DESIGN OF A MACHINE LEARNING CLASSIFIER-BASED WASTE RETRIEVAL ROBOT

BY

EBUBE OPARA	ENG1603829
EGHOSA MICHAEL OSAYANDE	ENG1607933

UNIVERSITY OF BENIN

BENIN CITY

DECEMBER 2022

DESIGN OF A MACHINE LEARNING CLASSIFIER-BASED WASTE RETRIEVAL ROBOT

BY

EBUBE OPARA	**ENG1603829**
EGHOSA MICHAEL OSAYANDE	**ENG1607933**

A PROJECT SUBMITTED TO THE DEPARTMENT OF ELECTRICAL/ELECTRONIC ENGINEERING, FACULTY OF ENGINEERING, UNIVERSITY OF BENIN, BENIN CITY, EDO STATE.

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF ENGINEERING (B.ENG) DEGREE IN ELECTRICAL/ELECTRONIC ENGINEERING

DECEMBER 2022

CERTIFICATION

This is to certify that this project work was carried out by Ebube Opara ENG1603829 and Michael Eghosa Osayande ENG1607933, students of the DEPARTMENT OF ELECTRICAL/ELECTRONIC ENGINEERING, UNIVERSITY OF BENIN, in partial fulfillment of the requirements for the award of the BACHELOR of ENGINEERING DEGREE in ELECTRICAL/ELECTRONICS ENGINEERING; under the supervision of *PROF MRS PATIENCE E. ORUKPE and CO-SUPERVISED BY ENGR EDOSA OSA*

PROF MRS PATIENCE E. ORUKPE	***PROF MRS PATIENCE E. ORUKPE***
PROJECT SUPERVISOR	HEAD OF DEPARTMENT

ENGR EDOSA OSA
CO-SUPERVISOR

DEDICATION

This work is dedicated to Almighty God for his guidance and protection all through this research work and seeing the completion a successful one. This project is also dedicated to our parents and guardians who gave us the opportunity and encouragement to grow and reach the end of an important phase in our lives.

ACKNOWLEDGEMENT

We would like to acknowledge Almighty God who gave us the guidance to choose Electrical/Electronic Engineering as a course of study and also see us through right to the very end of this bachelor’s degree.

We would like to say that there are so many people to thank, however, we will like to thank the Electrical/Electronic Engineering department staff, University of Benin, for providing the platform on which we were engaged with this project.

We want to say a big thank you to our project supervisors, Prof. (Mrs.) Patience E. Orukpe and Engr. E. Osa for their constant support, patience and guidance. A big thank you to the Head of Department (Electrical/Electronic Engineering) Prof. (Mrs.) Patience E. Orukpe. We will also like to say a warm thank you to Dr F. O. Agbontean who kicked started this project with us.

We are grateful to all staff and students of the department of Electrical/Electronic Engineering for making our period in University of Benin a time worthwhile and a successful one.

We want to thank our families for the encouragement and financial support throughout our time spent in the University of Benin, they encouraged us all through our years of learning and also appreciate our siblings for their care and vital support.

We are highly grateful.

ABSTRACT

The need to control and curb the litter of plastic waste, specifically plastic bottles, in the environment has risen with the continuous influx of plastic in the ecosystem. Their non-biodegradable nature makes it difficult for them to be destructible and harms the environment. To effectively manage and dispose of the excess plastic in the environment, an effective plastic waste collector is necessary. While developed countries have an efficient system of management, developing and underdeveloped countries struggle with setting up an organized system due to cost. Hence, the aim of this work is to design and simulate a simple and cheap machine-learning classifier-based waste retrieval robot to help collect and retrieve plastic bottles in their immediate environment.

The robot and test environment were designed in 3D making use of Webots simulation software. These were then used to train a Haars-Cascade classifier that was able to detect the test bottles. The classifier training was done using Google Collab and the robot control was developed and implemented using the Python programming language.

The classifier was trained for 14 levels and detected 203 bottles (189 true positive and 14 false positives) bringing the efficiency of our final Haars Cascade classifier to 93.10%.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND OF STUDY

Owing to the low cost of production of plastic and its indestructible and nonbiodegradable nature, there has been a consistent increase in the use of plastic for packaging and storage in manufacturing.

The increase in the use of plastic has led to an alarming increase in plastic waste. While this is expected, its non-biodegradable nature makes it last for considerably long periods of years without disintegrating.This slow disintegrating process has led to plastic saturation in the ecosystem which can only be managed by reducing their production (Geyer, R., Jambeck, J. R. & Law, K. L., 2017).

To achieve this, organizations would be required to recycle plastic waste instead of producing more. However, the retrieval and collection of plastic waste have been challenging to manage over the years.

A significant challenge faced in waste management is the separation of plastic waste from other types of waste at the point of disposal (Lee, J., & Park, J., 2017).

In most developing countries, there is a lack of an efficient waste management system, and creating an easy to understand system would require a collective effort of a fully committed government, the general public and private organisations.

There is a need for precise sorting of plastic waste to allow recycling and technology for sorting plastic waste especially in developing countries is rare and expensive by their standards. (Haider, J., & Azhar, A. 2019)

The sorting is usually done manually which is not an efficient method of sorting plastic from a waste pile. There is a need for a more efficient alternative of detecting plastic waste from a pile to make sorting easier.

To combat this, an efficient method that can be considered is the use of programmed robots for obstacle detection, especially plastic (Ejimofor et al., 2022).

Robots are widely used in industrial settings for performing tasks, particularly in hazardous environments (Gasser, L., 2017). Robots usually work with artificial intelligence to have human-like senses such as vision, touch, temperature sensing etc. (Britannica, n.d.)

Although most robots are not self-sufficient, artificial intelligence increases their functionality and they can be useful for plastic detection.

By employing robotics, which is the design and construction of robots to carry out tasks traditionally done by human beings, there is a possibility of creating a cheaper and more efficient method for plastic waste detection and retrieval.

1.2 PROBLEM STATEMENT

The increase in plastic waste in the ecosystem is at an alarming rate and there is a need for active detection and retrieval for recycling purposes.

Most developing countries do not have the economic or infrastructural means to build effective waste management systems. (The World Bank)

1.3 AIM

The aim of this work is to design an economic robot that can detect and retrieve plastic waste material objects utilising a machine learning classifier model.

1.4 OBJECTIVES

The objective of this project for plastic waste detection are:

1. To design and develop a 3D model of the test environment to run the simulation
2. To design and develop a 3D model of the robot
3. To train a machine learning classifier to detect the test 3D litter model
4. To design the robot control
5. To simulate the designed robot in the test environment

1.5 METHODOLOGY

1. The 3D model of the test environment will be performed using the Webots simulation software.
2. The 3D modeling of the robot will be performed using the Webots simulation software.
3. Data for training the machine learning model will be derived from the 3D models designed and the training will be done in a cloud environment i.e., Google collab. Python programming will be used, the Haars-Cascade classifier model. The model will be validated based on its accuracy.
4. The robot control will be developed and implemented using the Python programming language
5. The robot operation in the test environment will be simulated using the Webots simulation software.

1.6 SCOPE OF WORK

This work is limited to the retrieval of can bottles in developing countries such as Nigeria.

1.7 SIGNIFICANCE OF WORK

The project is designed to contribute to the efforts of plastic waste management and in turn, reduce plastic pollution.

With the use of cost-friendly materials, we want to develop a means for localized plastic waste management which can be improved upon with further work.

CHAPTER TWO

LITERATURE REVIEW

2.1 LITERATURE REVIEW

Computer Vision is one of the most powerful and compelling types of AI and we've almost experienced it in any number of ways without even knowing. The concept of computer vision is based on teaching computers to process an image at a pixel level and understand it (Shapiro, L.G., & Stockman, G.C., 2001).

Computer vision is the field of computer science that focuses on mimicking human vision by enabling computers to identify and process objects in images and videos in the same way that humans do (G. R.Girshick, et al., 2014). Computers are trained to recognize images from the vast available data of images on the internet. The algorithms were built to mimic the image recognition and processing by the human eye (J. Deng, et al., 2009).

Computer vision is all about pattern recognition. In a day, approximately 3 billion images are shared and algorithms have been trained to identify items based on their attributes. Attributes of an image are defined and stored in datasets and these attributes are compared with the extracted features of images the algorithm has been trained with (Garg, 2021).

There are common tasks that computer vision can be used for and they are (Szeliski, 2010):

1. **Object classification:** Systems with computer vision can inspect visual content like photos and videos and classify the object on a photo/video to a pre-defined category. A typical example is when the system can find a dog among all objects in the image.
2. **Object identification:** For identification, the system scans photos and videos and can point out a particular object in them. For example, the system can find a specific dog among the dogs in the image.

3. **Object tracking:** For this function, the system will process a video, find its object or objects of interest from the search criteria given and track its movement.

2.1.1 PATTERN RECOGNITION IN IMAGE PROCESSING

Pattern recognition is simply the computation of any given data by comparing it with existing knowledge in a database for similarities to determine the object. It has taken years of development from the 1960s up to this point and it includes a lot of methods and computations. The point of pattern recognition is for a system to be able to accurately describe an object based on predetermined characteristics (Duda, R. O., & Hart, P. E., 1973).

Pattern recognition has different subsections and they are given below:

1. Statistical Pattern Recognition

Statistical decision and estimation is based on feature vector distribution and it is a probability and statistical model. The statistical model is characterised by a family of class-conditional probability density functions. The features are put in an optional order and the set of features can be regarded as a feature vector. This method of pattern recognition deals with features only and does not consider the relationship between features (Bishop, C., 2006).

2. Data Clustering

This is an unsupervised method of machine learning and the aim is to find similar clusters from a mass data. There is usually no prior knowledge of the clusters in this method. This method is separated into two - hierarchical clustering and partitional clustering. (Dhillon, I.S., 2001)

3. Fuzzy Sets

This method follows the logic that the thinking process and languages of human beings are usually fuzzy and uncertain. Realistically, it is not always possible to have complete answers or classifications. The use of fuzzy sets in pattern recognition started in 1966, where the two basic operations, i.e. abstraction and generalisation were proposed by Bellman, R., Zadeh, L.A., & Mokhatab, R., 1966.

4. Neural Networks

Neural networks have been developing quickly since the first neural networks model was proposed in 1943. The approach of neural networks applies biological concepts to machines for pattern recognition. This led to the creation of artificial neural networks which is set up by trying to physiologically model the human brain (McCulloch, W. S., & Pitts, W., 1943). Neural networks are made up of different associate units. In addition, genetic algorithms applied in neural networks are statistical optimised algorithms proposed by Holland in 1975.

Neural Pattern Recognition has a higher appeal because it needs minimum prior knowledge. In any situation where it has enough layers and neurons, an ANN can create any complex decision region.

5. Haar's Cascade Classifiers

Haar's cascade classifiers are classifiers that were first used for real-time face detection then subsequently used for object detection. It is a machine learning object detection algorithm that identifies objects in an image and video.

A detailed description of Haar's classifiers can be seen in Paul Viola and Michael Jones's paper "Rapid Object Detection using a Boosted Cascade of Simple Features" (Viola, P., & Jones, M. J., 2004). In the paper, the authors proposed an algorithm that is capable of detecting objects in images, regardless of their location and scale in an image. Furthermore, the algorithm can run in real-time, making it possible to detect objects in video streams. Specifically, the authors focused on detecting faces in images. Still, the framework can be used to train detectors for arbitrary "objects," such as cars, buildings, kitchen utensils, and even bananas.

Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

- Positive images – These images contain the images which we want our classifier to identify.
- Negative Images – Images of everything else, which do not contain the object we want to detect.

One of the primary benefits of Haar cascades is their speed (Viola, P., & Jones, M. J. 2004).

6. Structural Pattern Recognition

This method does not follow feature extraction and segmentation, instead it focuses on the description of the structure. It throws light on how some less complex sub-patterns can make one pattern. The two main methods in structural pattern recognition are syntax analysis and structure matching.

If you take the relationship between each part of the object into consideration, structural pattern recognition is best. It deals with symbol information, and this method can find applications in high-level computation like image interpretation (Cherkassky, V. and Mulier, F., 2007).

Structural pattern recognition is always accompanied by statistical classification or neural networks for dealing with more complex problems of pattern recognition. A simple application is in the recognition of multidimensional objects.

Other forms of pattern recognition include:

1. Syntactic pattern recognition
2. Approximate reasoning approach to pattern recognition
3. A logical combinatorial approach to pattern recognition
4. Applications of Support Vector Machine (SVM) for pattern recognition
5. Using higher-order local autocorrelation coefficients to pattern recognition
6. A novel method and system of pattern recognition using data encoded as Fourier series and Fourier space.

For object detection in image processing for the sake of this project, the machine learning involved is the use of Haar's cascade classifier as it is more widely used for image processing.

2.1.2 IMAGE PROCESSING IN COMPUTER VISION

Image processing is the transformation of an image into a digital form and carrying out several operations to get relevant information out of it. (Gonzalez, R. C., & Woods, R. E., 2017).

Image processing systems treat images as 2D signals to be able to apply predetermined methods of signal processing. There are different types of image processing and some of them are (Gonzalez, R. C., & Woods, R. E., 2017):

1. Object Visualization - This helps the system to find objects that are not visible in the image
2. Object Recognition - This enables the system to differentiate or detect objects in the image
3. Image sharpening and restoration - This is the process of enhancing an original image to a better working quality.
4. Pattern recognition - This is done by simply measuring the various patterns around the objects in the image
5. Image retrieval - In this process, the system browses a database of images for images similar to the original image.

Image processing is a step-by-step process that starts with acquiring an image through camera capture or from an existing image (Gonzalez, R. C., & Woods, R. E., 2017). Several operations are then performed on it to get the desired information out of the image.

Image processing is usually done using machine learning which is a subset of artificial intelligence (Shao, L., Liu, J., Wang, F., & Lai, K. K., 2017). Image processing makes use of deep neural networks to process images and the most commonly used Neural Network is the Convolutional Neural Network (CNN). (Krizhevsky, A., Sutskever, I., & Hinton, G. E., 2017). CNN is commonly used for image recognition and processing. It processes an image block-by-block, from the upper left corner, and moves pixel-by-pixel till the last pixel to complete the verification.

The final results from the block verification are passed through a convolutional layer, where each data element is connected. Passing the results through the convolutional layer leads to the result from the verification data.

This result is what helps the system to identify the image. For a better understanding, the step-by-step process of image processing is explained in details:

1. Image Acquisition

This is the first step in image processing. This involves getting the image from its source, usually a hardware-based source such as optical sensors, digital cameras, webcams, from your computer storage or the internet.

2. Image Enhancement

Image enhancement is the process of highlighting the features of interest in the given image that is unclear and this is usually done by enhancing the brightness or saturation of the image.

3. Image Restoration

Image restoration involves improving the way an image appears. However, unlike image enhancement, mathematical and probabilistic models are used to improve its appearance as opposed to increasing brightness and saturation in image enhancement.

4. Colour Image Processing

Colour image processing includes analysing, transforming and interpreting any visual data given in colour. Colour image processing is divided into two major areas:

- Full-colour image processing: in full-colour processing, images are gotten from full colour sensors or cameras. This is the colour image processing that is usually in use.
- Pseudo-colour image processing: For pseudo-colour processing, false colours are assigned to a monochrome image.

The focus here is on assigning a colour to a particular monochrome intensity or range of intensities.

5. Wavelets and Multiresolution Processing

This is the use of wavelets transform to analyse a visual signal i.e., image into different frequency components in various degrees of resolution (i.e. multiresolution). This allows features that are not detected at one resolution to be easily spotted at another resolution.

6. Compression

Compression is reducing the size of an image in order to reduce the storage required to save it or the bandwidth required to transmit it. This is done to most images on the internet.

7. Morphological Processing

Morphological processing is a set of processing operations for processing images based on their shapes. It applies a structuring element to an input image to create an output image of the same size.

8. Segmentation

Segmentation is an essential procedure for object detection, image recognition, feature extraction, and classification tasks. depend on the quality of the segmentation process.

It involves dividing an image into various segments to increase the effectiveness of pattern recognition. Segmentation is one of the most difficult steps of image processing.

9. Representation and Description

Image representation and description is critical for successful detection and recognition of objects in a scene.

After an image has been separated into object and background regions, each region needs to be represented and described in a way that makes it easy for computer processing. This is especially useful during pattern recognition. Representation handles the characteristics of the image and its regional properties. While, description deals with extracting information that helps the system to differentiate classes of objects.

10. Recognition

Recognition simply assigns a label to an object based on its description. It is the process of identifying an object or a feature in any given visual data.
These are the steps images have to go through to be recognised by an Artificial Neural Network (ANN) designed for image processing such as CNN.

2.1.3 Open CV

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

Being a Berkeley Software Distribution (BSD) licensed product, OpenCV makes it easy for businesses to utilise and modify the code. It is a collection of functions and classes that implement many Image Processing and Computer Vision algorithms. OpenCV is a multi-platform API written in ANSI C and C++, hence it is able to run on both Linux and Windows. The library has more than 2500 optimised algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms (Fernández Villán, A. 2019).

These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. (Joshi, P. 2018).

2.1.4 ROBOT SIMULATION PACKAGES

Simulation has become an important part in design or development for an autonomous robot. Most simulation comes with accurate physic simulation, multi platform, high quality rendering and open source code. There are several reasons indicated that computer simulation is essentially tool for robotics field, particularly for mobile autonomous robot. Firstly, robots are fragile and hard to test and develop due to their complexity. The ability to test and develop in simulation as in real world environment can reduce the risks and cost of hardware failure. Moreover, there will be costly in develop a robot if the risk of failure is high. Simulations allow us to test the robot in software while waiting for the delivery of components. 5 In addition, with simulation, we got the ability to try our robot in difference kind of environment which not feasible to create in reality, such as fire and disaster scenarios. Finally, the most attraction for the software simulation was the ability to test run the robot as many time as we want. This process is needed for developing a complex algorithm and programms and disadvantages.

Table 2.1: Summary of the capabilities of some popular robotic simulation packages

Capability	Simulation Package			
	USARSim	Gazebo	Webots	JMEsim

Graphics fidelity	Very good	Very good	Excellent	Excellent
Physical Accuracy	Excellent	Good	Good	Excellent
Integration with ROS	Yes	Yes	Yes	Yes
Environment and robot models	Extensive	Some	Comprehensive	Some
Licensing	Commercial	Open source	Commercial	Open source
Multi platform	No	No	Yes	Yes

USARSim

USARSim (Urban Search and Rescue Simulator) is a robot simulation software developed for urban search and rescue (USAR) scenarios. It provides a comprehensive platform for the simulation of robots in complex and realistic environments, enabling the testing of new technologies and techniques before they are deployed in real-world USAR operations. USARSim is designed to support the development and testing of robots, sensors, and algorithms for various USAR missions such as exploration, localization, and navigation in urban environments.

USARSim is built on the Unreal Tournament game engine and provides an accurate simulation of the physical and environmental characteristics of urban environments. The software includes a large library of models, including buildings, vehicles, and other objects found in USAR scenarios, allowing users to create and customize their own environments. The software also provides a wide range of sensors, such as cameras, laser range finders, and microphones, which can be attached to the simulated robots to provide real-world sensor data.

One of the key features of USARSim is its ability to simulate multi-robot systems, allowing users to test and evaluate the coordination and collaboration of multiple robots in a single environment. The software provides a comprehensive API for the development of new algorithms and techniques for multi-robot systems, allowing users to quickly test and evaluate new ideas in a virtual environment.

USARSim also supports the development of autonomous robots, providing a platform for the testing of new algorithms and techniques for autonomous navigation, mapping, and exploration. The software provides a wide range of tools for the development and evaluation of autonomous systems, including visual odometry, global positioning systems (GPS), and inertial navigation systems (INS). USARSim has been widely used in both academia and industry for the development and evaluation of new technologies and techniques for USAR operations. The software has been used in numerous research projects, including studies on multi-robot coordination and collaboration, autonomous navigation and mapping, and human-robot interaction in USAR scenarios (Eggers, Braun, & Worn, 2008; Jain, Hall, & Stentz, 2005). In addition, USARSim has been used by a number of companies and organizations for the development and testing of new USAR technologies, including robots, sensors, and algorithms (Airobotic Systems, n.d.; Urban Robotics, n.d.).

USARSim is a comprehensive and versatile robot simulation software that provides a platform for the development and testing of new technologies and techniques for USAR operations. The software offers a wide range of capabilities and tools, including multi-robot simulation, autonomous navigation and mapping, and the development of new algorithms and techniques. USARSim has been widely used in both academia and industry, providing a valuable tool for the development and evaluation of new technologies for USAR operations.

Gazebo

Gazebo is a 3D multi-robot simulation environment that provides a comprehensive set of tools for simulating robots and their interactions with their environments and with each other. It is developed and maintained by Open Robotics and has a large community of contributors and users. Gazebo provides a range of features that allow for the simulation of a wide variety of robotic systems and environments, making it a popular choice for robot simulation and development.

Gazebo provides a realistic physics engine that enables the simulation of rigid body dynamics and collision detection. It also supports the modeling of complex sensor configurations, including cameras, lidars, and sonars. These features, combined with its ability to simulate a variety of robotic platforms, such as wheeled robots, flying robots, and underwater robots, make Gazebo a versatile tool for robot development and testing.

Gazebo also provides a flexible plugin architecture that enables the integration of custom components and algorithms into the simulation. For example, this allows for the simulation of custom sensors, actuators, and control algorithms, providing an effective way to test and refine these components prior to deployment on physical robots.

In addition to its simulation capabilities, Gazebo also provides a powerful graphics engine that enables the visualization of simulated environments and robots. The graphics engine supports the display of high-fidelity models, textures, and lighting effects, making it possible to generate photorealistic images and animations of the simulated robots and environments.

Gazebo has been widely adopted in the robotics community and has been used in a variety of applications, including robot design and development, testing of control algorithms, and performance evaluation. For example, it has been used in the development of autonomous vehicles, UAVs, and underwater robots (Ricardo, et al., 2018; Bandyopadhyay, et al., 2019; Wang, et al., 2019).

Despite its popularity and versatility, Gazebo does have some limitations. For example, it does not support the simulation of soft body dynamics, which can limit its usefulness for the simulation of certain types of robots, such as flexible robots or robots that use soft materials. Additionally, its physics engine may not provide the level of accuracy and detail required for certain applications, such as the simulation of high-precision robotic systems.

Gazebo is a powerful and flexible simulation environment that provides a comprehensive set of tools for simulating robots and their interactions with their environments. Its ability to simulate a wide range of robotic platforms and environments, combined with its flexible plugin architecture and high-fidelity graphics engine, make it a popular choice for robot development and testing.

JMonkey Engine Software Development Kit

The JMonkey Engine Software Development Kit (SDK) is an open-source, cross-platform, and high-performance 3D game engine used for developing interactive 3D games, simulations, and virtual reality applications. It provides a robust set of tools and features that make it a popular choice among game developers and robotics engineers for simulating robots.

The JMonkey Engine SDK is written in Java and provides a powerful graphics pipeline that supports real-time rendering of 3D scenes, including advanced lighting and shading techniques, physics simulations, and animated characters. It has a modular architecture that allows for easy customization and integration with other software tools. The JMonkey Engine SDK also includes a comprehensive set of tools for creating and manipulating 3D models, including 3D modeling, rigging, and animation tools.

One of the key features of the JMonkey Engine SDK is its physics simulation capabilities. The engine includes a built-in physics engine that supports realistic simulation of rigid bodies, soft bodies, and cloth. This makes it possible to simulate robots with detailed and accurate physical properties, including mass, friction, and collision detection. The physics engine can also be integrated with other physics simulation tools, such as the Bullet physics engine, for even more advanced simulation capabilities.

Another important aspect of the JMonkey Engine SDK is its support for multi-threading and parallel processing. This enables the simulation of large and complex environments, such as entire factories or cities, with a high degree of accuracy and performance. The JMonkey Engine SDK also provides a comprehensive set of tools for creating and controlling camera views, including first-person, third-person, and fly-through modes.

The JMonkey Engine SDK also provides support for virtual reality (VR) applications, making it an ideal platform for simulating robots in VR environments. With its support for VR controllers, such as the Oculus Rift and the HTC Vive, the engine provides an immersive and interactive simulation experience. Additionally, the JMonkey Engine SDK includes a scripting API that makes it easy to create custom behaviors and interactions for the simulated robots.

One of the strengths of the JMonkey Engine SDK is its active and supportive community. The JMonkey Engine community provides a wealth of resources and support for users, including tutorials, forums, and a repository of open-source plugins and extensions. The community also provides a platform for users to share their work and collaborate with other users.

The JMonkey Engine SDK is a powerful and versatile tool for simulating robots. With its advanced graphics pipeline, physics simulation capabilities, multi-threading support, and VR integration, the JMonkey Engine SDK provides a comprehensive solution for simulating robots in a variety of environments and scenarios. The active and supportive community also makes it an ideal choice for developers and engineers looking to build innovative and cutting-edge simulations.

Webots

Webots is a cutting-edge software application that is used to model and simulate robots. This powerful platform provides a comprehensive environment for the development, testing, and deployment of various types of robots. Webots is used by many universities, research institutes, and private companies around the world to model and simulate their robots in a virtual environment, allowing them to test their robots in a controlled setting before deploying them in the real world.

One of the main advantages of Webots is its ability to accurately model the physical characteristics of robots, including their geometry, mass, and other properties. This allows developers to test the dynamics of their robots and fine-tune their control algorithms before deployment. Webots also provides a visual interface that allows developers to interact with the simulated robots, making it easier to test and debug their systems.

Webots also provides advanced features for modeling and simulating environments. This includes the ability to model objects and terrain, as well as lighting and environmental conditions. This allows developers to simulate their robots in realistic environments, giving them a better understanding of how their robots will behave in the real world.

One of the key benefits of using Webots for robot simulation is its versatility. The software supports a wide range of robots, including wheeled robots, legged robots, aerial robots, and more. Additionally, Webots provides a flexible and open architecture, allowing developers to extend the software with their own custom plugins and components.

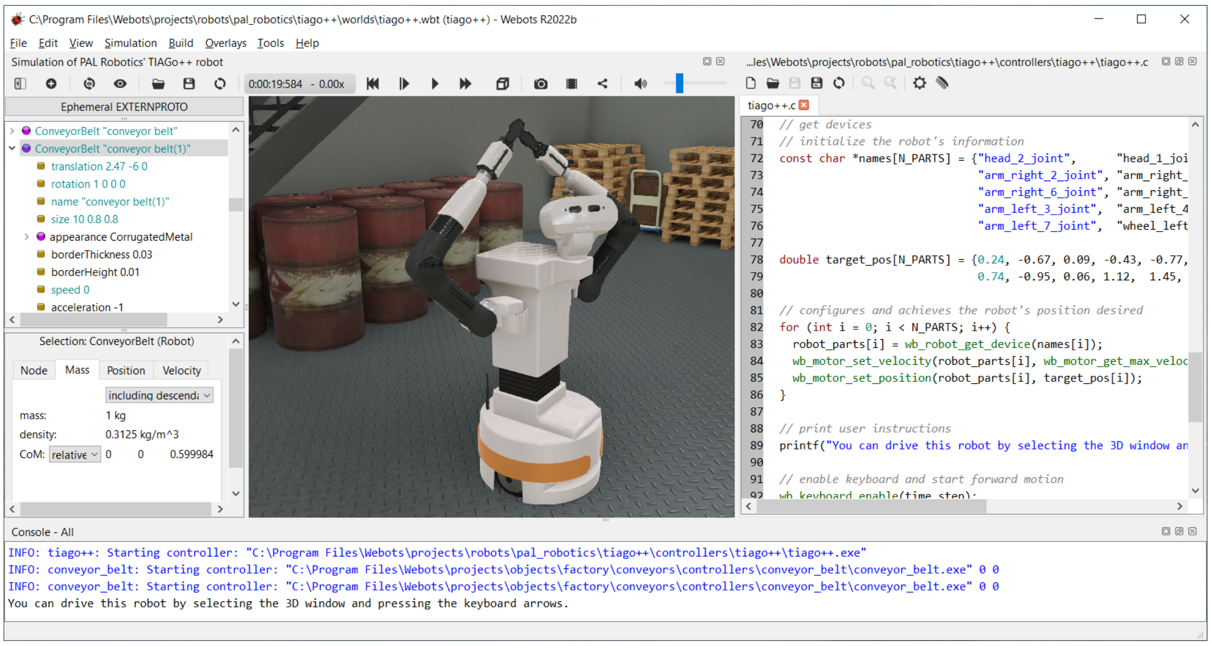
In addition to its powerful simulation capabilities, Webots also provides a range of tools for the development and deployment of robots. This includes a graphical user interface (GUI) for programming robots, as well as a scripting environment for automating repetitive tasks. Webots also provides tools for data analysis, allowing developers to visualize and analyze the results of their simulations.

Webots is widely used in the academic community, with many universities and research institutes using the software for their research in robotics. For example, Webots has been used in research on robotic walking (Liao, et al., 2018), grasping (Liu, et al., 2018), and perception (Lin, et al., 2019).

Webots is a versatile and powerful software application that provides a comprehensive environment for the development, testing, and deployment of robots. With its ability to accurately model physical characteristics and environments, Webots provides developers with a valuable tool for simulating and testing their robots in a virtual environment.

2.1.5 WEBOTS INTERFACE

This overview will cover the main components of the Webots interface and how they are used.


C:\Program Files\Webots\projects\robots\pal_robotics\tiago++\worlds\tiago++.wbt (tiago++) - Webots R2022b
File Edit View Simulation Build Overlays Tools Help
Simulation of PAL Robotics' TIAGo++ robot
0:00:19:584 - 0.00x
Ephemeral EXTERNPROTO
...les\Webots\projects\robots\pal_robotics\tiago++\controllers\tiago++\tiago++.c
tiago++.c
Selection: ConveyorBelt (Robot)
Node
Mass
Position
Velocity
including descend
Console - All

```
ConveyorBelt "conveyor belt"
ConveyorBelt "conveyor belt(1)"
  translation 2.47 -6 0
  rotation 1 0 0 0
  name "conveyor belt(1)"
  size 10 0.8 0.8
  appearance CorrugatedMetal
  borderThickness 0.03
  borderHeight 0.01
  speed 0
  acceleration -1
```

```
mass:     1 kg
density:  0.3125 kg/m^3
CoM: relative  0  0  0.599984
```

```c
// get devices
// initialize the robot's information
const char *names[N_PARTS] = {"head_2_joint",      "head_1_joi
                              "arm_right_2_joint", "arm_right_
                              "arm_right_6_joint", "arm_right_
                              "arm_left_3_joint",  "arm_left_4
                              "arm_left_7_joint",  "wheel_left

double target_pos[N_PARTS] = {0.24, -0.67, 0.09, -0.43, -0.77,
                              0.74, -0.95, 0.06, 1.12,  1.45,

// configures and achieves the robot's position desired
for (int i = 0; i < N_PARTS; i++) {
  robot_parts[i] = wb_robot_get_device(names[i]);
  wb_motor_set_velocity(robot_parts[i], wb_motor_get_max_veloc
  wb_motor_set_position(robot_parts[i], target_pos[i]);
}

// print user instructions
printf("You can drive this robot by selecting the 3D window an

// enable keyboard and start forward motion
wb_keyboard_enable(time_step);
```

```
INFO: tiago++: Starting controller: "C:\Program Files\Webots\projects\robots\pal_robotics\tiago++\controllers\tiago++\tiago++.exe"
INFO: conveyor_belt: Starting controller: "C:\Program Files\Webots\projects\objects\factory\conveyors\controllers\conveyor_belt\conveyor_belt.exe" 0 0
INFO: conveyor_belt: Starting controller: "C:\Program Files\Webots\projects\objects\factory\conveyors\controllers\conveyor_belt\conveyor_belt.exe" 0 0
You can drive this robot by selecting the 3D window and pressing the keyboard arrows.
```

Figure 2.1: Webots Interface

1. Workspace and Project Manager:

The Workspace and Project Manager provides a visual representation of the entire project and all the components within it. It allows users to create, manage and access projects, and includes a list of all the worlds, robots and controllers used in the project.

2. 3D View:

The 3D View is the main window for visualizing and interacting with the simulated robots and environments. It displays the 3D environment and the robots, and offers tools for controlling camera movement and zoom.

3. Node Tree View:

The Node Tree View displays all the nodes within the selected world, robot or controller. Nodes are the basic building blocks for constructing robots and environments, and the Node Tree View allows users to add, delete or modify nodes.

4. Properties View:

The Properties View provides detailed information about the selected node, and allows users to change its properties. It also displays information such as the node's name, position and orientation in 3D space.

5. Time Step Control:

The Time Step Control allows users to control the simulation speed and pause it if necessary. It displays the current time step and provides controls for adjusting the simulation speed and duration.

6. World Info:

The World Info displays the world settings and provides information about the current environment, such as its size, gravity, and sky color.

7. Console:

The Console displays information about the simulation and any errors that occur during the simulation. It is a useful tool for debugging and diagnosing issues with the simulation.

8. Text Editor

The text editor in Webots is a fundamental component of the development environment, allowing users to write, edit and run controllers. The text editor provides a rich set of features including syntax highlighting, auto-indentation, code completion, and line numbering, making it easier for users to write, debug, and test their controllers.

2.2 RELATED WORK

In the work by Ali, K., et al, 2018, the authors presented a system for detecting and classifying waste in real-time using smart garbage bins. The authors of the paper developed the system as a way to improve the efficiency and sustainability of waste management.

The system uses deep learning for waste detection and classification, which involves training artificial neural networks on large amounts of labeled data. The authors used a convolutional neural network (CNN) for this purpose, and trained the CNN on a dataset of images of different types of waste. Once the CNN was trained, it was able to classify new images of waste in real-time as they were fed into the system through a camera mounted on the smart garbage bin.

The authors conducted experiments to evaluate the performance of the system, and their results showed that it was able to achieve high accuracy rates for waste detection and classification. They also demonstrated that the system was able to operate in real-time, meaning that it was able to process and classify waste as it was being thrown into the garbage bin.

Overall, the main contribution of this research is the development of a system for real-time waste detection and classification using deep learning and smart garbage bins. The system has the potential to improve the efficiency and sustainability of waste management by enabling the automatic sorting and processing of waste.

This approach differed from ours in that it used deep learning rather than a Haar's cascade classifier, and it was implemented in smart garbage bins rather than an autonomous robot. One limitation of this approach was that it required a large amount of labeled data for training the deep learning model, which can be time-consuming and costly to acquire. Our project addressed this limitation by the Haar's cascade classifier which require less training data.

In the work by Al-Otaibi, M., et al. (2018), the authors presented the design and implementation of an autonomous waste sorting robot. The robot is designed to use computer vision and machine learning techniques to identify and classify different types of waste, and then manipulate the waste using robotic arms for sorting.

The authors of the paper developed the robot as a way to improve the efficiency and sustainability of waste management. They designed the robot to be autonomous, meaning that it can operate without human intervention, and to be able to manipulate and sort different types of waste using robotic arms. The robot uses a camera and machine learning algorithms to identify and classify different types of waste, and then manipulates the waste using robotic arms to sort it into different bins or containers.

The authors conducted experiments to evaluate the performance of the robot, and their results showed that it was able to accurately identify and classify different types of waste. They also demonstrated that the robot was able to manipulate the waste using its robotic arms, and sort it into different bins or containers.

Overall, the main contribution of this research is the design and implementation of an autonomous waste sorting robot that uses computer vision and machine learning techniques, as well as robotic arms, for waste manipulation and sorting. The robot has the potential to improve the efficiency and sustainability of waste management by enabling the automatic sorting and processing of waste.

This approach differed from ours in that it used machine learning techniques and robotic arms for waste manipulation, rather than relying on a Haar cascade classifier and a camera feed for detection and retrieval. One limitation of this approach was that it required the use of complex and expensive equipment, such as robotic arms, which may not be practical or feasible for all applications. Our project addressed this limitation by using a revolving plane for ingesting the waste, which is simpler and more cost-effective to implement.

In the work by Zhang, J., et al. (2019), the authors presented a proposal for a waste detection and classification system using convolutional neural networks (CNNs). The system is designed to be robust, meaning that it is able to accurately classify different types of waste even under challenging conditions, such as variations in lighting, background, and orientation.

The authors of the paper propose using CNNs for waste detection and classification because they are able to achieve high accuracy rates when trained on large amounts of labeled data. The authors trained a CNN on a dataset of images of different types of waste, and then tested the CNN on new images to evaluate its performance. Their results showed that the CNN was able to achieve high accuracy rates for waste detection and classification, and that it was robust to variations in lighting, background, and orientation.

The main contribution of this research is the proposal of a waste detection and classification system using CNNs, which are able to achieve high accuracy rates and are robust to variations in lighting, background, and orientation. The system has the potential to improve the efficiency and sustainability of waste management by enabling the automatic sorting and processing of waste.

The general scope of this project was the proposal of a waste detection and classification system using convolutional neural networks (CNNs). The system was able to accurately classify different types of waste, including drink cans, in real-time. This approach differed from yours in that it used CNNs for waste detection and classification, rather than a Haar cascade classifier. One limitation of this approach was that it required a significant amount of computational resources to train and deploy the CNN model, which may not be feasible for all applications. Our project addressed this limitation by using a Haar's cascade classifier for waste detection and classification, which required fewer computational resources.

CHAPTER THREE

METHODOLOGY AND DESIGN

This section details the methods carried out in order to achieve the aim of the project.

3.1 DESIGN AND MODELING OF THE TEST ENVIRONMENT

Several design softwares such as Webots, JMEsim, USARSim and Gazebo were surveyed base on their feature and capabilities. Webots was then chosen as the virtual robot development software in this project because of tits friendly user interface, wide range of robot models and advanced visualizations.

The test environment designed consisted of the following models:

1. Testing Arena
2. Litter
3. Light sources

TESTING ARENA

The testing arena was made up of two main components namely the floor and four bounding walls. A parquet floor with 10 x 10 m^2 square area was chosen. A parquet floor is a type of hardwood flooring made up of small, geometric pieces of wood that are arranged in a repeating pattern. It had an IBLStrength of 1. IBLStrength is a parameter used in the physics engine of Webots that allows to adjust the resistance of a material to collision impacts and forces, making the simulation more realistic. It is used to specify the strength of a material when it is impacted by a force. A higher value for IBLStrength indicates that the material is more resistant to deformation or breaking when impacted, while a lower value indicates that the material is more easily deformed or broken. Each of the four walls used were 1 meters in height and had a 0.2 meters thickness. It also had an IBLStrength of 1.

Litter

The litter used for the simulation was a bottle can drink. The 3D model of the can used was made available from webots inbuilt 3D models. The dimensions of the can are 0.03175 m x 0.03175 m x 0.1222 m.

LIGHT SOURCES

The sources of light used in the simulation were provided by the TexturedBackground and TexturedBackgroundLight nodes. In Webots, the TexturedBackground and TexturedBackgroundLight nodes are used to define the background appearance in a 3D simulation environment. The TexturedBackground node allows you to specify an image file to be used as the background texture. The texture is projected onto the background plane, which is a large plane located at a far distance from the camera, and it will appear to be infinitely far away. You can specify the position, orientation, and size of the background plane, as well as the position and orientation of the camera. The TexturedBackgroundLight node is similar to the TexturedBackground node, but it also allows you to specify the lighting conditions of the background. You can define the color and intensity of the ambient, diffuse, and specular light sources that illuminate the background. This allows you to create more realistic lighting conditions in the simulation environment.

Both TexturedBackground and TexturedBackgroundLight nodes are used to set up a realistic and visually appealing background in a simulation. They allow you to create a background that has a texture, lighting and shadows which makes the simulated environment much more realistic and similar to the real world. These nodes can be used in a wide range of applications, such as architectural visualization, robotics simulation, and virtual reality.

3.2 DESIGN AND MODELING OF THE ROBOT IN 3D

The modelling of the robot was also done using webots. The robot was designed to perform the task of searching and collecting the waste objects. The robot must first explore the arena in order to find something. It must be capable of avoiding the obstacles (walls), detect bottles and then somehow move the bottle to the recycling area.

LIST OF ROBOT REQUIREMENTS

1. Ability to move inside the test arena
2. No human interaction
3. On-board computation
4. Litter detection
5. Ability to move the litter
6. Obstacle avoidance

FUNCTION SPECIFICATION

1. External

The robot must be able to run automatically without interaction from external controller, and persist its stable behaviors to external noise and disturbance from the surrounding environment.

2. Internal

The robot must be able to move itself in the arena on wheels. It must detect and avoid obstacles, find the litters and retrieve them.

CRITICAL TECHNICAL POINTS

The critical technical points were (in order): the locomotion, obstacle avoidance, and bottle manipulation. We proceeded in this order to solve all these problems so that we could concentrate on one problem at a time.

SOLUTIONS IDENTIFICATION

1. Movement

The first consideration is the robot's locomotion. Several strategies for its movement are listed in Table 3.1, including the specific principles, advantages and disadvantages corresponding to the strategies.

Table 3.1: Movement strategies considerations

Strategy	**Principle**	**Advantage**	**Disadvantage**
Cartesian robot	Moves in x-y axis over the whole terrain	Can move over any terrain	Too expensive
Quadcopter	Exploits the 3rd dimension	Can move over any terrain, can see everything from above	Complicated design
2-wheel differential robot	2 wheeled robot	Easy to control, only 2 motors	Needs a 3rd passive wheel
4-wheel differential robot	4 wheeled robot	Stable mobile base	Requires 4 motors, more complex control
Hexapod	6 legged robot	Navigation through complex terrain	Hard to program, slow, lots of parts

Quadruped	4 legged robot	Can move over any terrain	Statically unstable, difficult to control
Biped	2 legged robot	Can access all terrains	Extremely difficult to program and control. Not statically stable
Synchrodrive	All wheels turn synchronously and the chassis doesn't rotate	None identified	Requires lots of moving parts
Crawling	Chassis consisting of multiple sections with actuated motors inbetween	Can access all terrains	Very difficult to control, expensive and time consuming to make
Hopping	Movement with a series of jumps	Can move over obstacles and large distances fast	Hard to control, to manufacture as there aren't any commercial products
Hovercraft	Movement on an air cushion	Can move over any terrain	Cannot move up slopes, difficult to control, noise, needs a lot of power, difficult to simulate

Through the analysis of the advantages and disadvantages for each listed strategy, the 4-wheel differential robot was chosen because it has a simple structure which is qualified enough for the flat arena, lower cost and more flexibility to add any needed components.

2. Obstacle Avoidance

It is crucial for the robot to do object detection, including the litter and surrounding walls, so that it could do the sequence of behaviors containing avoiding the walls and finding the bottles. Several strategies for object detection are listed in Table 3.2, including the advantages and disadvantages corresponding to the relative strategies.

Table 3.2: Obstacle avoidance strategies considerations

Strategy	**Advantage**	**Disadvantage**
Ultrasound	Linear response with distance, not affected by target materials, surfaces and color. Can detect small objects over long operating distances. Resistance to external disturbances such as vibration, infrared radiation, ambient noise, and EMI radiation	Must view a surface (especially a hard, flat surface) squarely (perpendicularly) to receive ample sound echo. Requires time for the transducer to stop ringing after each transmission burst before they are ready to receive returned echoes. Have a minimum sensing distance. Changes in the environment, target of density, smooth of surfaces affect ultrasonic response
Infrared	Detect infrared light from far distances over a large area. In	Incapable of distinguishing between objects.

	real-time and detect movement.	
Laser	Better accuracy more quickly. Easy alignment by employing visible red laser beam. Detects of very small targets due to small measuring spot size	Suffer from laser noise, stray light, and speckle effects interference.
Camera	Cheaper, more informative and more compact	Limitation of its view fields
Stereovision	Can do 3D vision.	Complex, poor dynamic range and still not very reliable
Millimeter Wave Radar	Accurate, excellent image identification and resolution	Too expensive.

At first we wanted to use only one camera to do everything in order to keep a simple system. The camera could do different kinds of image processing and in theory detect and differentiate all the objects. When testing we saw that the camera could not differentiate between the floor and walls using the color information and was actually quite slow when processing the video stream, this meant we had to use other sensors to complement the camera. In the end we chose to use the camera only for the bottle detection as it could do it reliably and use infra red sensors for wall and obstacle detection.

3. Litter Grasping

After the achievement of finding bottles, the actuator should have the capacity of grasping and storing the bottles, so that the robot could complete the recycling target. There are a lot of practical ways for the robot to the grasp bottles, and several strategies are listed in Table 3.3, including the principles, advantages and disadvantages corresponding to the relative strategies.

Table 3.3: Litter Grasping strategies considerations

Strategy	**Principle**	**Advantage**	**Disadvantage**
Robotic arm	Arm with a few actuators mounted onto the robot	Allows picking up bottle in every position and in every terrain	Difficult mechanical realisation and time consuming programming
Clamp	Grabbing a bottle in front of the robot with a clamp	Easy to realise	Needs good precision in positioning to grab a bottle
Suction	Suction mechanism to hold bottles	Easy pick-up and release	Requires compressor, energy consuming, can be hard to position on bottle
Pushing	Bottle being rolled with the robots chassis	No extra mechanical parts	Can be hard to perform complex movements while keeping bottle pushed. Hard when

			bottle positioned close to an edge or corner
Storage bay for single bottle	Robot ingests bottle inside a storage bay	Easy mechanical implementation, carrying bottle around relatively easy	Must return to base for every single bottle, must be well aligned with the bottle
Deployable cage	Deploying a cage to surround the object, and bring it back to the base. The bottle rolls on the floor	Very easy to implement, bottle position doesn't have to be exact, can grab bottle in any position or orientation	Can't bring bottle over rough terrain, only one bottle at a time
Compressed air	Blowing compressed air on the bottle to move the bottle around	No mechanical moving part	Complex aiming and bottle trajectory planing, requires compressor or to carry compressed air
Scotch	Sticky surface	Cheap, big supply	Adhesive wears off with dust

Net	Throwing a net	Robot doesn’t need to move around much	Requires good precision, launching system, retrieval system
Sweeper	Revolving plane	Cheap and easy to realise	Needs to be in close proximity to the litter

Through the analysis of the advantages and disadvantages for each listed strategy, a combination of the sweeper and the storage bay was chosen, which calls for much more simple mechanical structure, and actuator motion to grasp and store bottles, with less cost and risk but more availability and reliability.

3.3 TRAINING THE MACHINE LEARNING MODEL

Images that were used for training and testing the machine learning classifier were collected from the Webots simulation software. The images contained bottles in varied positions namely; bottles kept vertical to the floor and bottles kept horizontal to the floor at varied angles. The distance of the bottles from the camera ranged from 0 to 10 metres.

Table 3.4: Litter Grasping strategies considerations

Image Description	**Total Images Captured**	**Number used for training**	**Number used for testing**
Images with litter	500	300	200
Images without litter	1000	600	400

Neural Network and template matching were considered for object detection but were decided against because they were too computation-intensive. The Haar's Cascade Classifier was eventually chosen because it was easy to train and less computationally intensive.

Google Co-lab notebook was used to train the classifier in the cloud because it provides better hardware resources than the computers physically available.

A Haar's classifier looks for features of an image in different layers. At the top layer, it looks at features large enough to cover the entire window of the image. And at the bottom layer, it scans for finer details.

Because of this, the model is fast enough to detect the litter in real time at the end and it is quick to reject images whose areas do not match with the features in the top layer. After this, it spends more time to analyse other areas of the image that match with the features of the top-most layer and scrutinises its finer details at the bottom layer.

The accuracy and quality of the model depends more on the data that was used to train the model than the code. So, you need as much quality data as possible to get good results.

The classifier is fed two different types of data which are:

1. **Positive Image**: This will contain the exact image of the plastic bottle we are trying to detect.
2. **Negative Image**: These will be images that are not the plastic bottle we are searching for.

The Machine Learning algorithm has to see a variety of the right and the wrong objects in order for it to learn properly. We got different pictures of our plastic bottle in different conditions from lighting and positions.

The first step we employed was getting our positive and negative images, and after this we wrote some code to make it easy to determine which is which. The code for this can be found at Appendix A. It is advised to manually create folders for your negative and positive images each. With the OpenCV output window focused we press 'f' to capture a screenshot and save it to the positive folder, or 'd' to save a screenshot to the negative folder.

To prepare the negative samples to be used for training, we need to create a text file that lists where all our negative samples can be found and the code can be found in Appendix A. This script can be run manually from the Python console to generate the needed neg.txt file.

After doing this, we do the same thing to generate a similar file for the positive images. However this process is a little complicated compared to the first as it needs to contain additional information like the bounding box coordinates for each object we want to detect in each image.

We used the OpenCV annotation program since it is specifically designed specifically for creating this file.

But this simple *opencv_annotation.exe* command line program is only available on the OpenCV version 3.4. So we installed the newest version of 3.4 to get access to it and other Cascade Classifier programs we might need.

After extracting the ZIP file, we found the executable codes we need in */opencv/build/x64/vc15/bin/* in the location we chose to save the extracted file.

All the programs we used were all in that folder: opencv_annotation.exe, opencv_createsamples.exe, and opencv_traincascade.exe.

Even though we prepared our samples and trained our model with OpenCV 3.4, the classifier will still be usable in newer versions of OpenCV.

Now we can run the annotation program.

The annotation program opened each image in our positive folder one at a time in an OpenCV window. In each image we drew a box around the objects within it that we wanted to detect.

Next we created a vector file from all of our positive annotations.

The *-numPos* has to be an amount lower than the number of samples created by *createsamples*. We get the error message: Can not get new positive sample when the *-numPOS* value is higher than the samples created. In this case, it is advised to either lower your *-numPos* or lower the -*minHitRate* (the default value is 0.995).

A popular suggestion for *-numNeg* is to use half of *-numPos*. Using numbers higher than twice the number of negative to positive sometimes lead to better results.

The *-w* and *-h* must match what the values of the createsamples step.

The more *-numStages* the longer it will take to train the classifier.

We trained the classifier for 10 to 19. As the stages increased, the time it took to run became longer.

3.4 DESIGN OF THE ROBOT CONTROL

According to the targeted functions, the design theory and flowchart is shown as Figure 3.1. The obstacle avoidance was the first thing we had to implement, this meant that the robot was able to roam around without hitting anything and thus the next step of actually detecting bottles could be implemented. The obstacle detection is done using 4 infra-red sensors on the front of the robot.

The Python Programming Language was used for the robot controllers. We decided to use Python because: we were familiar with it and it has a lot of libraries. The robots controller code is presented in Appendix A.

The robot's default state is to search for any litter . When the robot detects a litter, it focuses on it in the picture and sends commands to move the robot either left or right to centre the picture of the plastic bottle it focused on. When the robot centres the plastic bottle in the image detected, it moves in a straight line towards the bottle until it longer sees the image of the bottle. When this happens, it means the plastic bottle is in front of the robot. When the image of the bottle disappears, the robot proceeds to retrieve the plastic bottle into the storage bay, after which the cycle repeats starting from searching for bottles. When the robot detects multiple bottles, it centres the first and gives it priority in retrieval before moving on the other bottles.

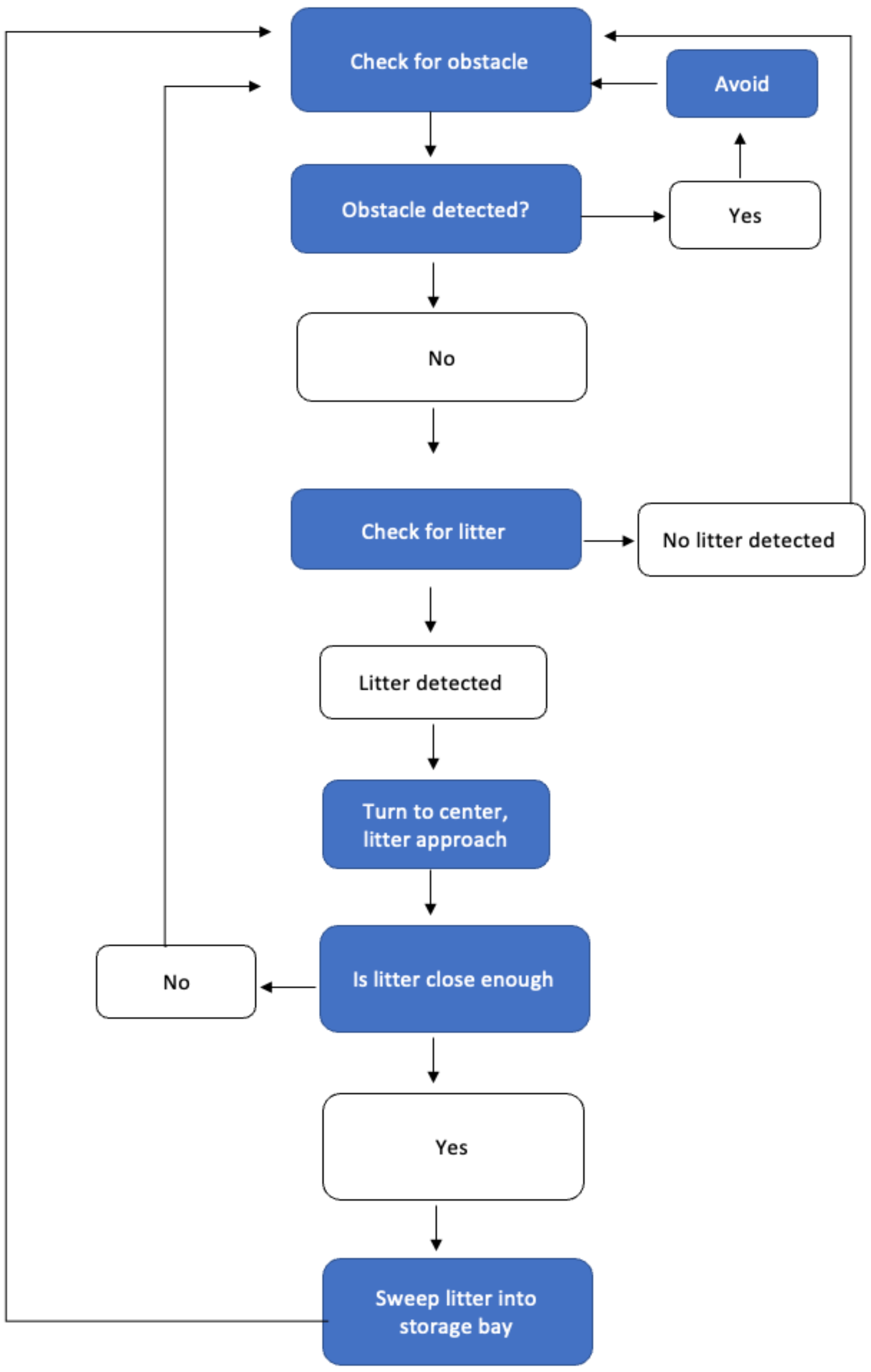


Figure 3.1: Flowchart of robot working

CHAPTER 4

RESULTS AND DISCUSSIONS

This section details the results obtained from the methodologies used.

4.1 TEST ENVIRONMENT MODELING RESULTS

The results from modeling the test environment are presented in the figures following

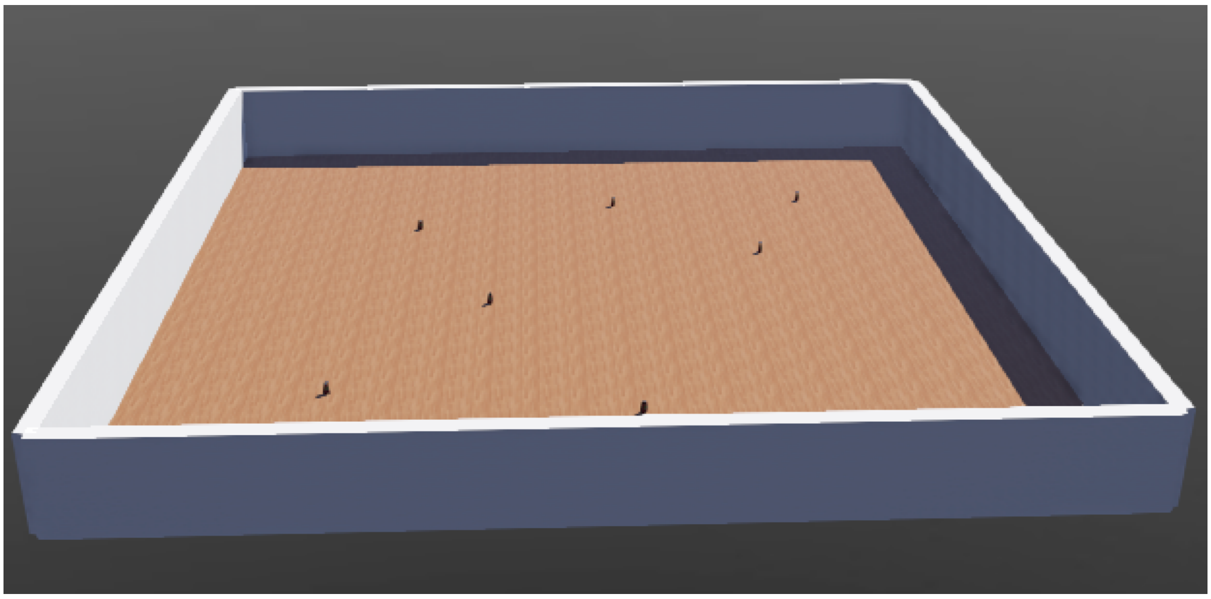


Figure 4.1: The test arena with floor and bounding walls

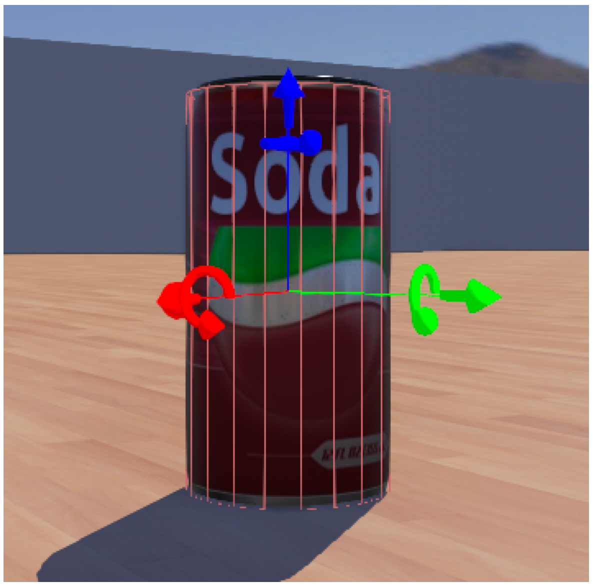


Figure 4.2: Litter (Can Bottle) – Front View

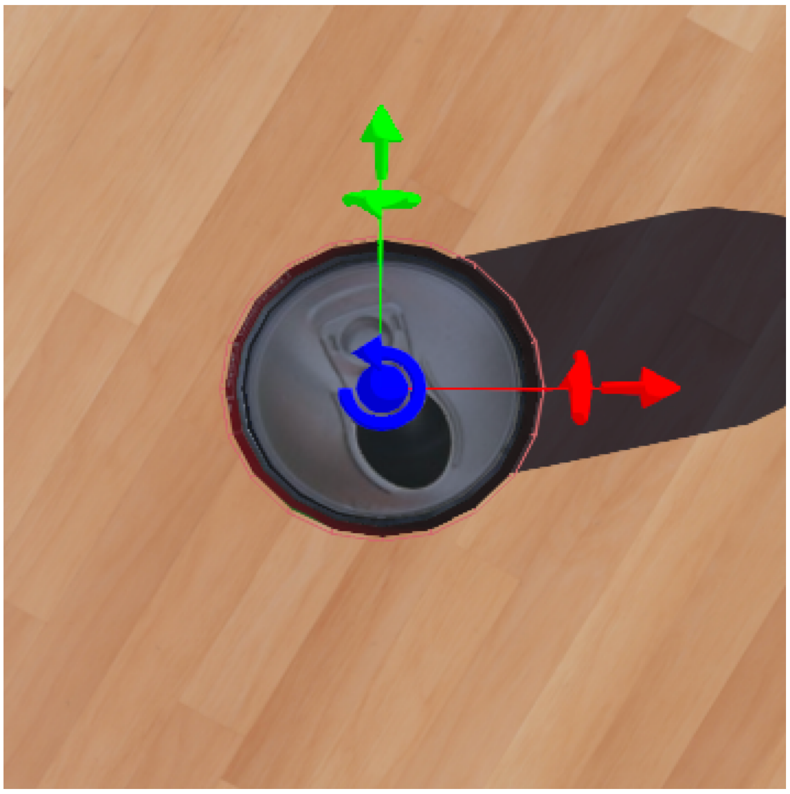
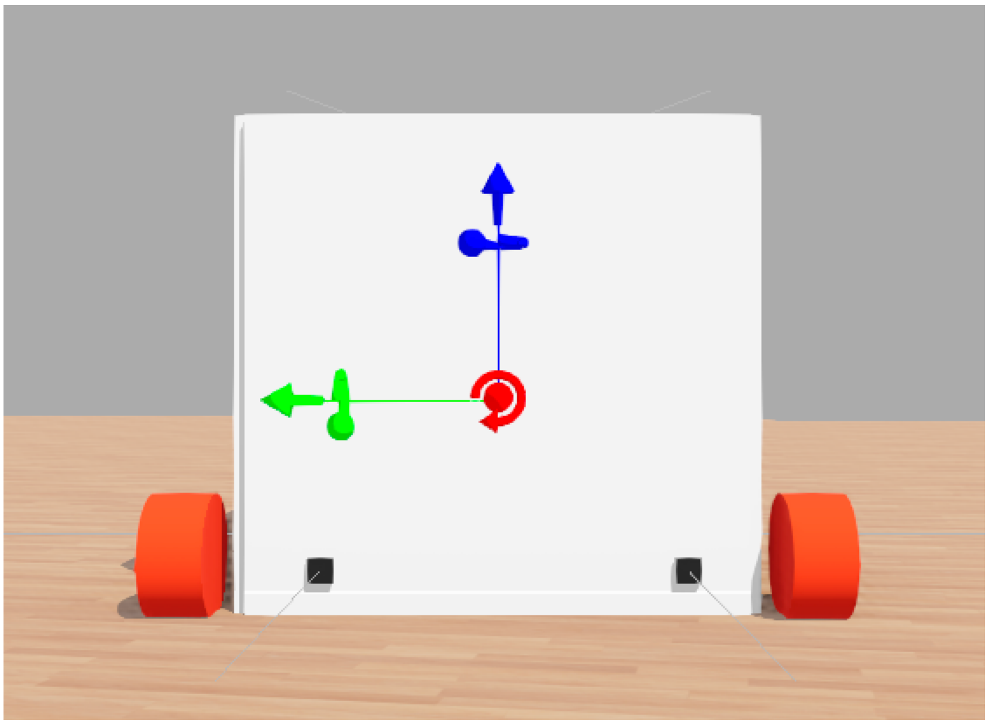
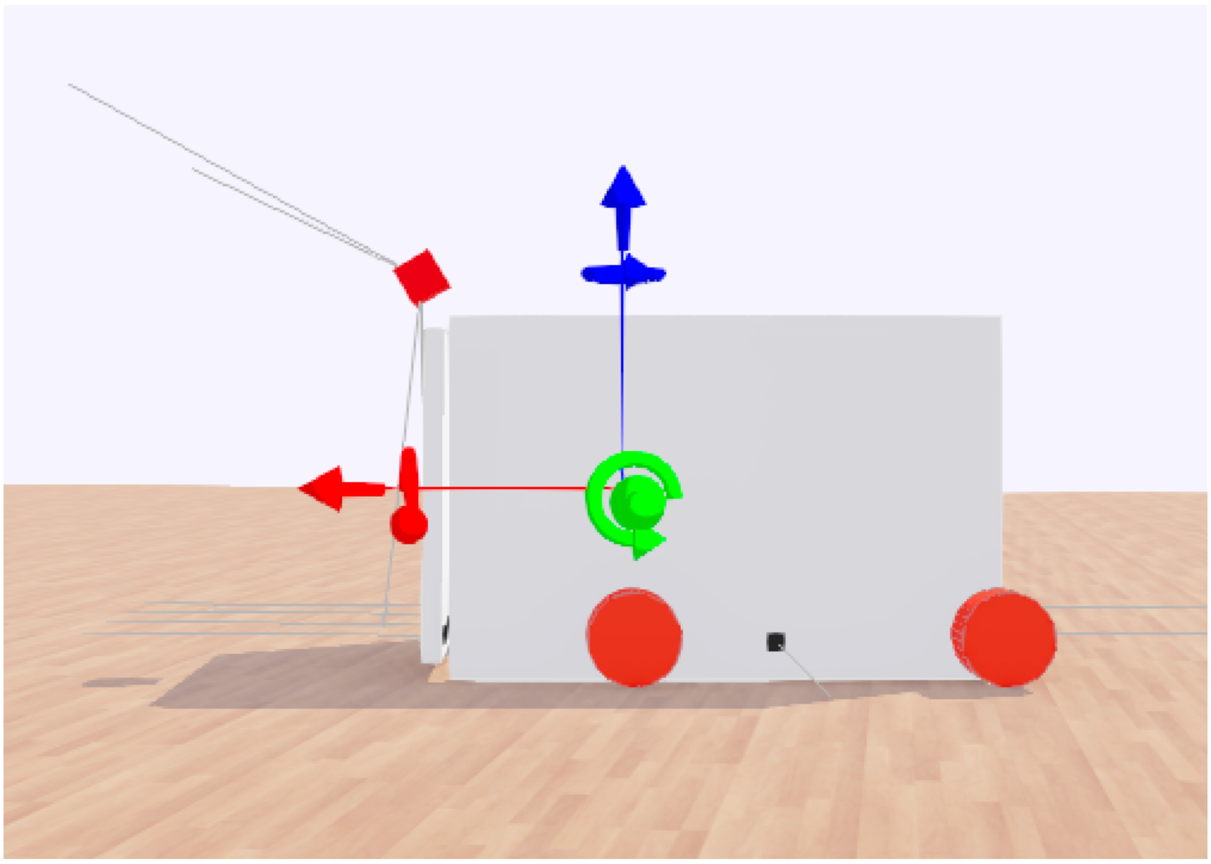


Figure 4.2: Litter (Can Bottle) – Top View

4.2 3D MODEL OF ROBOT

The final result of the 3D modelling of the robot is presented in the figures following





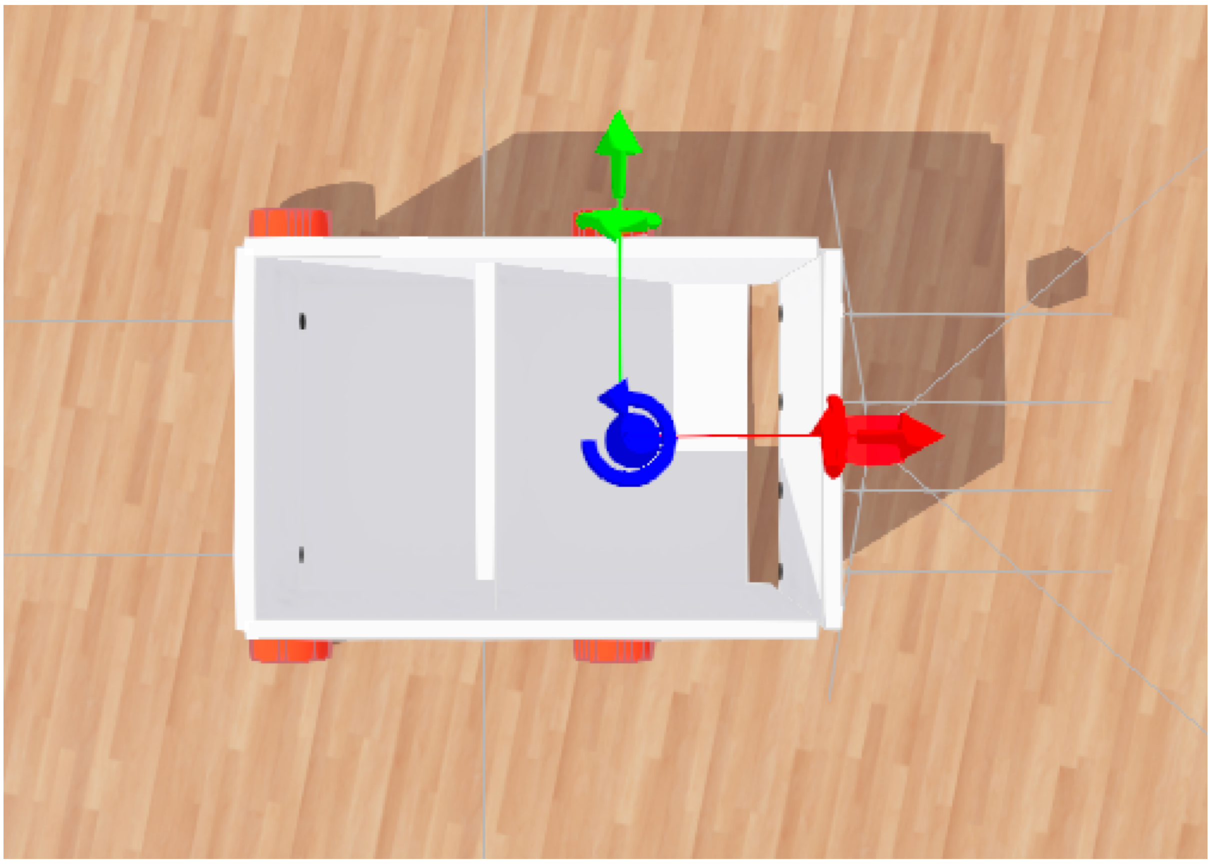


Figure 4.3: Robot 3D Model Top, Side and Front Views

4.3 HAAR'S CASCADE CLASSIFIER TRAINING

The table below gives the result of testing each classifier trained for different stages

Table 4.1: Summary of Haar's Cascade Re

Number of Stages	**Bottles Detected**	**Actual Bottles**	**Number of False Positives**	**Number of True Positives**
10	540	200	343	197
11	432	200	237	195
12	329	200	139	190
13	299	200	110	189
14	203	200	14	189
15	129	200	9	120
16	119	200	4	115
17	98	200	2	96
18	75	200	0	75
19	50	200	0	50

It was observed that the lesser the number of levels we trained for the classifier detected more bottles but with more false positives which was indicative of under-training. While the more the number of levels we trained for, the classifier detected less bottles (more misses) which was indicative of over-training. A balance was found when the classifier was trained for 14 levels which detected 203 bottles (189 true positive and 14 false positives) bringing the efficiency of our final Haar's Cascade classifier to 93.10%.

4.4 ROBOT'S CONTROL SIMULATION WITH WEBOTS

To test the robot's control algorithm, we used different scenarios with our test environment. We randomly placed several test bottles around the test arena for the robot to retrieve.

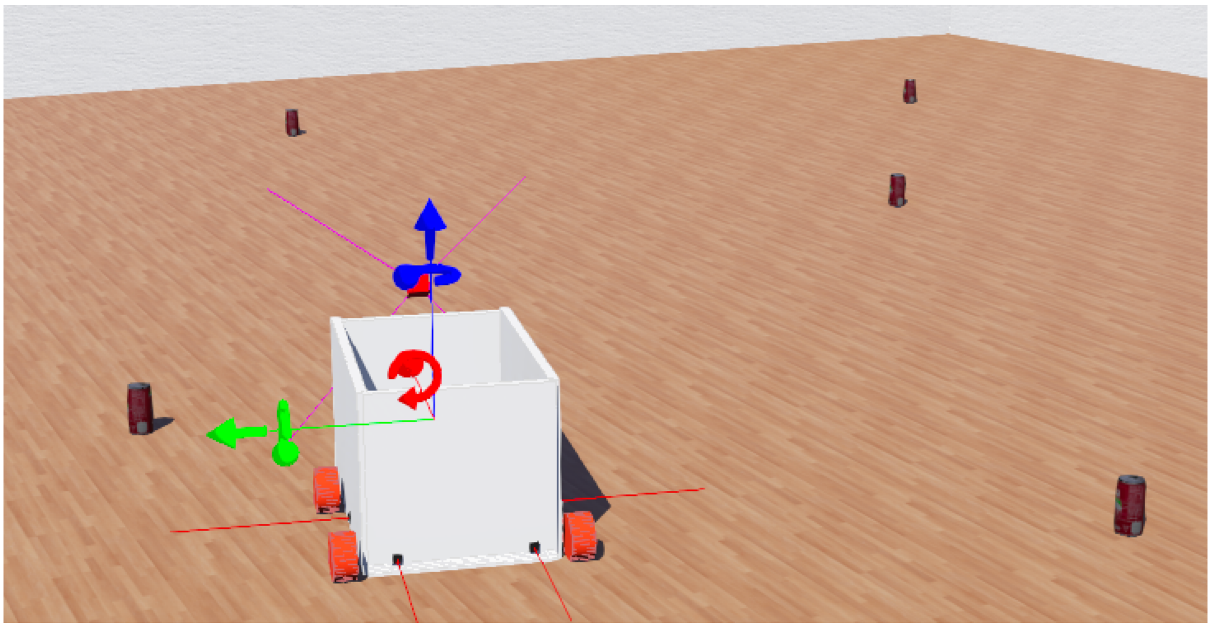


Figure 4.4: Robot simulation in webots

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1. CONCLUSION

We were able to design and develop a 3D model of our robot and test environment using the Webots simulation software. With these we proceeded to train a Haars-Cascade classifier to detect the test bottles in the 3D environment obtaining an accuracy of 93.10%. The robot control was developed and implemented using the Python programming language and simulated as well using Webots.

5.2. LIMITATIONS

The project only covered detecting one type of bottle and also didn't cover plastic bottle retrieval on any other surface other than a flat and smooth surface of the ground of LT1.

5.3. RECOMMENDATIONS

Based on the work carried out the following recommendations are made:

1. For future work, the programming language C++ can be used for programming the machine learning model.
2. Instead of testing via simulation, the robot can be constructed with reliable software.
3. The classifier can be trained to detect more than one type of plastic bottle.
4. Future work can also take into account plastic waste retrieval on a rocky terrain or a grass field for better efficiency.

REFERENCES

1. Geyer, R., Jambeck, J. R. & Law, K. L. (2017). Production, use, and fate of all plastics ever made.
2. Haider, J., & Azhar, A. (2019). Plastic waste management in developing countries: A review. Journal of Cleaner Production, 208, 817-829.
3. Lee, J., & Park, J. (2017). Advances in recycling and management of plastic solid waste.
4. Gasser, L. (2017). The robot in the garden: Telerobotics and telepistemology in the age of the Internet. MIT Press.
5. Shapiro, L.G., & Stockman, G.C. (2001). Computer Vision. Prentice Hall, Upper Saddle River, NJ.
6. G. R.Girshick, et al. (2014) "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In Proceedings of the 2014 Conference on Computer Vision and Pattern Recognition.
7. J. Deng, et al. (2009) "ImageNet: A Large-Scale Hierarchical Image Database". In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition.
8. Garg, K. (2021). Computer Vision: Algorithms, Techniques, and Applications. CRC Press.
9. Szeliski, R. (2010). Computer vision: algorithms and applications. Springer.
10. Duda, R. O., & Hart, P. E. (1973). Pattern classification and scene analysis. New York: Wiley.
11. Bishop, C. (2006). Pattern Recognition and Machine Learning (1st ed.). Springer.
12. Dhillon, I.S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 269-274.

13. Bellman, R., Zadeh, L.A., & Mokhatab, R. (1966). "Abstraction and generalization in fuzzy systems". Mathematical Biosciences, 1(1), pp. 75-94.
14. McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, 5(4), 115-133.
15. Holland, J.H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press, Ann Arbor.
16. Viola, P., & Jones, M. J. (2004). Rapid object detection using a boosted cascade of simple features. Computer vision and pattern recognition, 1(5), I-511.
17. Cherkassky, V. and Mulier, F. (2007). Learning from Data: Concepts, Theory, and Methods. Wiley.
18. Gonzalez, R. C., & Woods, R. E. (2017). Digital image processing (4th ed.). Pearson.
19. Shao, L., Liu, J., Wang, F., & Lai, K. K. (2017). Deep learning in medical image analysis. Annual Review of Biomedical Engineering, 19, 221-248.
20. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
21. Fernández Villán, A. (2019). Mastering OpenCV 4 with Python: A practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3. Packt Publishing Ltd.
22. Joshi, P. (2018). OpenCV 3.x with Python By Example: Enhance your understanding of Computer Vision and image processing by developing real-world projects in OpenCV 3.x. Packt Publishing Ltd.

23. Ali, K., et al. (2018). Real-Time Waste Detection and Classification in Smart Garbage Bins using Deep Learning. In 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 2512-2517). IEEE.
24. Al-Otaibi, M., et al. (2018). Autonomous Waste Sorting Robot: Design and Implementation. In 2018 5th International Conference on Control, Engineering & Information Technology (CEIT) (pp. 1-6). IEEE.
25. Zhang, J., et al. (2019). Robust Waste Detection and Classification using Convolutional Neural Networks. In 2019 International Conference on Electrical and Computer Engineering (ICECE) (pp. 305-310). IEEE.
26. Airobotic Systems. (n.d.). USARSim. Retrieved from https://airobotic.com/usarsim/
27. Eggers, J., Braun, A., & Worn, H. (2008). Simulation-based testing of a multi-robot system for urban search and rescue. In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 2693-2698).
28. Jain, A., Hall, E. L., & Stentz, A. (2005). Simulation-based comparison of multi-robot coordination algorithms for urban search and rescue. In 2005 IEEE International Conference on Robotics and Automation (pp. 1769-1776).
29. Urban Robotics. (n.d.). USARSim. Retrieved from https://urbanrobotics.com/usarsim/
30. Bandyopadhyay, S., Tyagi, A., & Kumar, V. (2019). Development of Autonomous Underwater Vehicle Using Gazebo Simulator. In 2019 Fourth International Conference on Control, Automation and Robotics (ICCAR) (pp. 1-6). IEEE.
31. Ricardo, M., Félix, J. G., & Bastos, R. (2018). Gazebo simulator for aerial robots: Development and assessment. Robotics and Autonomous Systems, 100, 1-12.
32. Wang, S., Zhang, L., & Chen, S. (2019). Development and evaluation of the autopilot system for a UAV in Gazebo simulation. In 2019 9th International Conference on Robotics and Artificial Intelligence (ICRAI) (pp. 1-6). IEEE.

33. JMonkey Engine. (2023). JMonkey Engine: 3D Game Engine. [online] Available at: https://jmonkeyengine.org/ [Accessed 2 Feb. 2023].
34. Liao, Y., Lin, C., & Chen, C. (2018). Study of a quadruped robot walking using Webots. In Proceedings of the 2018 International Conference on Robotics and Automation (ICRA) (pp. 1-6). IEEE.
35. Liu, W., Liao, Y., Lin, C., & Chen, C. (2018). An improved grasping strategy for a humanoid robot hand using Webots. In Proceedings of the 2018 International Conference on Robotics and Automation (ICRA) (pp. 1-6). IEEE.
36. Lin, C., Liao, Y., & Chen, C. (2019). A study of object recognition using Webots. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA) (pp. 1-6). IEEE.
37. AHR international. (n.d.). *Electric motor bearings and their application*. AHR International. Retrieved December 12, 2022, from https://www.ahrinternational.com/bearings_electric_motor_applications.html
38. Britannica. (n.d.). *Robotics | Definition, Applications, & Facts | Britannica*. Encyclopedia Britannica. Retrieved December 5, 2022, from https://www.britannica.com/technology/robotics
39. Burnett, R. (2020, March 24). *Understanding How Ultrasonic Sensors Work.* MaxBotix Inc. Retrieved December 18, 2022, from https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm
40. Components 101. (2019, April 18). *IRFZ44N MOSFET Pinout, Features, Equivalents & Datasheet*. Components101. Retrieved December 17, 2022, from https://components101.com/mosfets/irfz44n-datasheet-pinout-features

41. Dey, A. K. (n.d.). *Belt Drive: Types, Material, Applications, Advantages, Disadvantages [PDF] – Learn Mechanical*. Learn Mechanical. Retrieved December 18, 2022, from https://learnmechanical.com/belt-drive/
42. Ejimofor, M. I., Aniagor, C. O., Oba, S. N., Menkiti, M. C., & Ugonabo, V. I. (2022). *Artificial intelligence in the reduction and management of land pollution*. sciencedirect.com. Retrieved December 6, 2022, from https://www.sciencedirect.com/science/article/pii/B9780323855976000094
43. Klearstack. (2021, August 17). *Image Processing and Machine Learning: Redefining the Future*. KlearStack. Retrieved December 10, 2022, from https://www.klearstack.com/image-processing-and-machine-learning/
44. Tripathi, M. (2021, June 21). *Image Processing using CNN | Beginner's Guide to Image Processing*. Analytics Vidhya. Retrieved December 9, 2022, from https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/
45. The World Bank. (2019). What a Waste: A Global Review of Solid Waste Management. *International Journal For Environmental Research And Public Health*, (1060), 15-30. Pub Med Central. Retrieved November 28, 2022, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6466021/#B4-ijerph-16-01060

APPENDIX

APPENDIX A: ROBOT CONTROLLER CODE

```
from controller import Robot, Motor, Camera

from image_processing import ImageProcessor

from motor_control import MotorControl

from sweeper_control import SweeperControl

from slider_control import SliderControl

from distance_sensor_util import DistanceSensorUtil, ObstacleMovementState


robot = Robot()

timestep = int(robot.getBasicTimeStep())


imageProcessor = ImageProcessor(robot=robot)

motorControl = MotorControl(robot=robot)

sweepControl = SweeperControl(robot=robot)

sliderControl = SliderControl(robot=robot)

distanceSensor = DistanceSensorUtil(robot=robot)


def wait(seconds: int):

    duration = seconds * 1000

    steps = int(duration / timestep)


    for i in range(steps):

        robot.step(timestep)
```

```
while robot.step(timestep) != -1:
    # imageProcessor.step()

    # continue
    # check for obstacle state
    obstacleStats = distanceSensor.getObstacleStatus()

    if obstacleStats.frontLeft or obstacleStats.frontRight:
        if obstacleStats.frontLeft and obstacleStats.frontRight:
            motorControl.moveBackwardStep()
        if obstacleStats.right == False:
            print('obstacle: turn right')
            motorControl.turnRightStep()
            continue
        if obstacleStats.left == False:
            print('obstacle: turn left')
            motorControl.turnLeftStep()
            continue
        if obstacleStats.right == True and obstacleStats.left == True:
            if not obstacleStats.backLeft and not obstacleStats.backRight:
                print('obstacle: move back')
                motorControl.moveBackwardStep()
                continue
            else:
```

```
            print('DEADLOCK')
            continue

    # else check if there is bottle
    imageProcessor.step()
    bottleX = imageProcessor.bottleX
    bottleY = imageProcessor.bottleY

    if bottleX:

        # if bottle check if it is in the center
        new_var = (bottleX < 0.45) or (bottleX > 0.57)

        if new_var:
            print('bottle not in line of sight')
            # if not align
            if (bottleX < 0.5):
                print('aligning left')
                motorControl.turnLeftStep()
            else:
                print('aligning right')
                motorControl.turnRightStep()
            continue

        # if aligned move forward step
        else:
            print('moving to bottle')
            motorControl.moveForwardStep()

            # check if bottle is within range
            isBottleInRange = bottleY >= 0.87

            if isBottleInRange:
                wait(2)
                print('bottle in range')
                # if so deploy the sweeper and move to bin
                sweepControl.sweep()
                wait(2)
                sliderControl.moveToBin()
            else:
                print('bottle not in range')
            continue

    else:
        print('is roaming')
        motorControl.moveForwardStep()

    pass
```

APPENDIX B: MOTOR CONTROLLER CODE

```
from math import pi


_SPEED: int = 5

INF = float('inf')


WHEEL_RADIUS = 0.05


def calculateRadianDistanceFromLinearDistace(distance: float):

    return distance/WHEEL_RADIUS


class MotorControl:

    r1 = None

    r2 = None

    l1 = None

    l2 = None

    robot = None


    def __init__(self, robot) -> None:

        self.r1 = robot.getDevice('wheel_right1')

        self.r2 = robot.getDevice('wheel_right2')

        self.l1 = robot.getDevice('wheel_left1')

        self.l2 = robot.getDevice('wheel_left2')

        self.robot = robot

        pass
```

```
def _rightMove(self, position: float, speed: int = _SPEED, ):
    self.r1.setPosition(position)
    self.r1.setVelocity(speed)
    self.r2.setPosition(position)
    self.r2.setVelocity(speed)

def _leftMove(self, position: float, speed: int = _SPEED):
    self.l1.setPosition(position)
    self.l1.setVelocity(speed)
    self.l2.setPosition(position)
    self.l2.setVelocity(speed)

def _move(self, position: float, speed: int = _SPEED):
    self._leftMove(position=position, speed=speed)
    self._rightMove(position=position, speed=speed)

def _turn(self, right: int, left: int):
    '''
    right or left should take these values -1, 1 or 0
    '''
    radianDistance = calculateRadianDistanceFromLinearDistace(0.05)
    duration = (radianDistance/_SPEED) * 1000
    time_step = int(self.robot.getBasicTimeStep())
    steps = int(duration / time_step)


    for i in range(steps):
        self._rightMove(position=INF, speed=_SPEED*right)
        self._leftMove(position=INF, speed=_SPEED*left)
        self.robot.step(time_step)
    self.stop()




def stop(self):
    self._move(position=INF, speed=0)


def moveForwardStep(self):
    self._turn(right=1,left=1)


def moveBackwardStep(self):
    self._turn(right=-1,left=-1)


def turnLeftStep(self):


    self._turn(right=1,left=0)


def turnRightStep(self):
    self._turn(right=0,left=1)
```

APPENDIX C: IMAGE PROCESSING CODE

```
from controller import Display

import numpy as np

import cv2


class ImageProcessor:


    bottleX = None

    bottleY = None


    def __init__(self, robot):

        time_step = int(robot.getBasicTimeStep())

        self.robot = robot

        self.camera = robot.getDevice('camera')

        self.camera.enable(time_step)

        self.camHeight = self.camera.getHeight()

        self.camWidth = self.camera.getWidth()

        if self.camera.hasRecognition():

            self.camera.recognitionEnable(time_step)

        self.display = robot.getDevice('camera_display')


    def step(self):

        cameraData = self.camera.getImage()
    self.previewImage = np.frombuffer(
        cameraData, np.uint8).reshape((self.camWidth, self.camHeight, 4))
    self.detectBottles()

    if cameraData:
        ir = self.display.imageNew(
            self.previewImage.tobytes(), Display.BGRA, self.camWidth, self.camHeight)
        self.display.imagePaste(ir, 0, 0, False)
        self.display.imageDelete(ir)

def detectBottles(self):
    objects: list = self.camera.getRecognitionObjects()
    if len(objects) > 0:
        bottle = objects[0]
        y_check=None
        for object in objects:
            if not y_check:
                _,y_check=list(bottle.getPositionOnImage())
                bottle=object
                continue
            _,y2= list(bottle.getPositionOnImage())
            if y2>y_check:
                bottle=object
        w, h = list(bottle.getSizeOnImage())
        x, y = list(bottle.getPositionOnImage())
        norm_x = (x/self.camWidth)
        norm_y = (y/self.camHeight)
        self.bottleX, self.bottleY = norm_x, norm_y
        cv2.rectangle(self.previewImage, (x-w, y-h),
                      (x+w, y+h), (255, 255, 255), 1)
    else:
        self.bottleX = None
        self.bottleY = None
    print(self.bottleX,self.bottleY)
```

APPENDIX D: DISTANCE SENSOR CONTROLLER CODE

```
from enums import ObstacleMovementState
class ObstacleStatus():

    def __init__(
        self, left: bool, right: bool, frontRight: bool, frontLeft: bool, backRight: bool, backLeft:
bool,
    ):
        self.right: bool = right
        self.left: bool = left
        self.frontRight: bool = frontRight
        self.frontLeft: bool = frontLeft
        self.backRight: bool = backRight
        self.backLeft: bool = backLeft


class DistanceSensorUtil:

    def __init__(self, robot):
        time_step = int(robot.getBasicTimeStep())

        self.sensorBottle = robot.getDevice('distance_sensor_center')
        self.sensorBottle.enable(time_step)

        self.sensorBottle2 = robot.getDevice('distance_sensor_center_2')
        self.sensorBottle2.enable(time_step)

        self.sensorFrontWallLeft = robot.getDevice('distance_sensor_left')
        self.sensorFrontWallLeft.enable(time_step)

        self.sensorFrontWallRight = robot.getDevice('distance_sensor_right')
        self.sensorFrontWallRight.enable(time_step)

        self.sensorLeftWall = robot.getDevice('distance_sensor_left_wall')
        self.sensorLeftWall.enable(time_step)
        self.sensorRightWall = robot.getDevice('distance_sensor_right_wall')
        self.sensorRightWall.enable(time_step)

        self.sensorBackWallRight = robot.getDevice(
            'distance_sensor_back_wall_right')
        self.sensorBackWallRight.enable(time_step)

        self.sensorBackWallLeft = robot.getDevice(
            'distance_sensor_back_wall_left')
        self.sensorBackWallLeft.enable(time_step)

        robot.step(time_step)

    def getObstacleStatus(self) -> ObstacleStatus:
        frontLeft = self.sensorFrontWallLeft.getValue()
        frontRight = self.sensorFrontWallRight.getValue()

        backLeft = self.sensorBackWallRight.getValue()
        backRight = self.sensorBackWallLeft.getValue()

        left = self.sensorLeftWall.getValue()
        right = self.sensorRightWall.getValue()

        return ObstacleStatus(
            frontRight=frontRight < 100,
            frontLeft=frontLeft < 100,
            backRight=backRight < 100,
            backLeft=backLeft < 100,
            left=left < 100,
            right=right < 100,
        )
```

APPENDIX D: ENUMS

```
import enum


class StateCodes(enum.Enum):
    STATE_SEARCHING='STATE_SEARCHING'
    STATE_FETCHING_BOTTLE='STATE_FETCHING_BOTTLE'
    STATE_MOVING_TO_BIN= 'STATE_MOVING_TO_BIN'


class ReturnCodes(enum.Enum):
    OK='OK'
    FAIL='FAIL'
    REPEAT='REPEAT'


class ObstacleMovementState(enum.Enum):
    FORWARD=1
    BACKWARD=2
    LEFT=3
    RIGHT=4
    WAIT=5
```

APPENDIX E: SLIDER CONTROLLER

```
from math import pi
_SLIDE_SPEED: int = 0.05
_ANGLE_SPEED: int = 0.5
INF = float('inf')


class SliderControl:
    slider = None
    rotation_motor= None
    rotation_position_sensor= None
    slider_position_sensor= None


    def __init__(self, robot) -> None:
        time_step = int(robot.getBasicTimeStep())
        self.rotation_motor = robot.getDevice('slab_rotational_motor')
        self.rotation_position_sensor = robot.getDevice('slab_rotational_position')
        self.slider = robot.getDevice('slab_linear_motor')
        self.slider_position_sensor = robot.getDevice('slider_position')
        self.slider_position_sensor.enable(time_step)
        self.rotation_position_sensor.enable(time_step)
        self.robot = robot
        # the next line is necessary tto initialize the position_sensor value
        self.robot.step(time_step)
        pass


    def _turn(self, position: float, speed: int ):
        self.rotation_motor.setPosition(position)
        self.rotation_motor.setVelocity(speed)


    def _slide(self, position: float, speed: int, ):
        self.slider.setPosition(position)
        self.slider.setVelocity(speed)


    def moveToBin(self):
        time_step = int(self.robot.getBasicTimeStep())
        slideDistance = 0.5  # to turn 2pi we set this higher than like 3pi
        duration = (slideDistance/_SLIDE_SPEED) * 1000
        slideSteps = int(duration / time_step)

        angleDistance = pi  # to turn 2pi we set this higher than like 3pi
        duration = (angleDistance/_ANGLE_SPEED) * 1000
        angleSteps = int(duration / time_step)

        # step 1: slide up
        pos = self.slider_position_sensor.getValue()
        if pos< 0.3:
```

```
    pos=0.3
    for i in range(slideSteps):
        self._slide(position= pos, speed=_SLIDE_SPEED)
        self.robot.step(time_step)

# step 2: rotate clockwise
pos2= self.rotation_position_sensor.getValue()
if pos2 > -(pi/2):
    pos2=-(pi/2)
    for i in range(angleSteps):
        self._turn(position= pos2, speed=_ANGLE_SPEED)
        self.robot.step(time_step)

# step 3: rotate counter clockwise
pos3= self.rotation_position_sensor.getValue()
if pos3 <= -(pi/2):
    pos3=0
    for i in range(int(angleSteps/2)):
        self._turn(position= pos3, speed=_ANGLE_SPEED)
        self.robot.step(time_step)

# step 4: slide down
pos4 = self.slider_position_sensor.getValue()
if pos4 >= 0.3:
    pos4=0
```

```
        for i in range(int(slideSteps/2)):
            self._slide(position= pos4, speed=_SLIDE_SPEED)
            self.robot.step(time_step)
```

APPENDIX F: SWEEPER CONTROLLER

```
from math import pi
_SPEED: int = 2.5
INF = float('inf')


class SweeperControl:
    sweeper = None
    position_sensor= None


    def __init__(self, robot) -> None:
        time_step = int(robot.getBasicTimeStep())
        self.sweeper = robot.getDevice('sweeper_motor')
        self.position_sensor = robot.getDevice('sweeper_position_sensor')
        self.position_sensor.enable(time_step)
        self.robot = robot
        # the next line is necessary tto initialize the position_sensor value
        self.robot.step(time_step)
        pass


    def _turn(self, position: float, speed: int = _SPEED, ):
        self.sweeper.setPosition(position)
        self.sweeper.setVelocity(speed)

    def sweep(self):

        radianDistance = 3 * pi  # to turn 2pi we set this higher than like 3pi
        duration = (radianDistance/_SPEED) * 1000
        time_step = int(self.robot.getBasicTimeStep())

        steps = int(duration / time_step)
        pos = self.position_sensor.getValue() + (2 * pi)

        for i in range(steps):
            self._turn(position= pos, speed=_SPEED)
            self.robot.step(time_step)
```