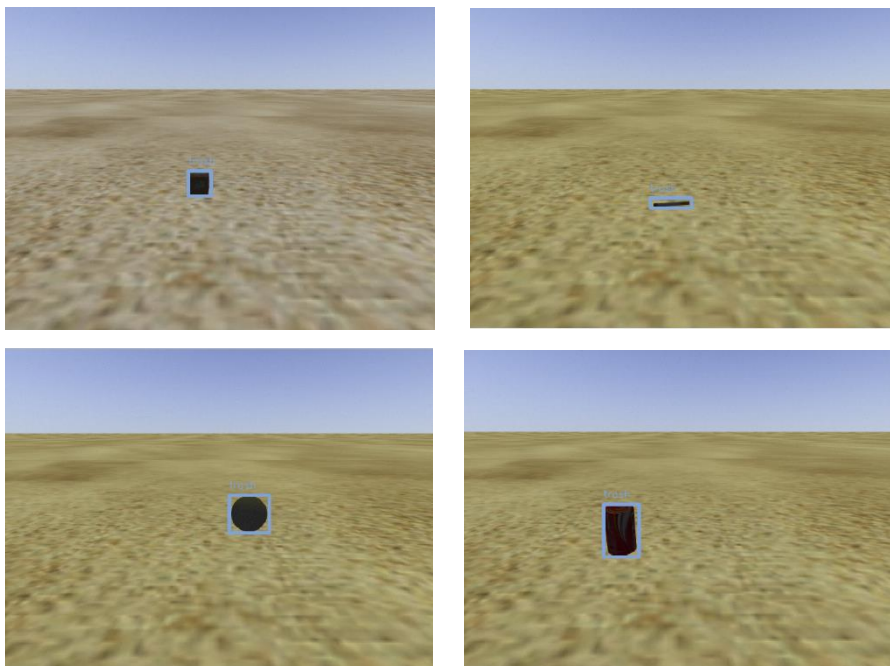


Figure 26 - Detection of objects 1 and 2 in the upper row, and object 3 and 4 last row.

Placing the objects in front of the camera and at a distance below 1 meter, as seen in the images above, all the objects were detected successfully. More tests were conducted to delineate zones of detection with confidence thresholds of 35%, 50% and 85%, using objects 2 and 4, shown in the figures below.

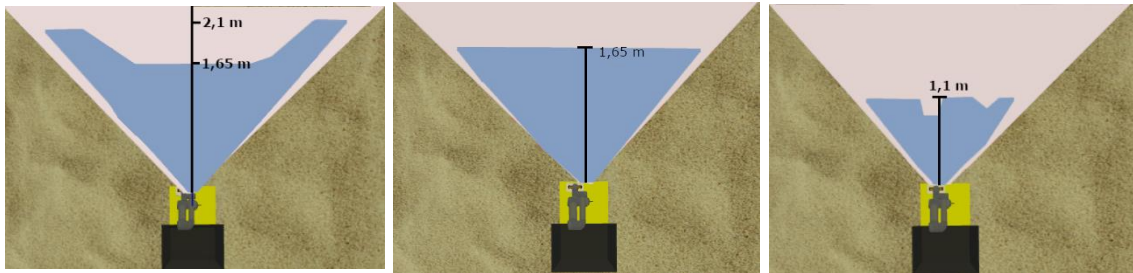


Figure 27 – Detection zone with confidence of 35%, 50% and 85% for object 4.

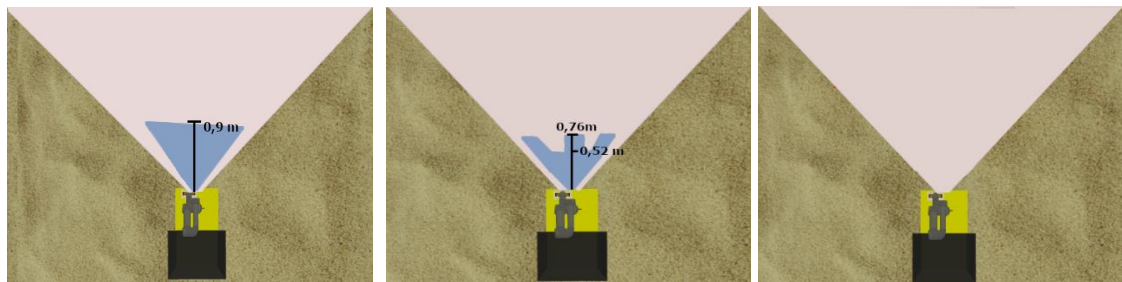
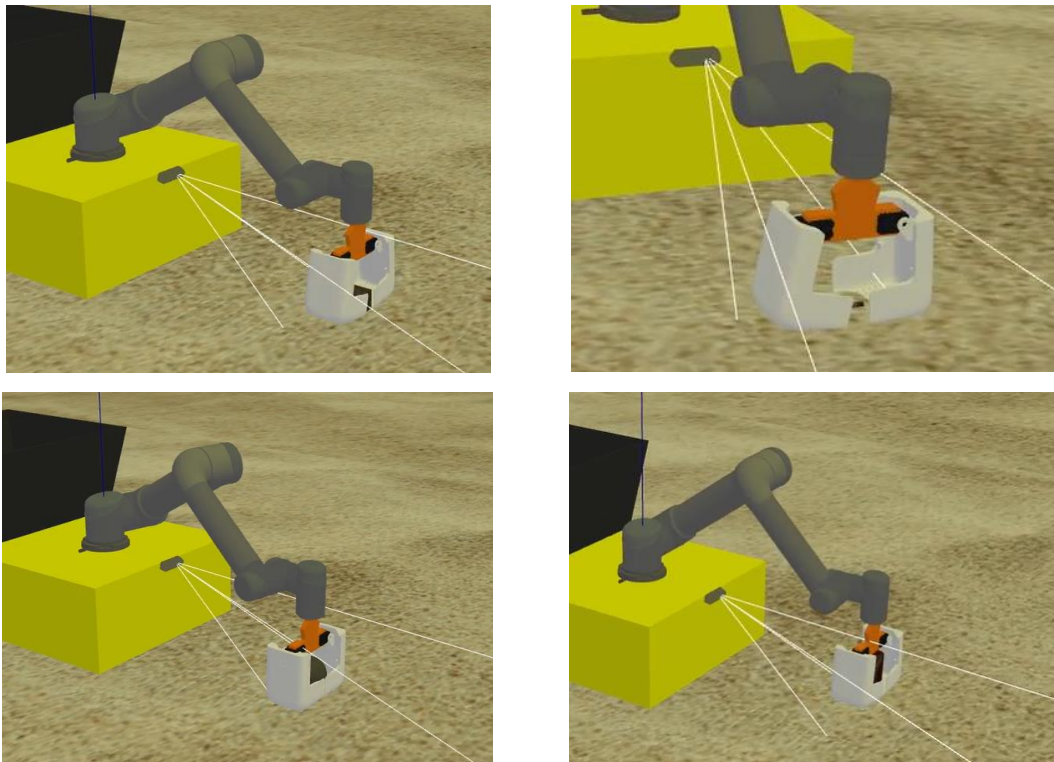


Figure 28 – Detection zone with confidence of 35%, 50% and 85% for object 2.

In white are delimited the image FoV capture, while in cyan is delimited the detection zone for the tested objects. It is concluded that because of object 4 being dimensional larger the distance of detection is higher for object 2. Object 4 gets a maximum range of 2.1 meters with 35% of confidence threshold, 1.65 meters with 50% and 1.1 meters with 35%. Object 2 gets a maximum range of 0.9 meters with 35% of confidence threshold, 0.76 meters with 50% and no detection with 85%. Both results have gaps within the zone due to noise in the detection provoked by shadow effects on the object and camera position.

### 5.2. Litter Collection

The testing of the capability of catching and securing litter, started in the 1<sup>st</sup> version of design C, presented in Figure 16, but was the 2<sup>nd</sup> version of design C, presented in Figure 17, that presented better results for the reasons mentioned in Chapter 4.2. As shown in Figure 29, the end-effector tool was successful on collecting the objects, but object 4 is orientation sensitive and took a second try to catch it. Furthermore, it is concluded that if object 4 is laid flat on its side and with the same heading as Boxbot, it will not be caught.



*Figure 29 - Tool design C v2 collecting objects 1 and 2 in the upper row, and object 3 and 4 lower row.*

These results were obtained with the following parameter values: approach Z offset of 18 cm, end-effector position offset of 2 cm in the X and Z axis, and 1 cm in the Y axis.

### 5.3. System Simulation

Here is going to be presented results of the simulation, i.e., putting in practice the tests done above, adding the motion aspect. These results were obtained in an area of 10x10m divided by 4 traversals, as shown in Figure 30 and Figure 31. Remind that the objects are numbered as its correspondents in ANNEX C.

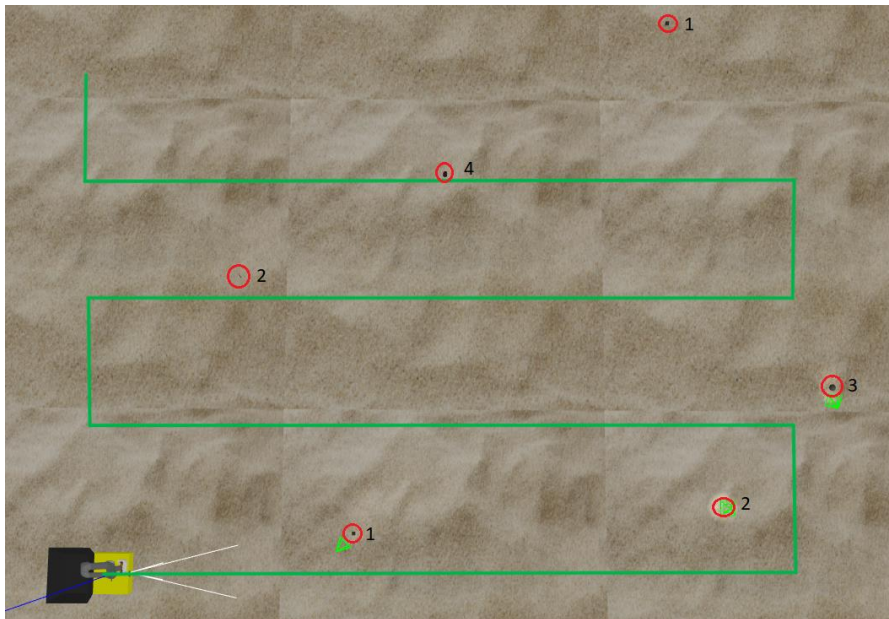


Figure 30 – Boxbot defined sweeping trajectory and trash.

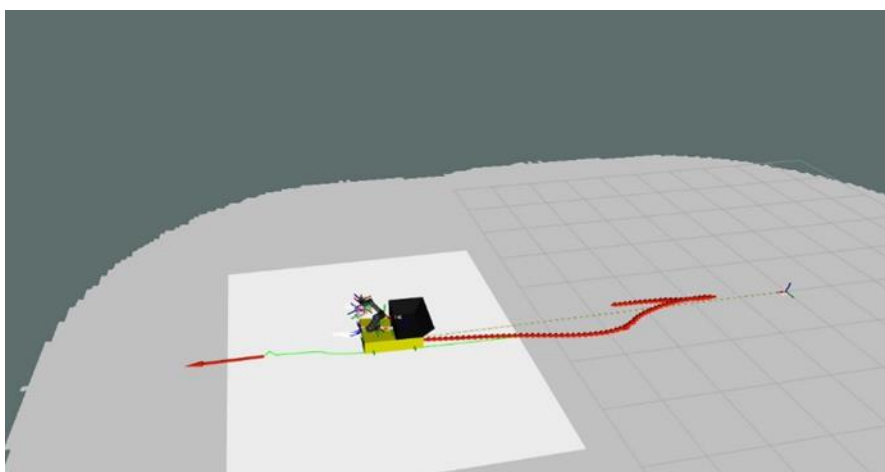


Figure 31 – Boxbot detour from the defined trajectory to catch object 1 resuming the pathing to the defined trajectory.

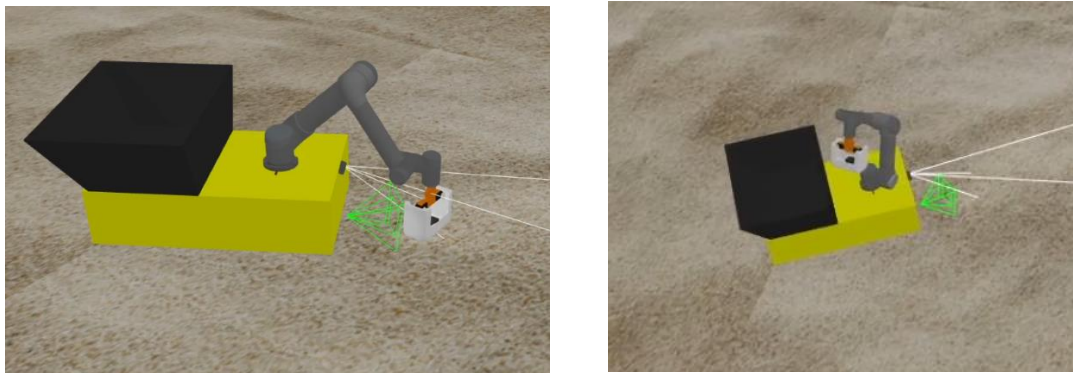


## Results

---

Running the simulation and following the path shown in the figure above, the first object was detected successfully. Boxbot could locate it in the world successfully and after some attempts in moving towards the litter, Boxbot arrived at a parking position adjacent to the object. Boxbot was not always successful in traveling to the point. Sometimes it starts over maneuvering, i.e., moving forward then backwards trying to adjust its orientation. The main cause of such behavior was not being able to achieve the optimal set of parameters for `move_base`, `ros_controllers` and world physics which are correlated. Getting the optimal set of parameters, it's only done by trial and error.

After stopping in the vicinity of the first object, Boxbot was successful in placing the robotic arm above the object for litter collection, opening the scoops, picking the object up, and dropping it the deposit basket, presented in Figure 32, just like intended.



*Figure 32 - Boxbot collecting and depositing the object 1.*

For the following two litter objects, detection failed and therefore there was no litter collection. The first cigarette, labelled with the number 2 in Figure 30, is assumed to fail detection due to being out of range for detection, according to the zoning tests done earlier. The same reason is applied to object 3. In Figure 33 is shown the miss detection of `yolov7_ros`, in the way to pass by objects 2 and 3.



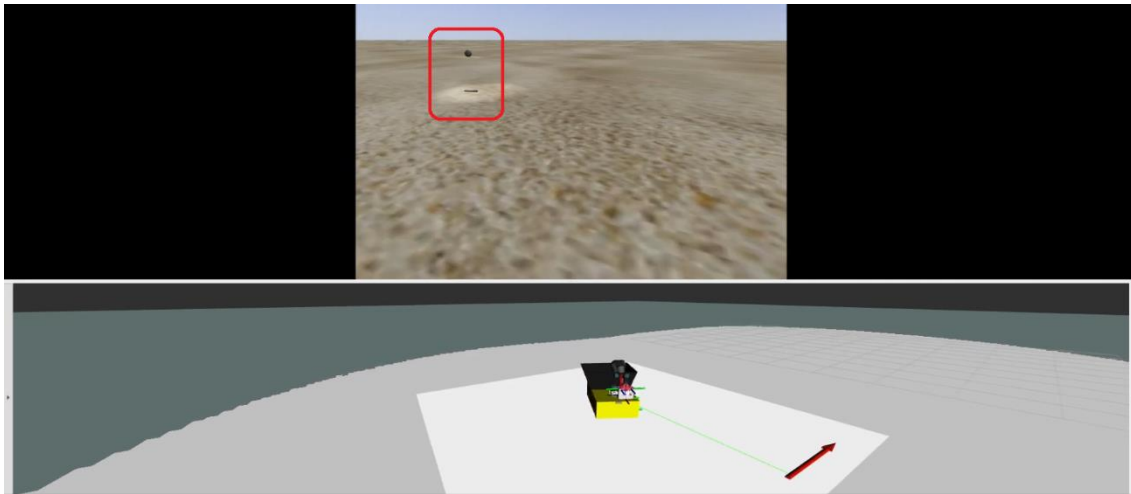


Figure 33 - Rviz image of yoloV7 with no detection returned identified by the red rectangle.

After passing through these two objects, Boxbot detected the second cigarette and located it in the world, as shown in Figure 34, making it replan its trajectory. This happened without much maneuvering making it successful in stopping next to the object. The collection procedure was also completed with success.

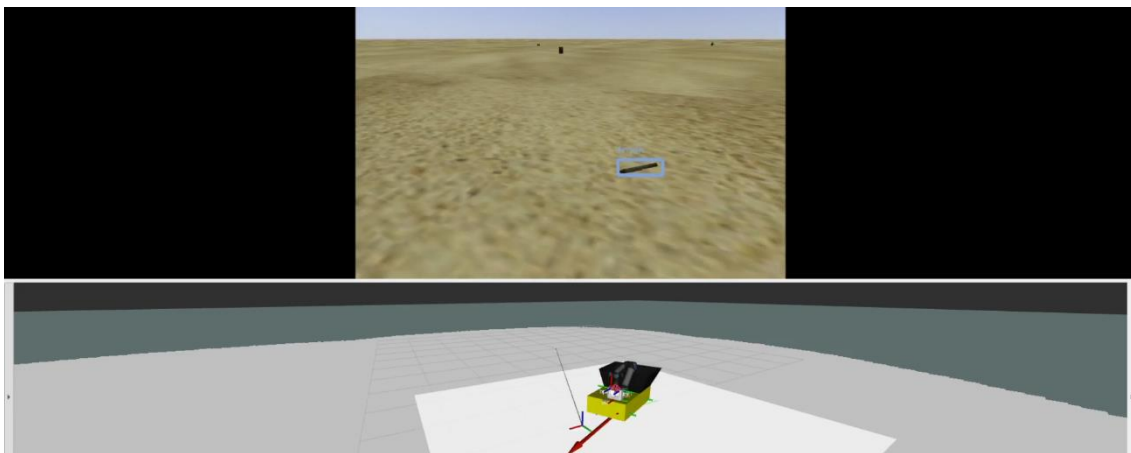


Figure 34 - Rviz visualization of Boxbot detecting and planning parking for the collection of object 2.

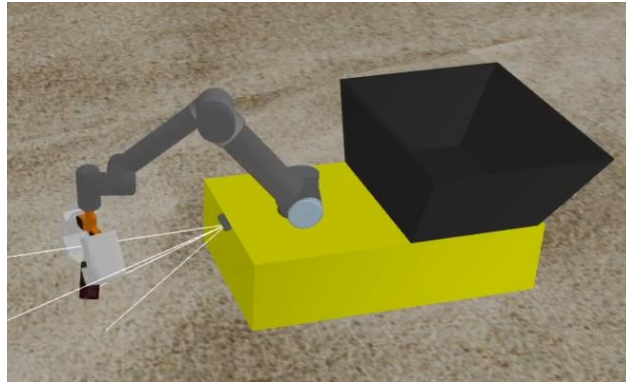
Object 4 was successfully detected and Boxbot started its routine of parking next to the litter. Here, there were multiple attempts on approaching as it starts over maneuvering. Upon parking a flaw was display. The placement offset of the end-

---

## Results

---

effector, i.e., the parameter to place the end-effector above the litter, is fixed and set in the launch of the simulation. This parameter had to be adjusted to avoid the tool colliding with object 4, as the scoops are opening (presented in Figure 35), but this high is uncalled for the other objects.



*Figure 35 - Boxbot trying to collect object 4.*

These results validate the definition of proof of concept, i.e., that the concept and design are feasible. The operation of the SWEEP mode is presented in ANNEX I or by using the [https://youtu.be/HWGezZx22wU?si=GnkmhxO9XSlpTl\\_e](https://youtu.be/HWGezZx22wU?si=GnkmhxO9XSlpTl_e) video link.

## 6. Conclusion

This dissertation aimed to develop a robot for autonomously cleaning beaches by combining AI, navigation, robotic arm control, and ROS. The robot localization and environment perception in beach locations were issues that required research and study to implement a proper solution. Therefore, it was possible to start developing a virtual robot, which led to the realization of a simulation in ROS.

It was understood the need of tools to train a model. Tools like, TensorFlow or Pytorch, libraries for GPU and dataset. The TACO dataset was the best found, but the script provided to train the model was outdated, resulting in a model that did not detect litter. In the search for datasets and hosted inference, Roboflow proved to be a great service in obtaining quick results in this area. The introduction of the YOLOv7 detection algorithm made it possible to train a model based on a given dataset, in a simple and direct way, since this application resulted from a model capable of detecting litter.

In the integration of the robotic arm in ROS, the use of Moveit package is concluded to be necessary in these types of projects, as removes the need to program inverse kinematics and the controllers. Early designs of grippers attached to the robotic arm showed some bad results on moving the end-effector to the goal coordinate. It was concluded that the arm functions better with the gripper having the same vector of operation as its end-effector.

Finally, the use of a virtual machine diminished the PC resources and a disabled the GPU usage which led to a temporal mismatch between video inference and depth data in the ROS simulation using AI. Using a virtual machine (VMware) is the most practical way to install an operating system (Linux) in order to use ROS in project development.

## Conclusion

---

Overall, with the obtained results in the ROS simulation it can be concluded that all the developed work integrated in the simulation works and provides a foundation for further improvements and potential real-world applications.

### 6.1. Future Work

In future work is needed to:

- Improve the image recognition capabilities by enriching the dataset with more classes;
- Perform behavior tests outside of the virtual machine, utilizing GPU support;
- Improving the local move\_base planner parameters, as already described in fifth chapter;
- Development the MAP mode. Program this additional mode that move Boxbot to prior mapped litter positions for collection;
- Development of a prototype to conduct experiments.

## References

- [1] Encyclopaedia Britannica, 30 June 2020. [Online]. Available: <https://www.britannica.com/topic/refuse>.
- [2] Cambridge Dictionary, "Cambridge Dictionary," [Online]. Available: <https://dictionary.cambridge.org/pt/dicionario/ingles/litter>.
- [3] R. Halligan, "What is the difference between an Operational Concept Description (OCD) and a Concept of Operations (CONOPS)?," 4 October 2022. [Online]. Available: <https://www.ppi-int.com/resources/systems-engineering-faq/what-is-the-difference-between-an-ocd-conops/>.
- [4] World Bank, "Trends in Solid Waste Management - World Bank," 20 September 2018. [Online]. Available: [https://datatopics.worldbank.org/what-a-waste/trends\\_in\\_solid\\_waste\\_management.html](https://datatopics.worldbank.org/what-a-waste/trends_in_solid_waste_management.html).
- [5] Agência Portuguesa do Ambiente, "Relatório do Estado do Ambiente 2020/21," 2021.
- [6] Agência Portuguesa do Ambiente, "Lixo marinho em praias - Resultados da campanha 2022," 2022. [Online]. Available: <https://apambiente.pt/residuos/lixo-marinho-em-praias-resultados-da-campanha-2022>.
- [7] International Organization for Standardization (ISO), "ISO 14688-1:2017," *Geotechnical investigation and testing — Identification and classification of soil — Part 1: Identification and description*, December 2017.
- [8] Mike, "Solarino Is A Remote Control Roomba For Picking Up Trash On The Beach," 28 May 2019. [Online]. Available: <https://mikeshouts.com/dronyx-solarino-beach-cleaner-robot/>.
- [9] The Searial Cleaners, "BeBot | The beach cleaning robot," [Online]. Available: <https://searial-cleaners.com/our-cleaners/bebot-the-beach-cleaner/>.
- [10] .BB, "Project.BB," 2023. [Online]. Available: <https://project.bb/>.
- [11] The Ocean Cleanup Technologies B.V., "The Ocean Cleanup," 2023. [Online]. Available: <https://theoceancleanup.com/>.
- [12] IADYS, "IADYS - Interactive Autonomous DYnamic Systems," [Online]. Available: <https://www.iadys.com/>.
- [13] Seabin, "The Seabin Project," [Online]. Available: <https://seabin.io/>.
- [14] "Ambiente Cascais," 20 Novembre 2018. [Online]. Available: <https://ambiente.cascais.pt/pt/noticias/clube-naval-cascais-recebe-seabin-primeiro-dispositivo-limpeza-oceanica>.

## References

---

- [15] Câmara Municipal de Cascais, "Clube Naval de Cascais recebe "seabin", primeiro dispositivo para limpeza oceânica," 19 November 2018. [Online]. Available: <https://www.cascais.pt/noticia/clube-naval-de-cascais-recebe-seabin-primeiro-dispositivo-para-limpeza-oceanica>.
- [16] "Inovação Tecnológica," 28 October 2009. [Online]. Available: <https://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=robo-gari-recolhe-lixo-ajuda-pessoas&id=010180091028>.
- [17] ROBOTECH srl, "ROBOTECH srl," [Online]. Available: <https://www.robotechsrl.com/dustcart-en-urban-robot/>.
- [18] Corvid Cleaning, "Corvid Cleaning," [Online]. Available: <https://corvidcleaning.com/>.
- [19] T. Kitamura, *Fusion2URDF*, G. repository, Ed., GitHub, 2020.
- [20] P. Baheti, "Image Recognition: Definition, Algorithms & Uses," 05 October 2022. [Online]. Available: <https://www.v7labs.com/blog/image-recognition-guide>.
- [21] P. Proença and P. Simões, "TACO: Trash Annotations in Context for Litter Detection," 17 March 2020.
- [22] datagen, "MS COCO Dataset: Using it in Your Computer Vision Projects," [Online]. Available: <https://datagen.tech/guides/image-datasets/ms-coco-dataset-using-it-in-your-computer-vision-projects/>.
- [23] C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022.
- [24] D. Spodarets, "A Guide to the YOLO Family of Computer Vision Models," 12 February 2023. [Online]. Available: <https://dataphoenix.info/a-guide-to-the-yolo-family-of-computer-vision-models/>.
- [25] ROS.org. Open Robotics, "ROS/Introduction," [Online]. Available: <http://wiki.ros.org/ROS/Introduction>.
- [26] Open Source Robotics Foundation, 2020. [Online]. Available: <http://sdformat.org/>.
- [27] Litter, *LitterDetector Dataset*, Roboflow, Ed., Roboflow Universe, 2022.
- [28] Maestrovirtuale.com, "Maestrovirtuale.com," [Online]. Available: <https://maestrovirtuale.com/poluicao-do-lixo-causas-consequencias-e-exemplos/>.



# ANNEX A

List of ROS packages used:

Package	Description
arm_controll_pkg <sup>1</sup>	Higher-level controller linking with moveit_ros_move_group package.
asv_wave_sim	Gazebo plugin with wave development for visual realism. Here is also added the created beach world and object descriptors files for this dissertation.  <a href="#">GitHub - srmainwaring/asv_wave_sim at gazebo11</a>
boxbot_navigation <sup>1</sup>	Boxbot center of operation as it does movement planner, mode selector, arm commands.
controller_manager	Lower-level motor controller.
gazebo_ros	Engine for world simulation.
interactive_marker_twist_server	Serve interactive markers for control of a robot drive base.
litter_detection <sup>1</sup>	Package responsible for transform to a 3D pose, list, and queue the objects received via image recognition.
move_base	High-level controller for Boxbot movement and obstacle avoidance composed with multiple parameters such as the global planner, local planner, footprint, and many other.
moveit_ros_move_group	High-level controller for UR5 planning and inverse kinematics movement.
robot_localization	GPS integration is done within this package with the use of EKF localization method.
robot_state_publisher	robot_state_publisher uses the URDF specified by the parameter robot_description and the joint positions from the topic joint_states to calculate the forward kinematics of the robot and publish the results via tf.

---

<sup>1</sup> Packages created.

---



Package	Description
realsense	Package of the camera used for its descriptor. This package it is only fully functional with a real Intel Realsense camera.  <a href="#">GitHub - leggedrobotics/realsense-ros-rsl: Intel(R) RealSense(TM) ROS Wrapper for D400 series, SR300 Camera and T265 Tracking Module</a>
realsense_gazebo_plugin	Package needed to simulate the camera on Gazebo.  <a href="#">GitHub - pal-robotics/realsense_gazebo_plugin</a>
rviz	Visual debugging package.
twist_mux	Mux selector between multiple cmd_vel topic publishers according to a given priority.
universal_robot	Package need for its UR5 descriptor.  <a href="#">GitHub - ros-industrial/universal_robot: ROS-Industrial Universal Robots support (https://wiki.ros.org/universal_robot)</a>
yolov7_ros	Package responsible for image recognition and video inference given a weighted trained model.  <a href="#">GitHub - lukazso/yolov7-ros: ROS package for official YOLOv7</a>

Ros dependencies (other packages that the previous are dependent on):

- eigen-stl\_containers
- random\_numbers
- object\_recognition\_msgs
- octomap\_msgs
- ddynamic\_reconfigure
- graph\_msgs
- ruckig
- pybind11\_catkin
- warehouse\_ros
- eigenpy
- rosparam\_shortcuts
- move\_base\_msgs
- vision\_msgs
- robot\_localization
- interactive\_marker\_twist\_server
- twist\_mux
- move\_base
- trac-ik-kinematics-plugin
- dwa\_local\_planner
- effort-controllers
- JointTrajectoryController
- hector-gazebo-plugins

## ANNEX A

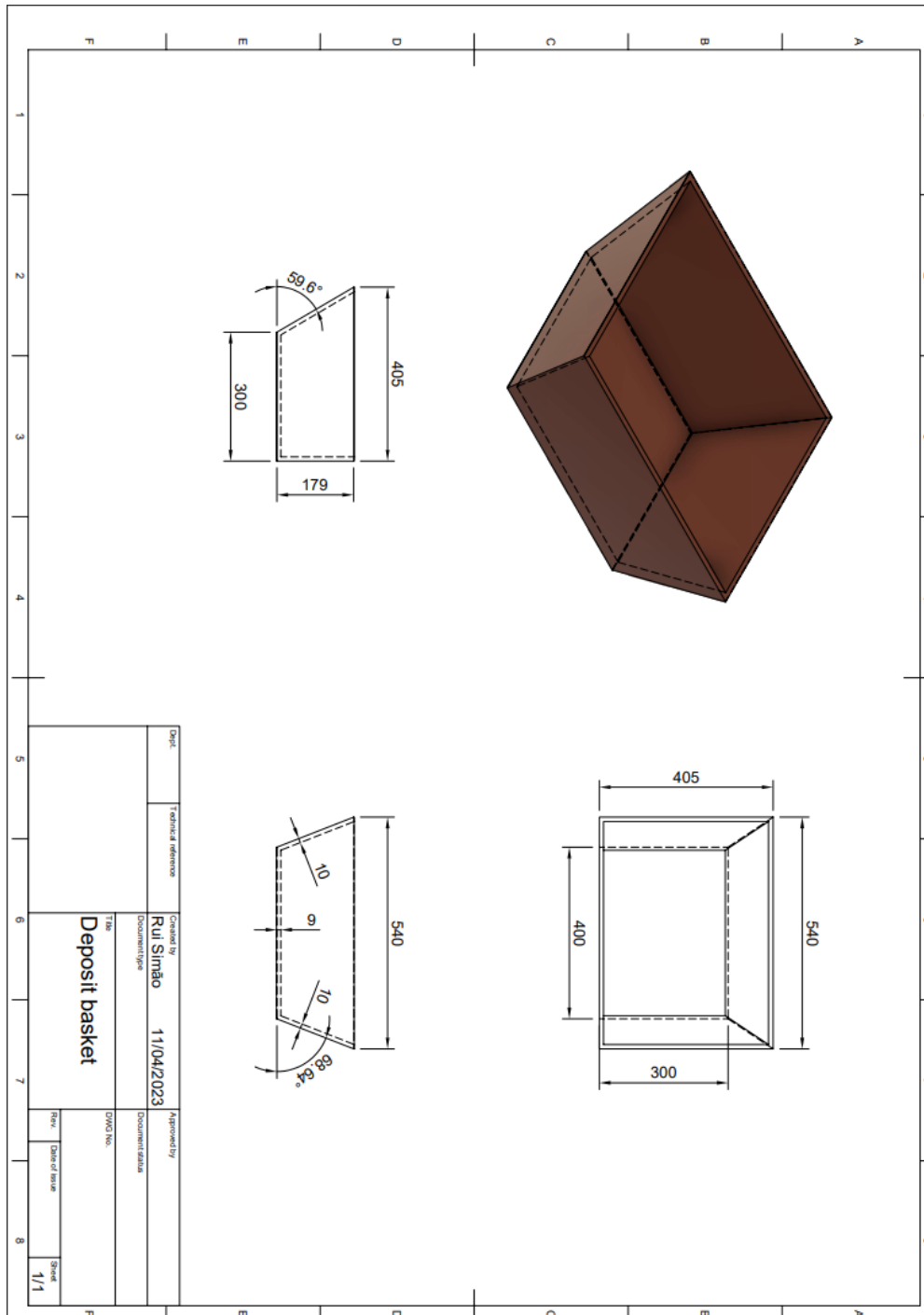
---

Other dependencies:

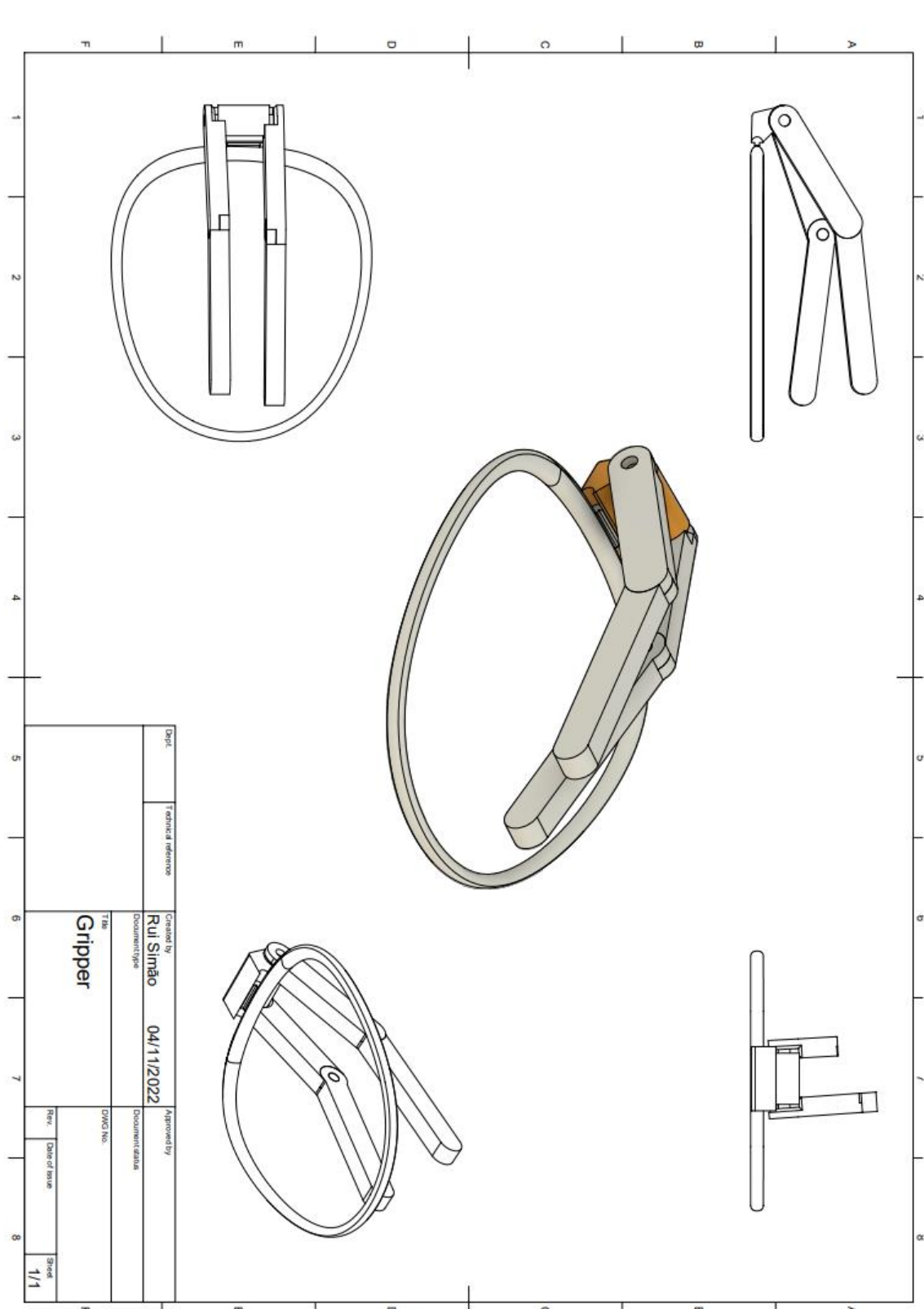
Libraries:	Python modules (pip install):
CGAL	Yolov7_ros requirements text list
Pyrealsense2	scipy
Intel RealSense SDK 2.0	

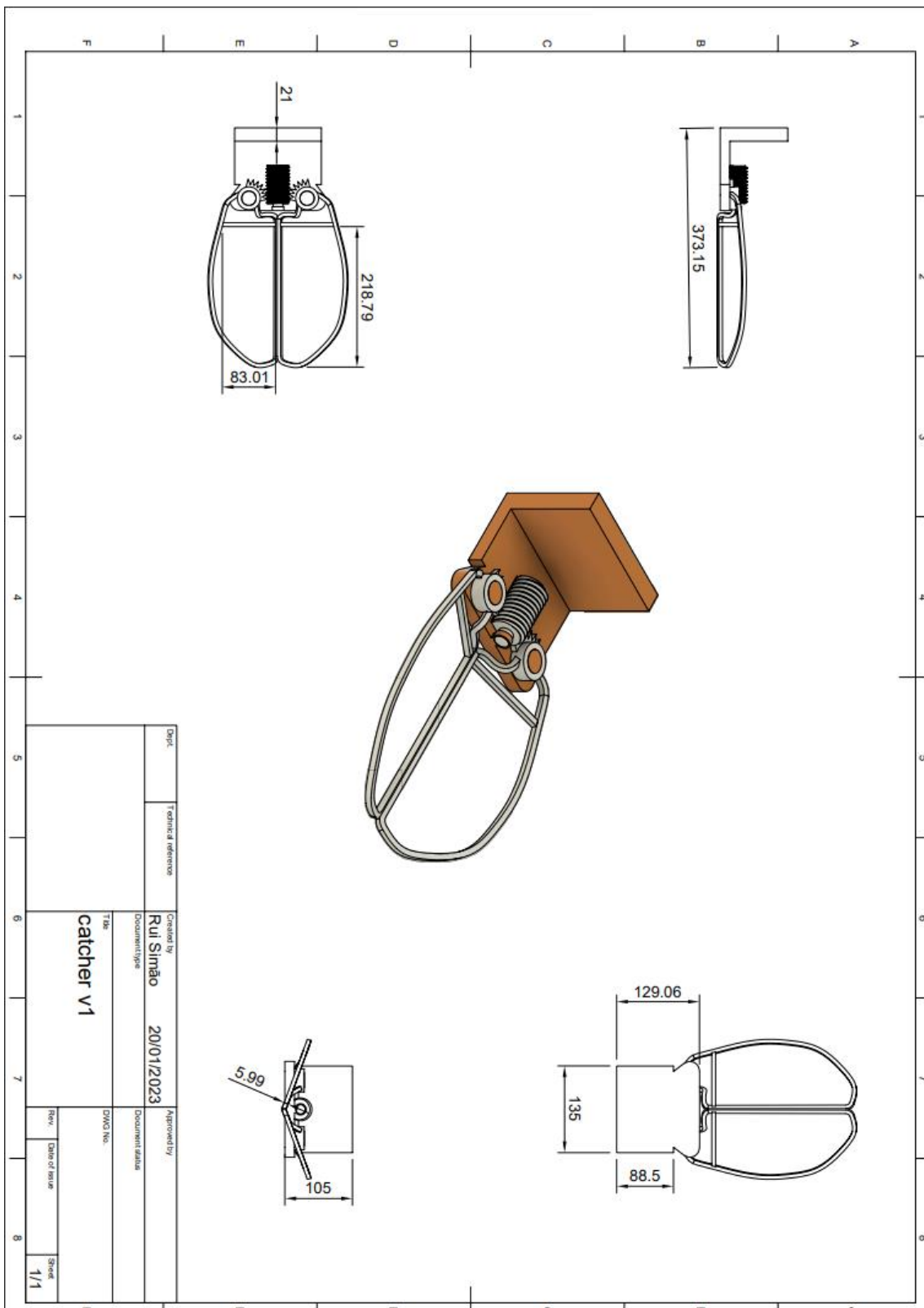
# ANNEX B

Drawings of the model designs for the deposit basket and gripper.

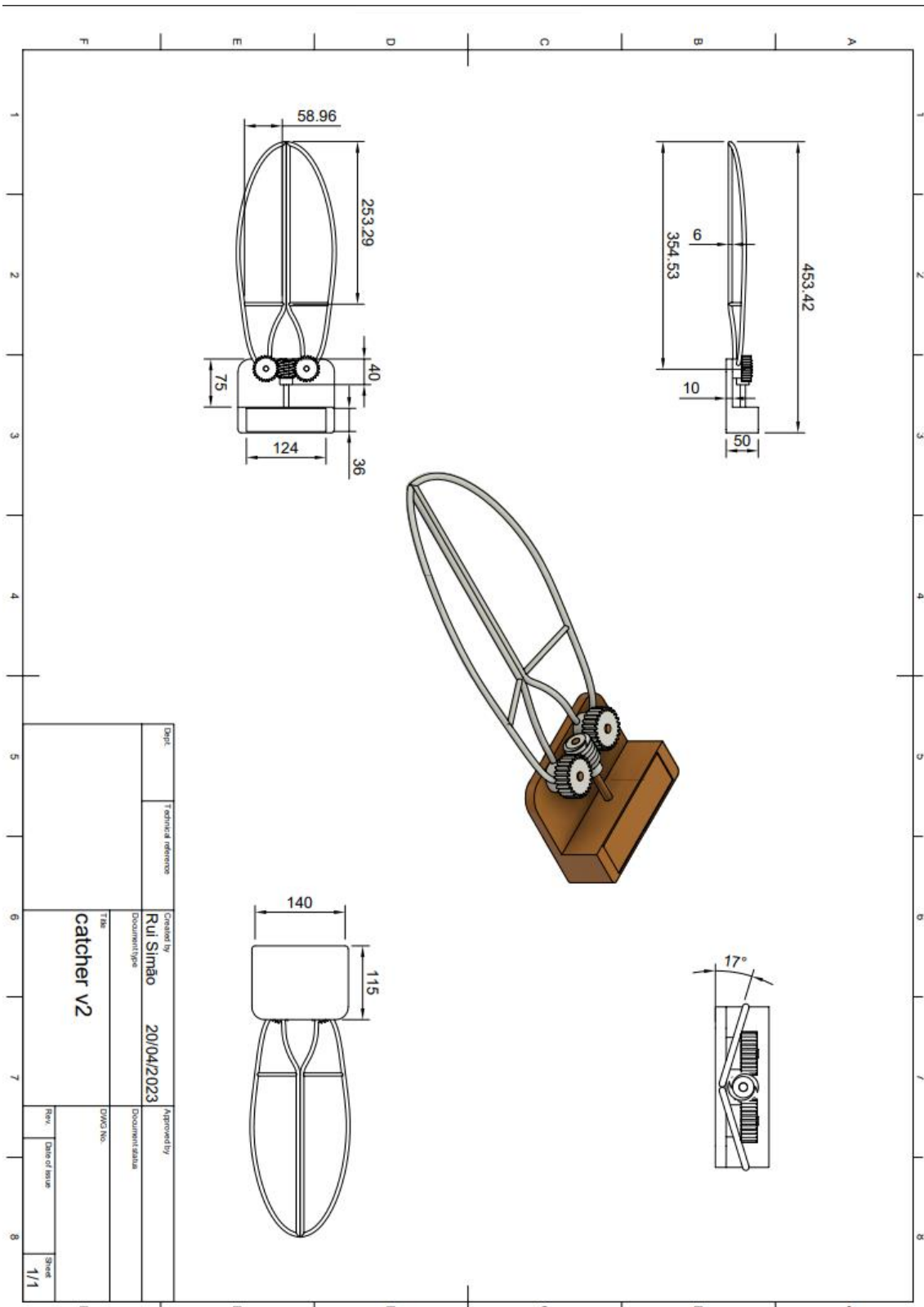


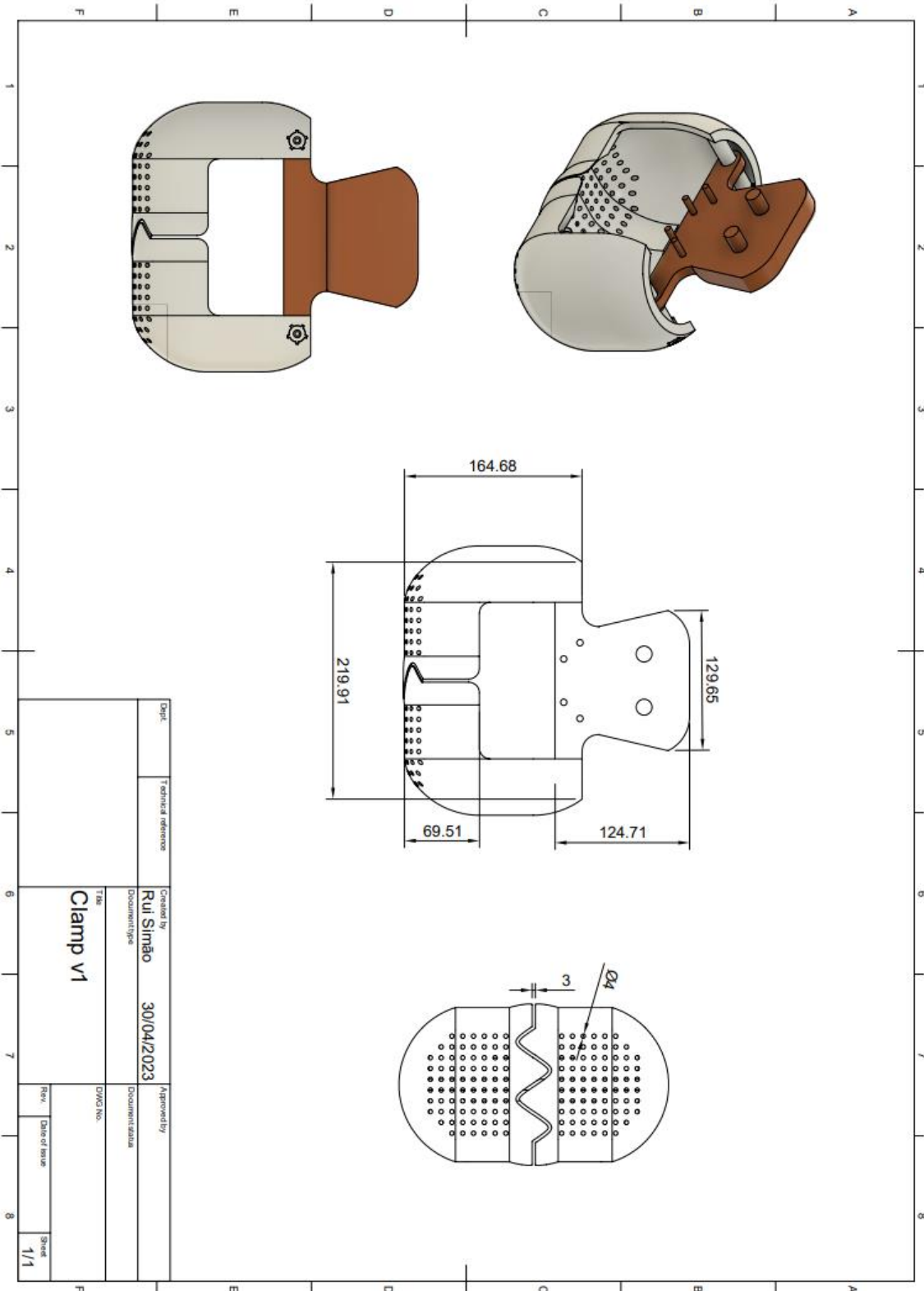
ANNEX B



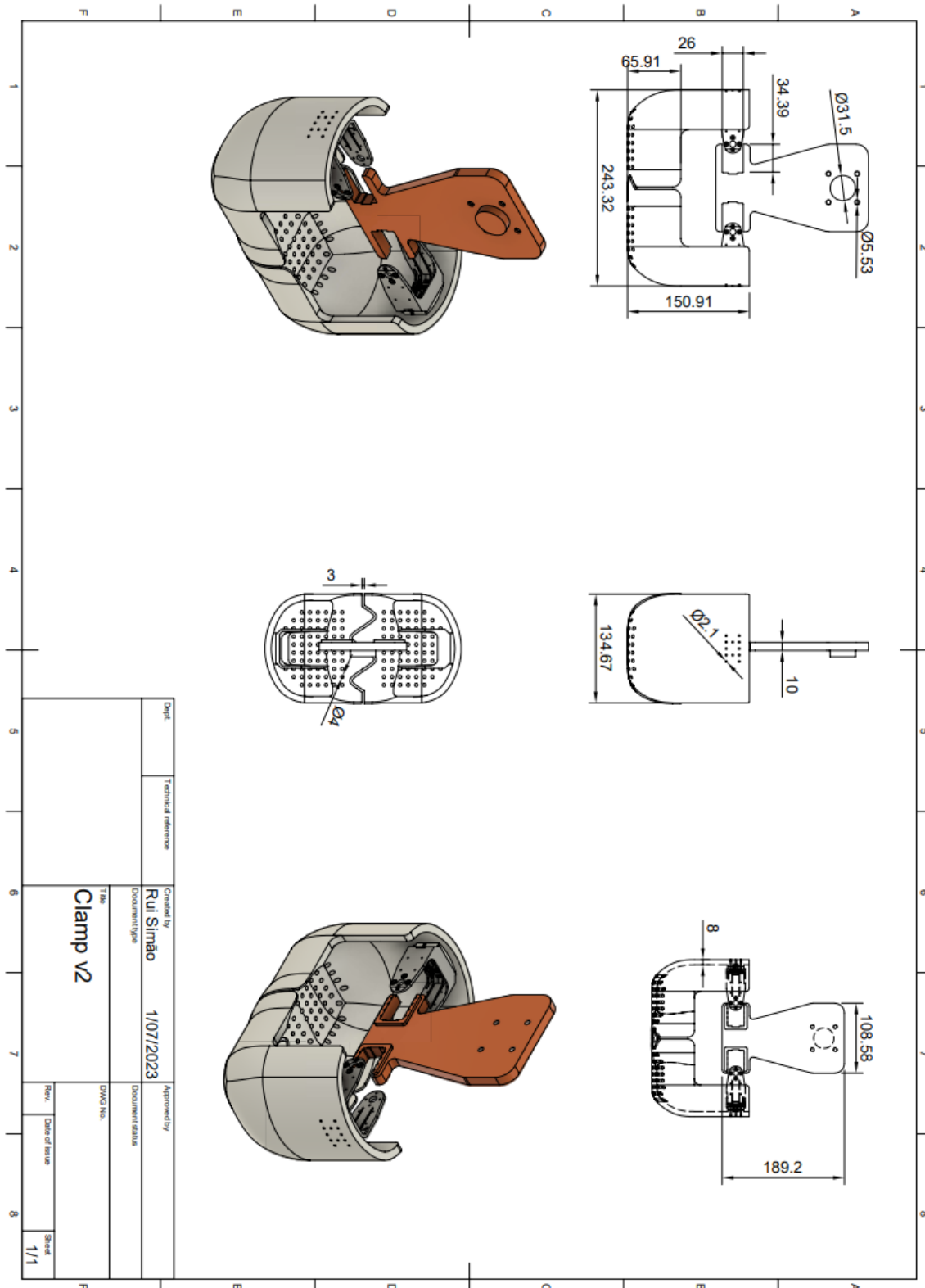


ANNEX B

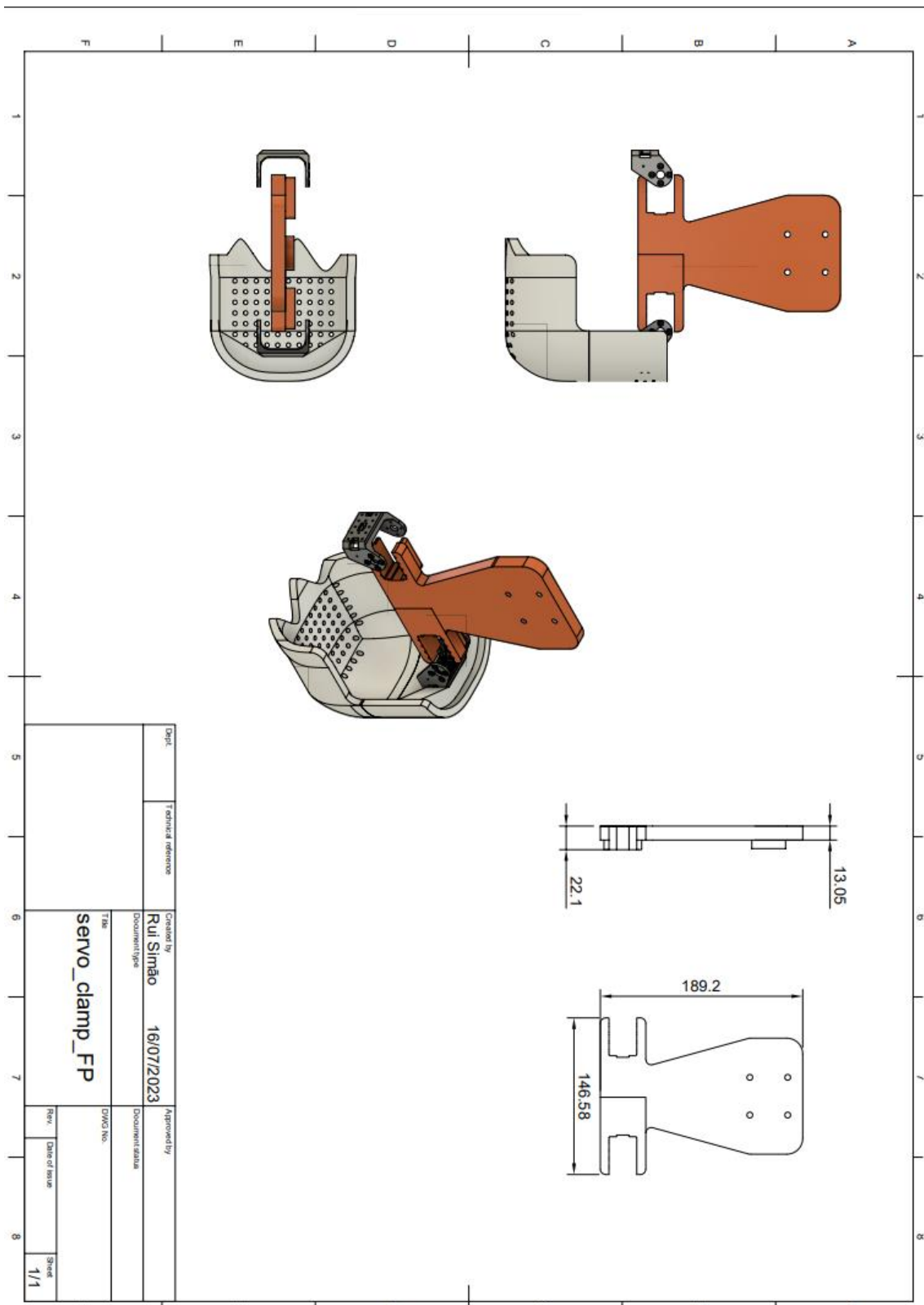




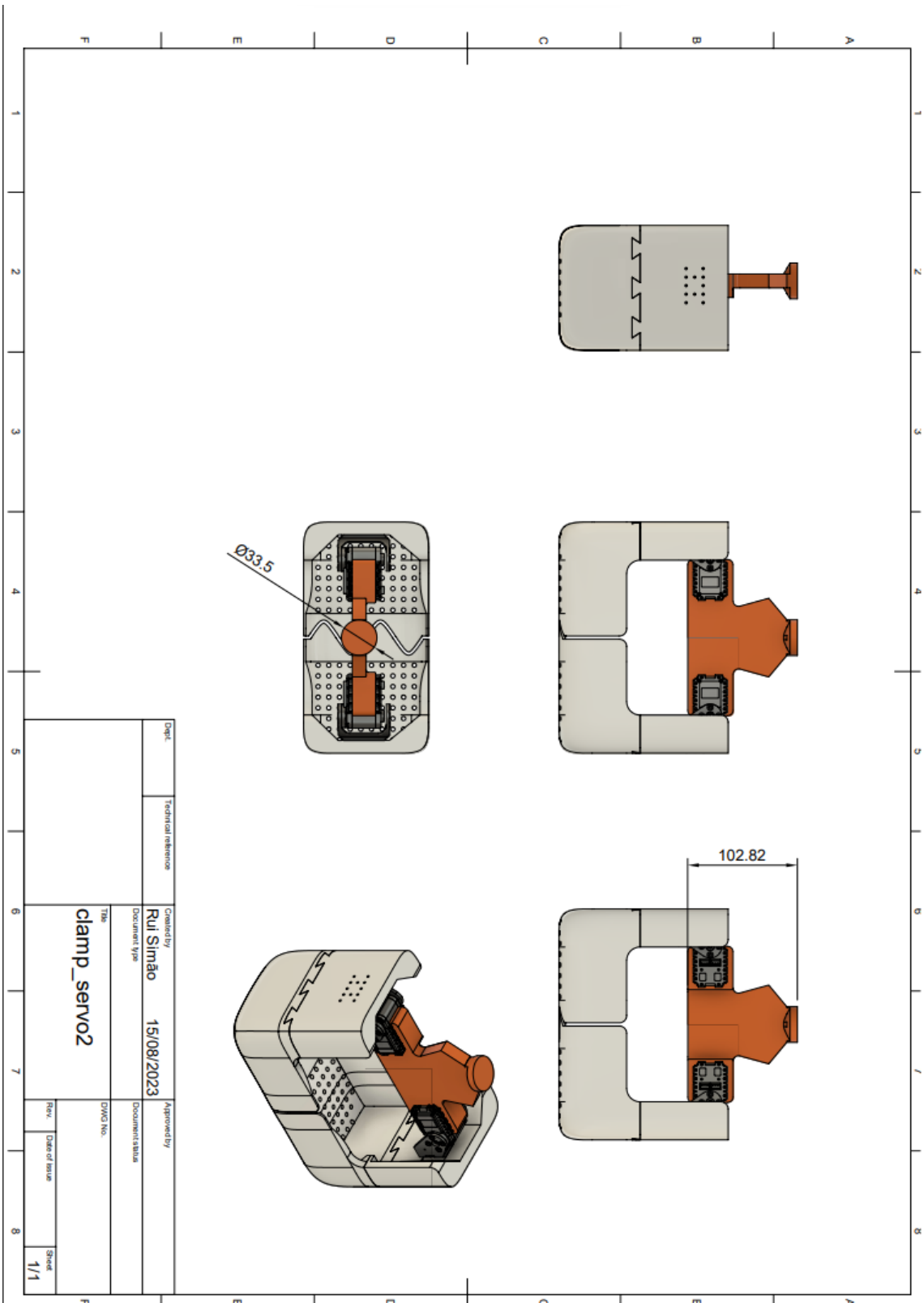
ANNEX B





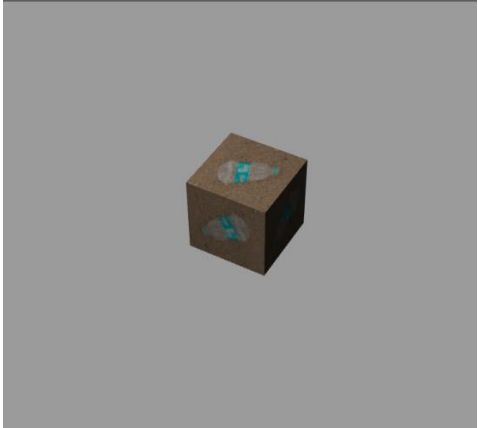

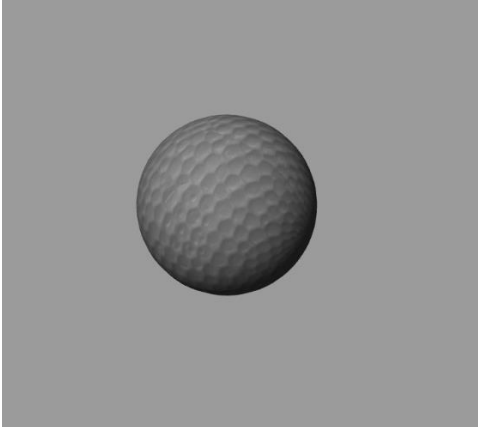



ANNEX B



## ANNEX C

All the objects created in Gazebo representing litter.

	
Object 1 – Cube textured with plastic bottle. Dimensions: 5 x 5 x 5 cm	Object 2 – Cigarette. Dimensions: 8.5 cm x 8.6 mm $\varnothing$
	
Object 3 – Sphere of Styrofoam. Dimensions: 10 cm $\varnothing$	Object 4 – Metal soda can. Dimensions: 12cm x 6.6 cm $\varnothing$

# ANNEX D

Gazebo world file descriptor created.

```
<sdf version='1.4'>
  <world name='default'>
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
      <pose>13.575 5.5375 1000.0 0 0 0</pose>
    </include>
    <!--<physics name="default_physics"
default="true" type="ode">
      <max_step_size>0.01</max_step_size>
      <real_time_factor>1</real_time_factor>
<real_time_update_rate>100</real_time_update_rate>
      <ode>
        <solver>
          <type>quick</type>
          <iters>150</iters>
          <sor>1.4</sor>
        </solver>
      </ode>
    </physics>-->
    <scene>
      <ambient>0.01 0.01 0.01 1.0</ambient>
      <sky>
        <clouds>
          <speed>12</speed>
        </clouds>
      </sky>
      <shadows>1</shadows>
    </scene>
    <spherical_coordinates>
      <latitude_deg>63.4520171</latitude_deg>
      <longitude_deg>10.3844778</longitude_deg>
    </spherical_coordinates>

    <include>
      <name>beach</name>
      <uri>model://sand</uri>
      <pose>0 0 0.1 0 0 0</pose>
    </include>
    <include>
      <name>fakeBeach</name>
      <uri>model://sand_no_c</uri>
      <pose>0 0 0.12 0 0 0</pose>
    </include>
```

World  
scenario and  
physics  
description

Beach files  
descriptor  
where last is a  
separate file  
without  
collision

```
<include>
  <name>pbottle</name>
  <uri>model://pbottle</uri>
  <pose>3.615 0.6 0.16 0 0 0</pose>
</include>
<include>
  <name>step_pbottle</name>
  <uri>model://step</uri>
  <pose>3.615 0.6 0.1 0 0 0</pose>
</include>

<include>
  <name>cigarrete</name>
  <uri>model://cigarrete</uri>
  <pose>9.27 1.05 0.16 1.5 0 0</pose>
</include>
<include>
  <name>step_cigarrete</name>
  <uri>model://step</uri>
  <pose>9.27 1.05 0.1 0 0 0</pose>
</include>
<include>
  <name>styrofoam</name>
  <uri>model://Styrofoam</uri>
  <pose>11.05 2.875 0.16 -1.2 -.5 0.35</pose>
</include>
<include>
  <name>step_sty</name>
  <uri>model://step</uri>
  <pose>11.05 2.875 0.1 0 0 0</pose>
</include>

<include>
  <name>cigarrete2</name>
  <uri>model://cigarrete</uri>
  <pose>1.755 4.6 0.16 1.5 0 0.6254</pose>
</include>
<include>
  <name>step_cigarrete2</name>
  <uri>model://step</uri>
  <pose>1.755 4.6 0.1 0 0 0.6254</pose>
</include>

<include>
  <name>SodaCan</name>
  <uri>model://RustySodaCan</uri>
  <pose>5 6.25 0.16 0 0 0</pose>
</include>
```

Descriptor of the  
object litter  
models with the  
elevated invisible  
step.

```
<include>
  <name>step_can</name>
  <uri>model://step</uri>
  <pose>5 6.25 0.1 0 0 0</pose>
</include>

<include>
  <name>pbottle2</name>
  <uri>model://pbottle</uri>
  <pose>8.615 8.8 0.16 0 0 0</pose>
</include>
<include>
  <name>step_pbottle2</name>
  <uri>model://step</uri>
  <pose>8.615 8.8 0.1 0 0 0</pose>
</include>

<include>
  <uri>model://ocean_waves</uri>
</include>
<gui fullscreen='0'>
  <camera name='user_camera'>
    <pose frame=''>9.0 -12.0 6.0 0.0 0.3
2.2</pose>
    <view_controller>orbit</view_controller>

<projection_type>perspective</projection_type>
  </camera>
</gui>
</world>
</sdf>
```

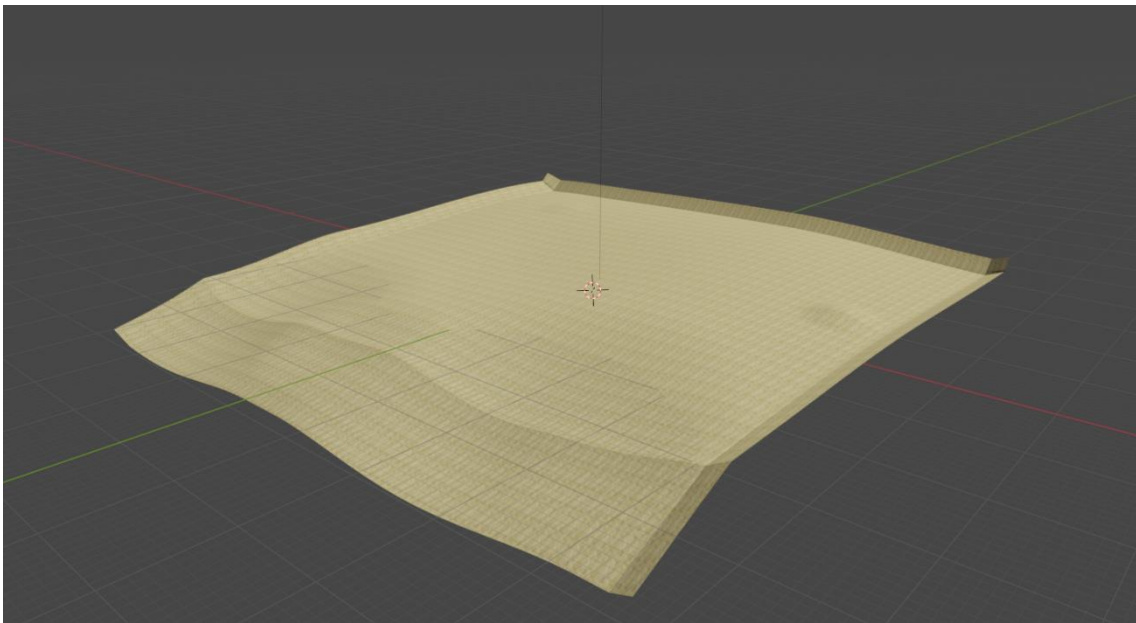
Descriptor of the object litter models with the elevated invisible step.

Including wave plugin

## ANNEX E

First version of sand floor, developed in Blender, before being discarded due to incompatibilities with Gazebo.

This would have intersected with the waves plugin and have slip physics.



# ANNEX F

Universal Robotics UR5 model specifications:

## UR5 Technical specifications

Item no. 110105

### 6-axis robot arm with a working radius of 850 mm / 33.5 in

<b>Weight:</b>	18.4 kg / 40.6 lbs		
<b>Payload:</b>	5 kg / 11 lbs		
<b>Reach:</b>	850 mm / 33.5 in		
<b>Joint ranges:</b>	+/- 360°		
<b>Speed:</b>	All joints: 180°/s. Tool: Typical 1 m/s. / 39.4 in/s.		
<b>Repeatability:</b>	+/- 0.1 mm / +/- 0.0039 in (4 mils)		
<b>Footprint:</b>	Ø149 mm / 5.9 in		
<b>Degrees of freedom:</b>	6 rotating joints		
<b>Control box size (WxHxD):</b>	475 mm x 423 mm x 268 mm / 18.7 x 16.7 x 10.6 in		
<b>I/O ports:</b>		<b>Controlbox</b>	<b>Tool conn.</b>
	Digital in	16	2
	Digital out	16	2
	Analog in	2	2
	Analog out	2	-
<b>I/O power supply:</b>	24 V 2A in control box and 12 V/24 V 600 mA in tool		
<b>Communication:</b>	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP		
<b>Programming:</b>	Polyscope graphical user interface on 12 inch touchscreen with mounting		
<b>Noise:</b>	Comparatively noiseless		
<b>IP classification:</b>	IP54		
<b>ISO Class Cleanroom robot arm:</b>	5		
<b>ISO Class Cleanroom control box:</b>	6		
<b>Power consumption:</b>	Approx. 200 watts using a typical program		
<b>Collaboration operation:</b>	15 Advanced Safety Functions Tested in accordance with: EN ISO 13849:2008 PL d EN ISO 10218-1:2011, Clause 5.4.3		
<b>Materials:</b>	Aluminum, PP plastic		
<b>Temperature:</b>	The robot can work in a temperature range of 0-50°C		
<b>Power supply:</b>	100-240 VAC, 50-60 Hz		
<b>Cabling:</b>	Cable between robot and control box (6 m / 236 in) Cable between touchscreen and control box (4.5 m / 177 in)		



Universal Robots A/S  
Energivej 25  
DK-5260 Odense S  
Denmark  
+45 89 93 89 89

www.universal-robots.com  
sales@universal-robots.com



Intel® RealSense™ D435 RGB-D specifications:

Essentials

Product Collection	<a href="#">Intel® RealSense™ Cameras</a>
Code Name	<a href="#">Products formerly Double Springs</a>
Marketing Status	Launched
Launch Date <a href="#">?</a>	Q1'18
Depth Technology	Active Stereoscopic

Operational Specifications

Operating Range (Min-Max)	~.3m - 3m
Depth Resolution and FPS	1280 X 720
Depth Field of View	85.2 x 58

Supplemental Information

Datasheet	<a href="#">View now</a>
-----------	--------------------------

Components

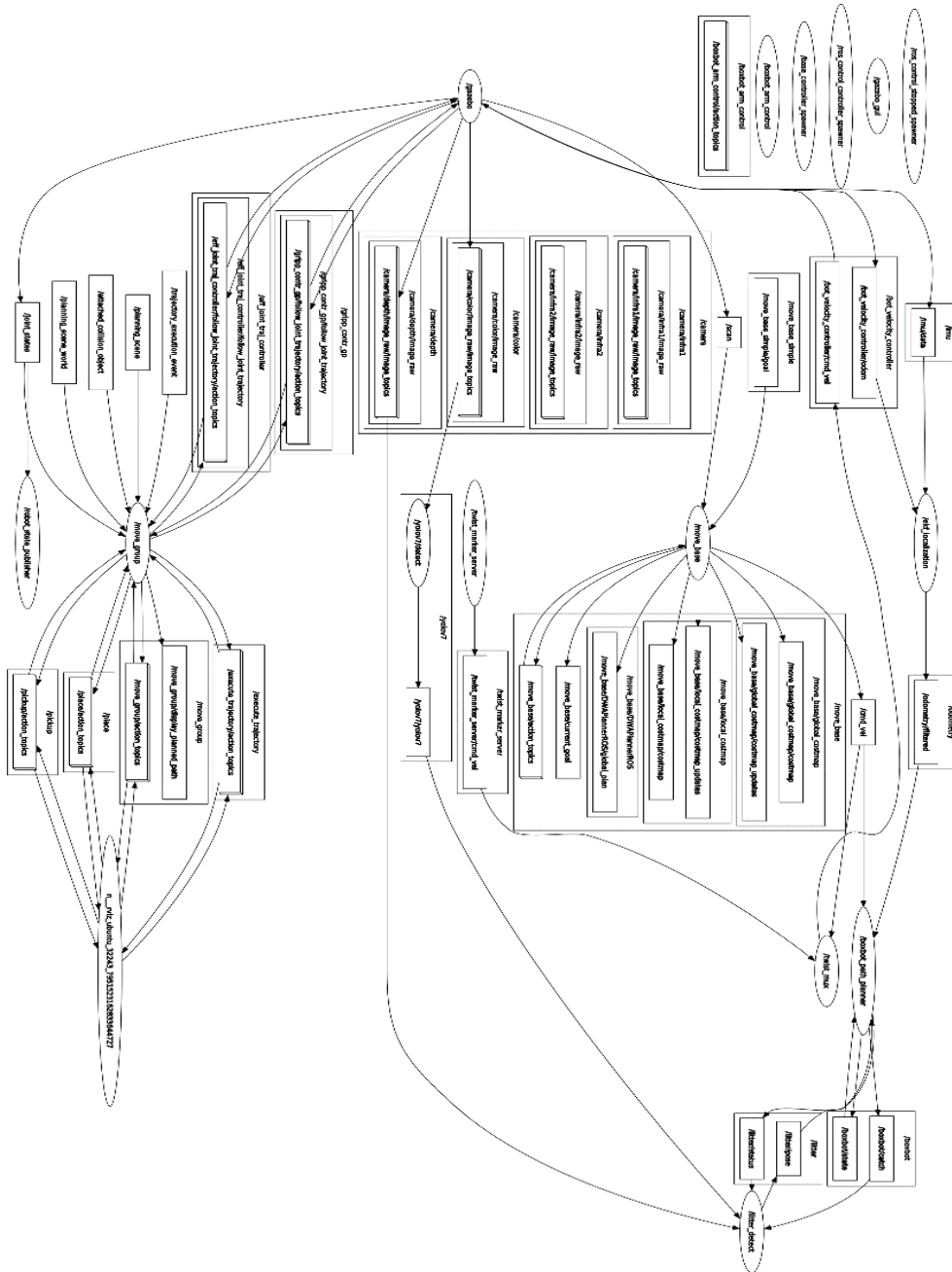
RGB Sensor	Yes
Tracking Module	No

Module Specifications

Dimensions	90 mm x 25 mm x 25 mm
System Interface Type	USB 3.0 Type C

# ANNEX G

rqt\_graph<sup>2</sup> showing all active topics and nodes and how they interconnect.



<sup>2</sup> ROS debugging tool.

## ANNEX H

Config file with the robotic arm description. Within are the named operation “*home*”, “*collected*”, “*up*”, “*deposit*”, “*closed*”, “*open*” and their joints preset positions.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This does not replace URDF, and is not an extension of URDF.
  This is a format for representing semantic information about the
  robot structure.
  A URDF file must exist for this robot as well, where the joints and
  the links that are referenced are defined
-->
<robot name="Boxbot">
  <xacro:arg name="model" default=""/>
  <!--GROUPS: Representation of a set of joints and links. This can be
  useful for specifying DOF to plan for, defining arms, end effectors, etc-
  ->
  <!--LINKS: When a link is specified, the parent joint of that link
  (if it exists) is automatically included-->
  <!--JOINTS: When a joint is specified, the child link of that joint
  (which will always exist) is automatically included-->
  <!--CHAINS: When a chain is specified, all the links along the chain
  (including endpoints) are included in the group. Additionally, all the
  joints that are parents to included links are also included. This means
  that joints along the chain and the parent joint of the base link are
  included in the group-->
  <!--SUBGROUPS: Groups can also be formed by referencing to already
  defined group names-->
  <group name="manipulator">
    <chain base_link="ur5_base_link" tip_link="center_tool"/>
  </group>
  <group name="gripper">
    <joint name="interface_joint"/>
    <joint name="R_joint"/>
    <joint name="L_joint"/>
  </group>
  <!--GROUP STATES: Purpose: Define a named state for a particular
  group, in terms of joint values. This is useful to define states like
  'folded arms'-->
  <group_state name="home" group="manipulator">
```

```
<joint name="ur5_elbow_joint" value="0"/>
<joint name="ur5_shoulder_lift_joint" value="0"/>
<joint name="ur5_shoulder_pan_joint" value="0"/>
<joint name="ur5_wrist_1_joint" value="0"/>
<joint name="ur5_wrist_2_joint" value="1.5708"/>
<joint name="ur5_wrist_3_joint" value="0"/>
</group_state>
<group_state name="collected" group="manipulator">
  <joint name="ur5_elbow_joint" value="2.07694"/>
  <joint name="ur5_shoulder_lift_joint" value="-2.12930"/>
  <joint name="ur5_shoulder_pan_joint" value="0"/>
  <joint name="ur5_wrist_1_joint" value="0.0372"/>
  <joint name="ur5_wrist_2_joint" value="1.5708"/>
  <joint name="ur5_wrist_3_joint" value="0"/>
</group_state>
<group_state name="up" group="manipulator">
  <joint name="ur5_elbow_joint" value="0.7621"/>
  <joint name="ur5_shoulder_lift_joint" value="-1.9704"/>
  <joint name="ur5_shoulder_pan_joint" value="0"/>
  <joint name="ur5_wrist_1_joint" value="1.2269"/>
  <joint name="ur5_wrist_2_joint" value="1.5708"/>
  <joint name="ur5_wrist_3_joint" value="0"/>
</group_state>
<group_state name="deposit" group="manipulator">
  <joint name="ur5_elbow_joint" value="-0.8365"/>
  <joint name="ur5_shoulder_lift_joint" value="-1.5987"/>
  <joint name="ur5_shoulder_pan_joint" value="0"/>
  <joint name="ur5_wrist_1_joint" value="-3.6063"/>
  <joint name="ur5_wrist_2_joint" value="-1.7474"/>
  <joint name="ur5_wrist_3_joint" value="0"/>
</group_state>
<group_state name="closed" group="gripper">
  <joint name="R_joint" value="0"/>
  <joint name="L_joint" value="0"/>
</group_state>
<group_state name="open" group="gripper">
  <joint name="R_joint" value="-1.2217"/>
  <joint name="L_joint" value="1.2217"/>
</group_state>
<!--END EFFECTOR: Purpose: Represent information about an end
effector.-->
<end_effector name="geef" parent_link="center_tool" group="gripper"
parent_group="manipulator"/>
<!--PASSIVE JOINT: Purpose: this element is used to mark joints that
are not actuated-->
```

```
<passive_joint name="wheel_Fl_joint"/>
<passive_joint name="wheel_Fr_joint"/>
<passive_joint name="wheel_Rl_joint"/>
<passive_joint name="wheel_Rr_joint"/>
  <!--VIRTUAL JOINT: Purpose: this element defines a virtual joint
between a robot link and an external frame of reference (considered fixed
with respect to the robot)-->
  <!--<virtual_joint name="base_fixed" type="fixed"
parent_frame="base_body" child_link="body"/>-->
  <!--DISABLE COLLISIONS: By default it is assumed that any link of the
robot could potentially come into collision with any other link in the
robot. This tag disables collision checking between a specified pair of
links. -->
    <disable_collisions link1="ur5_base_link_inertia" link2="body"
reason="Adjacent"/>
    <disable_collisions link1="ur5_base_link_inertia" link2="buck_link"
reason="Never"/>
    <disable_collisions link1="ur5_base_link_inertia" link2="camera_link"
reason="Never"/>
    <disable_collisions link1="ur5_base_link_inertia"
link2="ur5_shoulder_link" reason="Adjacent"/>
    <disable_collisions link1="ur5_base_link_inertia" link2="wheel_Fl"
reason="Never"/>
    <disable_collisions link1="ur5_base_link_inertia" link2="wheel_Fr"
reason="Never"/>
    <disable_collisions link1="ur5_base_link_inertia" link2="wheel_Rl"
reason="Never"/>
    <disable_collisions link1="ur5_base_link_inertia" link2="wheel_Rr"
reason="Never"/>
    <disable_collisions link1="body" link2="buck_link"
reason="Adjacent"/>
    <disable_collisions link1="body" link2="camera_link"
reason="Adjacent"/>
    <disable_collisions link1="body" link2="ur5_shoulder_link"
reason="Never"/>
    <disable_collisions link1="body" link2="wheel_Fl" reason="Adjacent"/>
    <disable_collisions link1="body" link2="wheel_Fr" reason="Adjacent"/>
    <disable_collisions link1="body" link2="wheel_Rl" reason="Adjacent"/>
    <disable_collisions link1="body" link2="wheel_Rr" reason="Adjacent"/>
    <disable_collisions link1="buck_link" link2="camera_link"
reason="Never"/>
    <disable_collisions link1="buck_link" link2="ur5_shoulder_link"
reason="Never"/>
    <disable_collisions link1="buck_link" link2="wheel_Fl"
reason="Never"/>
```

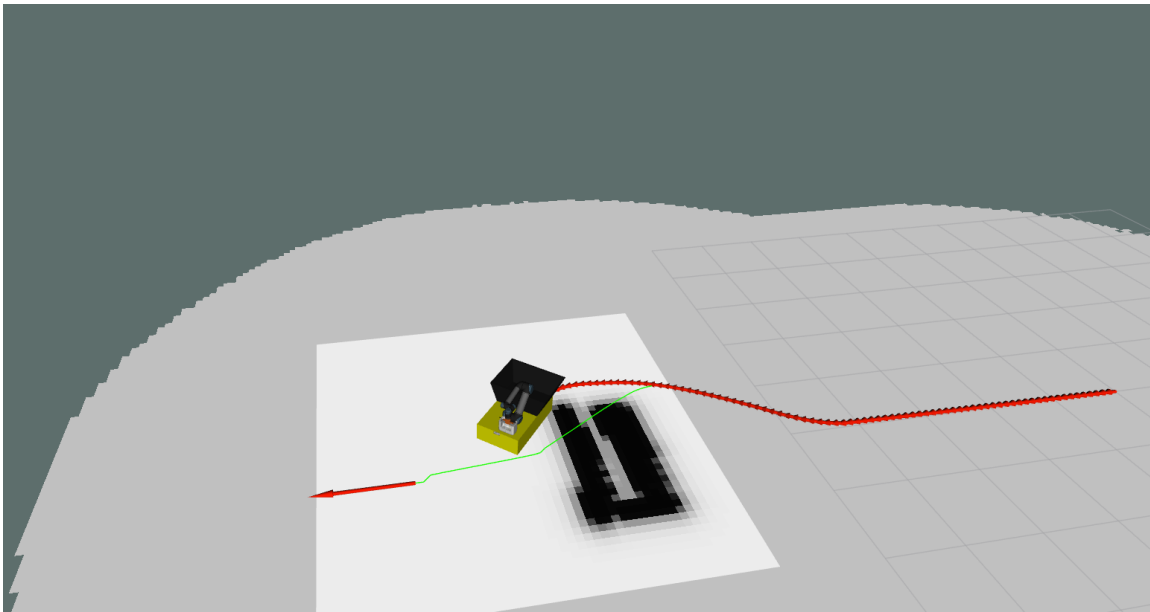
```
<disable_collisions link1="buck_link" link2="wheel_Fr"
reason="Never"/>
<disable_collisions link1="buck_link" link2="wheel_R1"
reason="Never"/>
<disable_collisions link1="buck_link" link2="wheel_Rr"
reason="Never"/>
<disable_collisions link1="camera_link" link2="ur5_shoulder_link"
reason="Never"/>
<disable_collisions link1="camera_link" link2="wheel_F1"
reason="Never"/>
<disable_collisions link1="camera_link" link2="wheel_Fr"
reason="Never"/>
<disable_collisions link1="camera_link" link2="wheel_R1"
reason="Never"/>
<disable_collisions link1="camera_link" link2="wheel_Rr"
reason="Never"/>
<disable_collisions link1="clamp_base_link" link2="clamp_l"
reason="Adjacent"/>
<disable_collisions link1="clamp_base_link" link2="clamp_r"
reason="Adjacent"/>
<disable_collisions link1="clamp_base_link" link2="ur5_wrist_1_link"
reason="Never"/>
<disable_collisions link1="clamp_base_link" link2="ur5_wrist_2_link"
reason="Never"/>
<disable_collisions link1="clamp_base_link" link2="ur5_wrist_3_link"
reason="Adjacent"/>
<disable_collisions link1="clamp_l" link2="clamp_r" reason="Never"/>
<disable_collisions link1="clamp_l" link2="ur5_wrist_2_link"
reason="Never"/>
<disable_collisions link1="clamp_l" link2="ur5_wrist_3_link"
reason="Never"/>
<disable_collisions link1="clamp_r" link2="ur5_wrist_2_link"
reason="Never"/>
<disable_collisions link1="clamp_r" link2="ur5_wrist_3_link"
reason="Never"/>
<disable_collisions link1="ur5_forearm_link"
link2="ur5_upper_arm_link" reason="Adjacent"/>
<disable_collisions link1="ur5_forearm_link" link2="ur5_wrist_1_link"
reason="Adjacent"/>
<disable_collisions link1="ur5_shoulder_link"
link2="ur5_upper_arm_link" reason="Adjacent"/>
<disable_collisions link1="ur5_shoulder_link" link2="wheel_F1"
reason="Never"/>
<disable_collisions link1="ur5_shoulder_link" link2="wheel_Fr"
reason="Never"/>
```

```
<disable_collisions link1="ur5_shoulder_link" link2="wheel_R1"
reason="Never"/>
<disable_collisions link1="ur5_shoulder_link" link2="wheel_Rr"
reason="Never"/>
<disable_collisions link1="ur5_upper_arm_link" link2="wheel_R1"
reason="Never"/>
<disable_collisions link1="ur5_upper_arm_link" link2="wheel_Rr"
reason="Never"/>
<disable_collisions link1="wheel_F1" link2="wheel_Fr"
reason="Never"/>
<disable_collisions link1="wheel_F1" link2="wheel_R1"
reason="Never"/>
<disable_collisions link1="wheel_F1" link2="wheel_Rr"
reason="Never"/>
<disable_collisions link1="wheel_Fr" link2="wheel_R1"
reason="Never"/>
<disable_collisions link1="wheel_Fr" link2="wheel_Rr"
reason="Never"/>
<disable_collisions link1="wheel_R1" link2="wheel_Rr"
reason="Never"/>
<disable_collisions link1="ur5_wrist_1_link" link2="ur5_wrist_2_link"
reason="Adjacent"/>
<disable_collisions link1="ur5_wrist_1_link" link2="ur5_wrist_3_link"
reason="Never"/>
<disable_collisions link1="ur5_wrist_2_link" link2="ur5_wrist_3_link"
reason="Adjacent"/>
</robot>
```

# ANNEX I

In this annex are shown images, regarding the execution of two simulations.

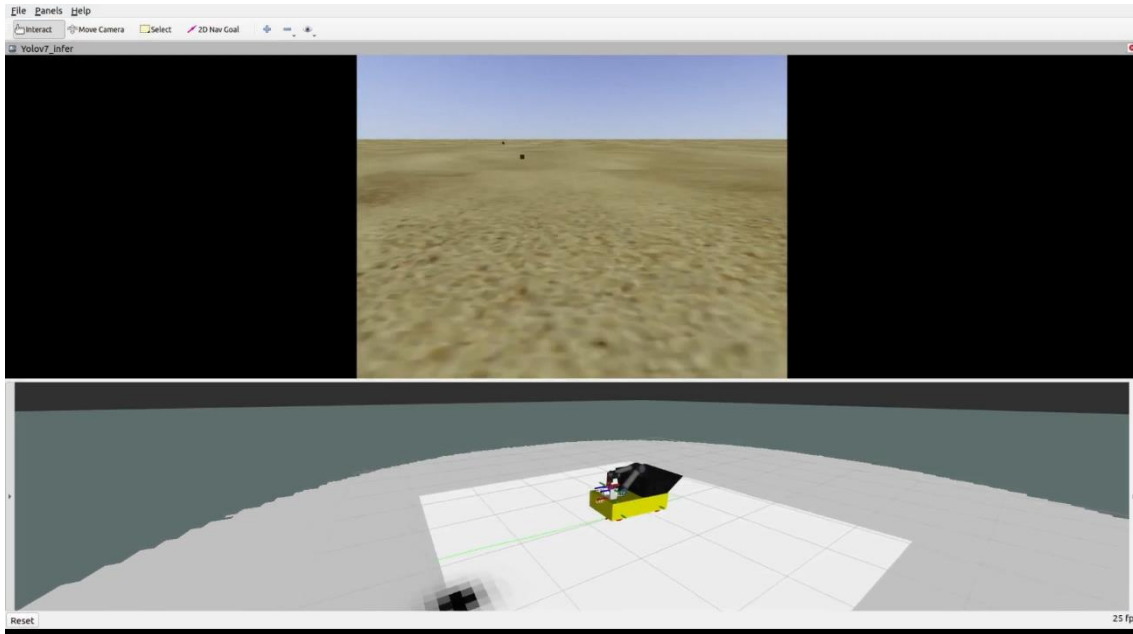
Simulation of person avoidance:



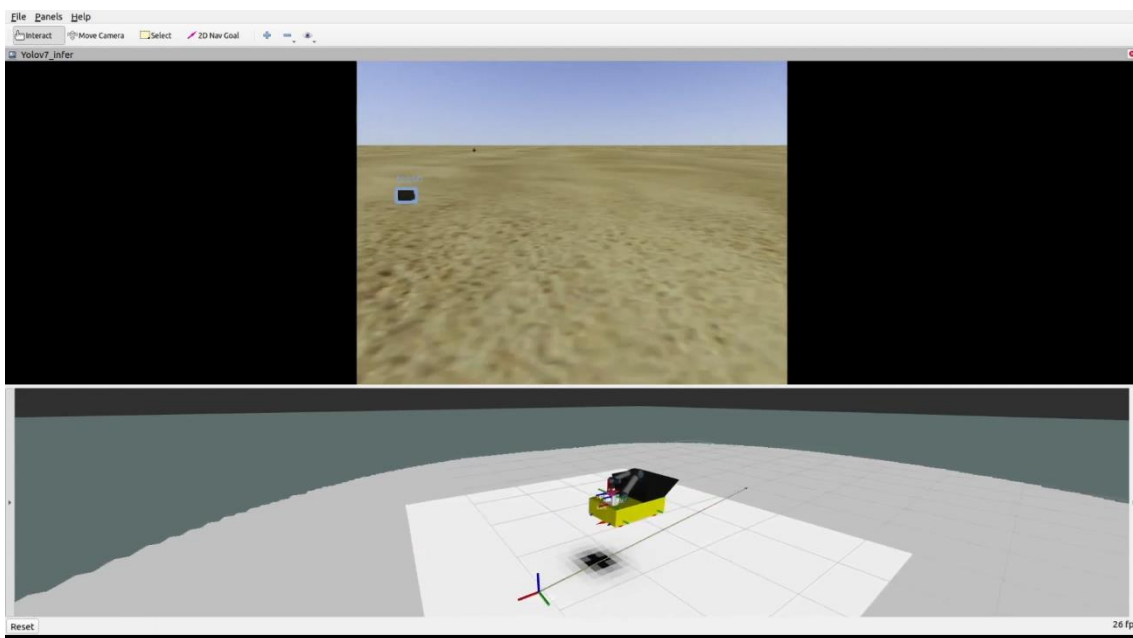
Rviz image showing Boxbot replanning its path to avoid a laying down person. To impersonate a person, it was used a rectangular box with dimensions of 1.80 x 0.5 x 0.3 meters, which is mapped on the costmap in black. In red, the path done by Boxbot, and in green, the planned path after realization that a straight trajectory was not doable.



Simulation of litter detection and catching using SWEEP mode:



Boxbot is in SWEEP mode and beginning the operation. At the distance it can be seen a litter object that has not been detected yet.

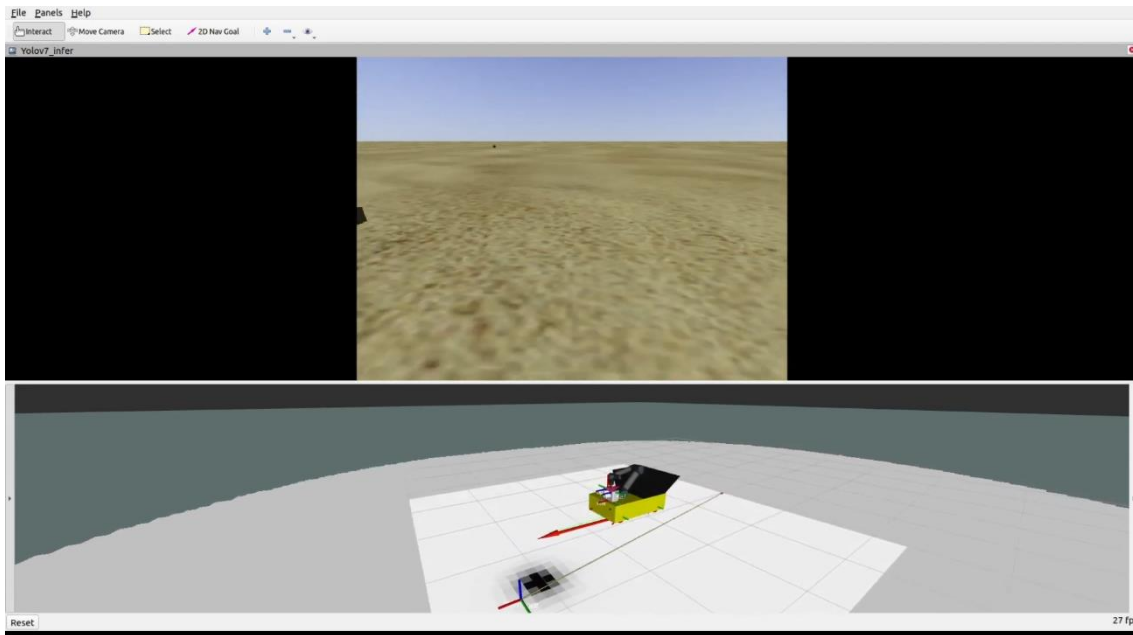


First detection of litter, in this case, a cube textured with plastic bottle, which led Boxbot to interrupt its trajectory. Here it is seen the litter referential indicating its pose relatively to the simulation world.

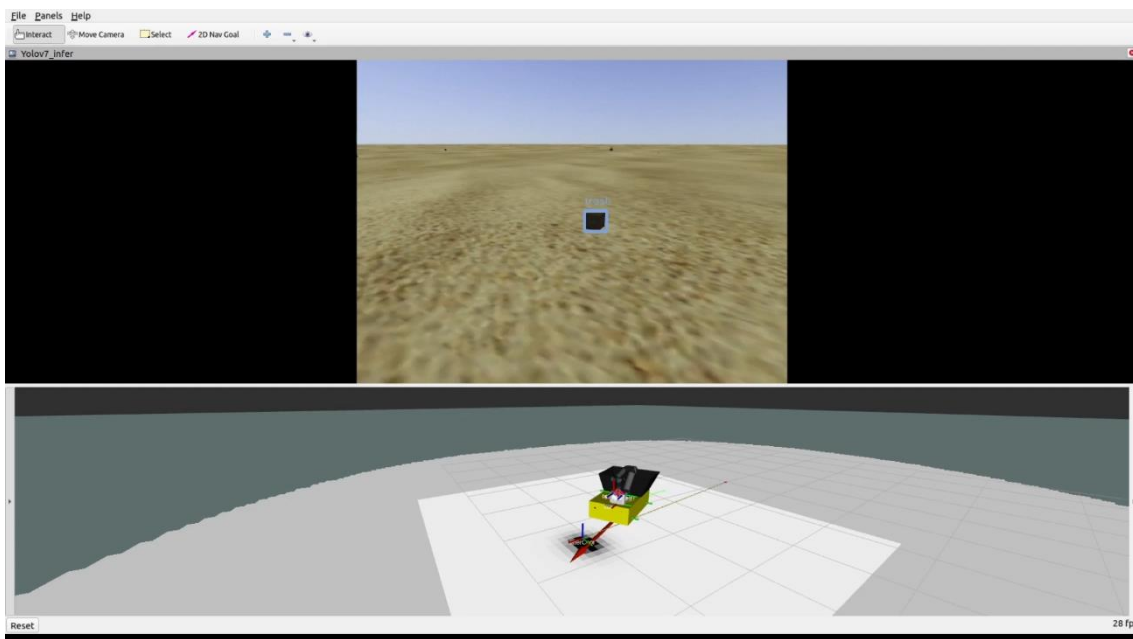
---

## ANNEX I

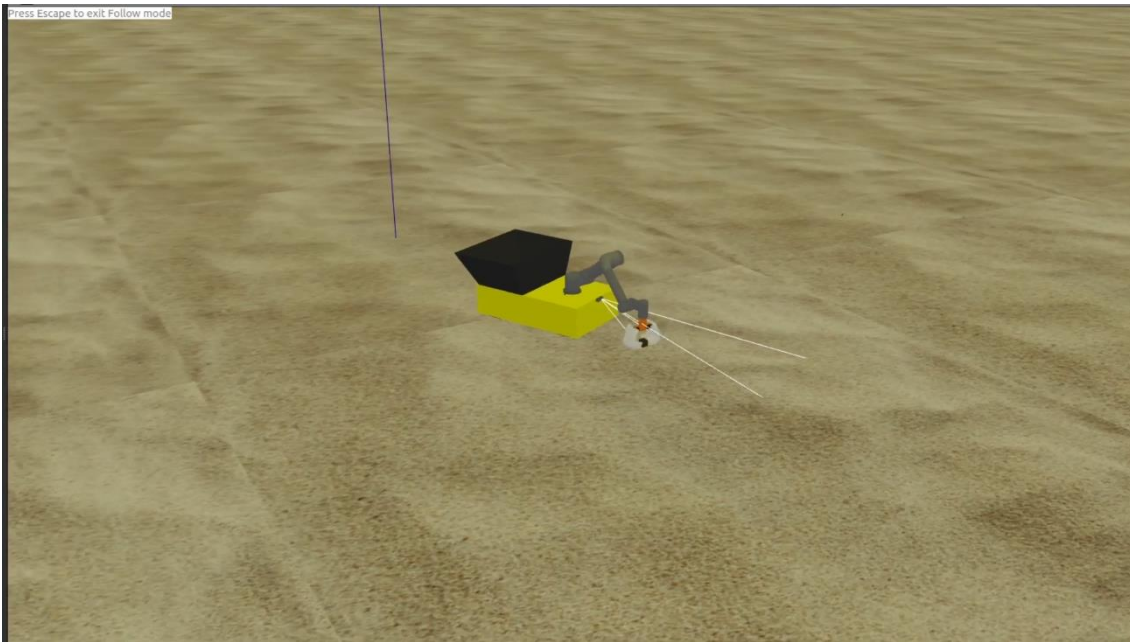
---



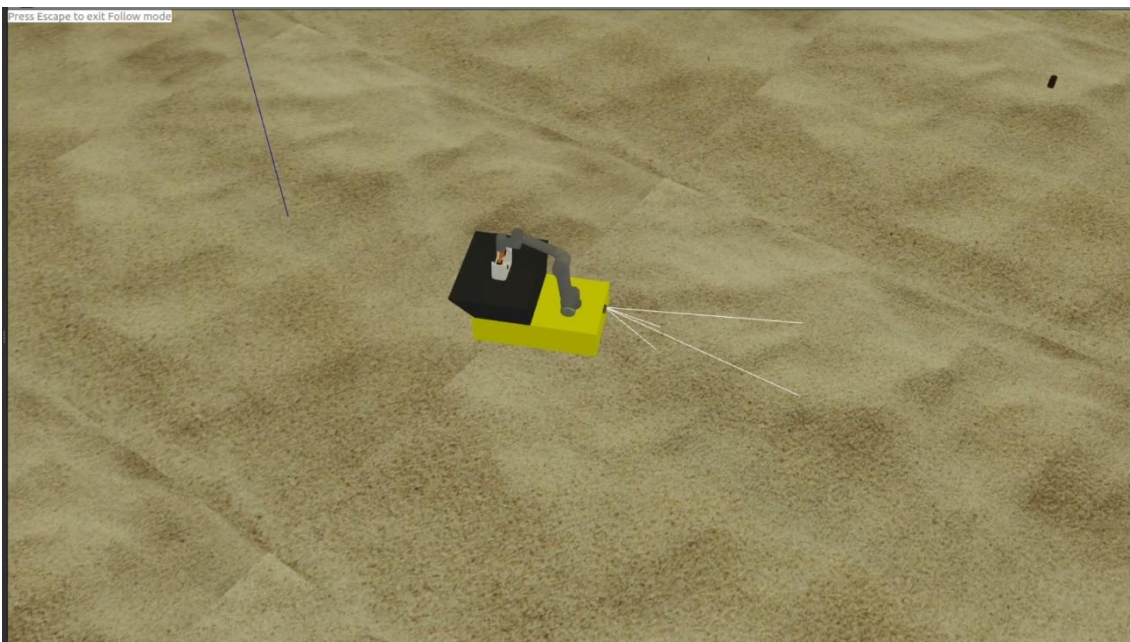
The figure above depicts the necessary Boxbot reversing, due to the misperception of litter, caused by the data delay offset of the yolo\_v7 detection and the depth data. Represented by the frame of reference is the litter's position. The red vector is the new pose waypoint (desired position and orientation of the robot) for the robot to reverse.



After reversing was completed, the litter has re-entered the camera's FoV. Boxbot has re-perceived the litter and corrected its position. A new waypoint, marked by the vector, has been set and Boxbot is moving towards it.

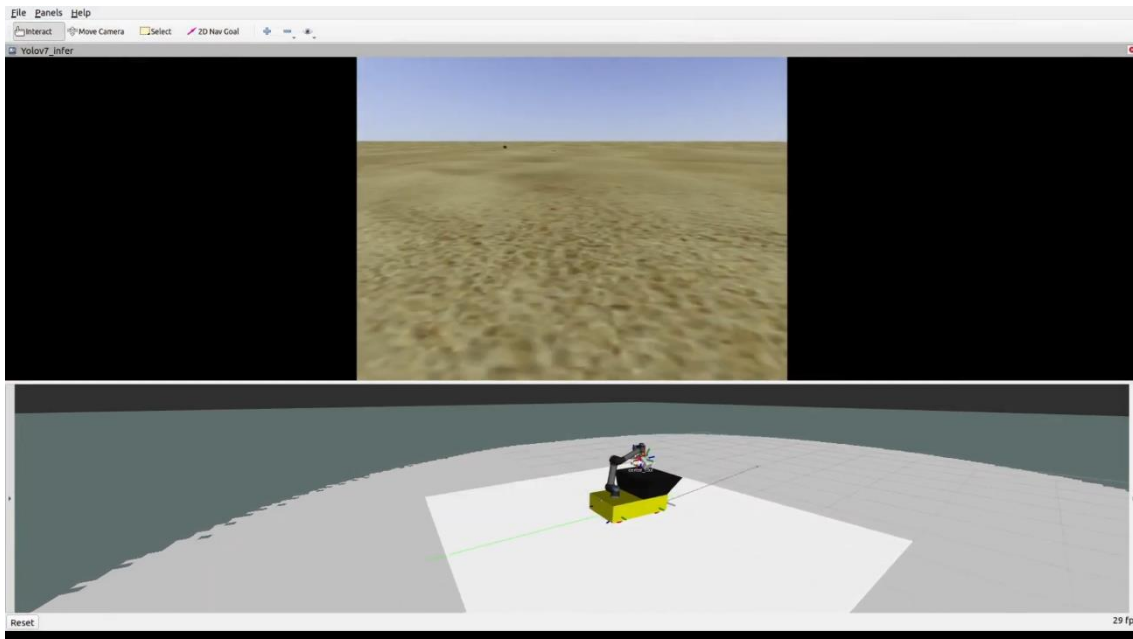


Boxbot is catching the litter using the arm and the gripper, taking into account to the preset end-effector pose offset parameters.

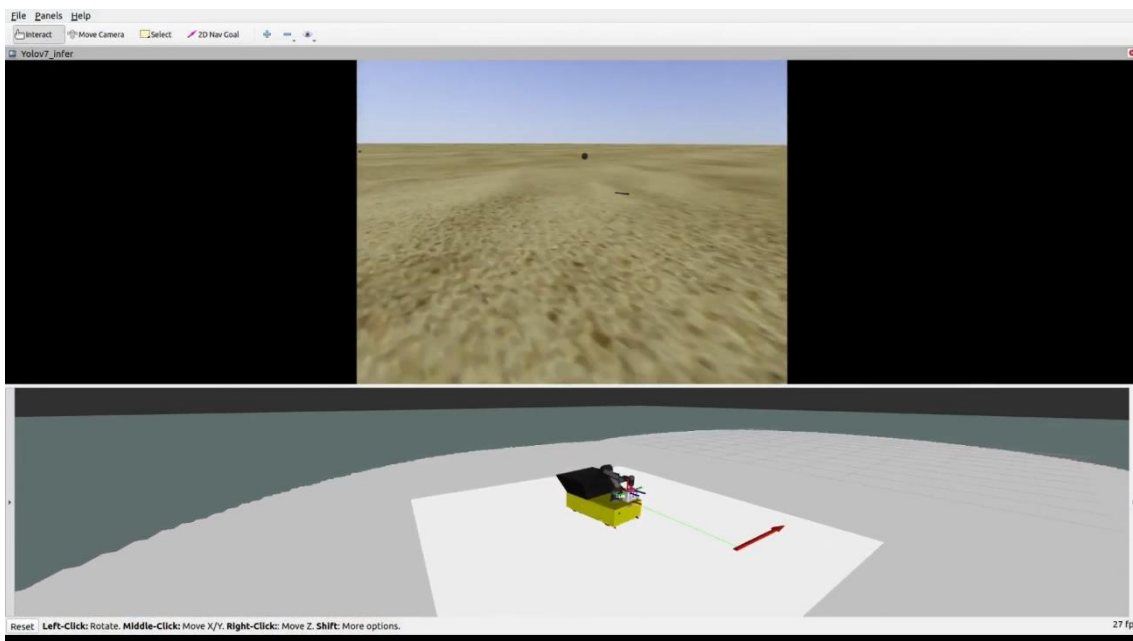


Boxbot's gripper has caught the litter, and the robotic arm moves to deposit the litter in the deposit basket.

## ANNEX I



Boxbot continues its path to the previous waypoint, that is, it resumes the waypoint defined before the litter detection and catch.



In the figure above, a new waypoint is set (red vector) for Boxbot to do the second line of sweeping.