Human-Following Robot in a Virtual Environment

Author: B924007

Loughborough University

22COP328: MSc Data Science

Supervisor: Dr Haibin Cai

Date: 2023

Abstract

Human-following robots are a core concept in understanding the field of robotics, particularly in showing the potential of how robots operate and interact with humans. The logic and concepts gained from the design of a human-following robot can be built upon and applied into a range of HRC (Human Robot Collaboration) projects. These applications could include human-helper robots that aid in hospitals, tour guide robots that can show humans around places of interest like a museum, virtual robots that can interact in games and websites etc. While most research in robotics focuses on real-life application, the importance of virtual robot development and testing is undervalued as it is an extremely safe and cost-effective manner in which robots can be developed prior to real world deployment.

The focus of this project is to design a virtual human-following robot that can identify and follow a human within different simulation environments developed using Gazebo, by making use of ROS and relevant python packages. Firstly, a virtual environment is made so that PyTorch and ROS can work in collaboration. Several Gazebo scenarios are also created which contain a human model that walks different navigation routes with the presence of obstacles. Using this human model, an image set is obtained and augmented which can then be used to train a deep learning model.

A virtual robot (TurtleBot3 Waffle Pi) is created and trained on a real-time object detection algorithm (YOLOv3) that can identify a human. The optimal model is able to reach a mAP of 60% which is sufficient for the context of the project. This is discussed as the trained YOLOv3 model is then tested using the developed human detection algorithm. This algorithm creates a green bounding box around the detect human and shows that the trained YOLOv3 model has been successfully applied to the robot's camera and is able to correctly identify the human in Gazebo from ranges of 0.5-10m.

Finally, a human tracking algorithm is developed which makes use of the gathered data in the human detection algorithm and create a control command that instructs the robot to follow the human based on the collected data. The tracking algorithm was successful in following the human in a straight line; however, it performed poorly when a more sophisticated algorithm with angular velocity implementations was used. An investigation was carried out into what caused these limitations so that the algorithm can be built on in the future. In terms of accuracy, the robot performs quite well as it is able to achieve a 60% human detection accuracy. A basic safety logic is also implemented which ensures that the robot maintains a safe distance from the human to prevent any collisions occurring.

This report provides a detailed account into the development of a virtual human-following robot from scratch, as well as testing the performance of the robot and carrying out several investigations to drive future suggestions both project-specific and industrially specific so that ideas and concepts from this report can be built upon further and applied into the wider field of robotics.

Contents

List of Figures

1 Introduction

1.1 Project Context & Background

Robotics has been a technological sector rapidly increasing in popularity over the last few decades as it gains ever-growing potential with a range of applications suitable to every industry. Robotics is a branch of engineering that focuses on creating intelligent machines that can aid humans in a variety of ways, while abiding to laws such as 'Asimov's Three Laws of Robotics' developed in 1940, essentially guiding the behaviour and design of autonomous robots in a safe and effective manner (Anderson, 2008). While the first programmable robot 'Unimate' was created to move scalding metal pieces in 1961 (Singh et al., 2013), today industrial robots are used for countless day-to-day tasks from simply carrying objects mindlessly, to efficiently coexisting with humans to carry our highly technical tasks such as surgeries. In recent years, robotics has emerged as a significant field of research and development due to its extraordinary potential to revolutionize several industries such as manufacturing, healthcare, and transportation (Yasar & Hanna, 2023). Ultimately, the application for robotics could and will be applied to every industry as it aims to increase efficiency of existing technology and make life easier by paving the way for developing autonomous systems that can interact with humans seamlessly.

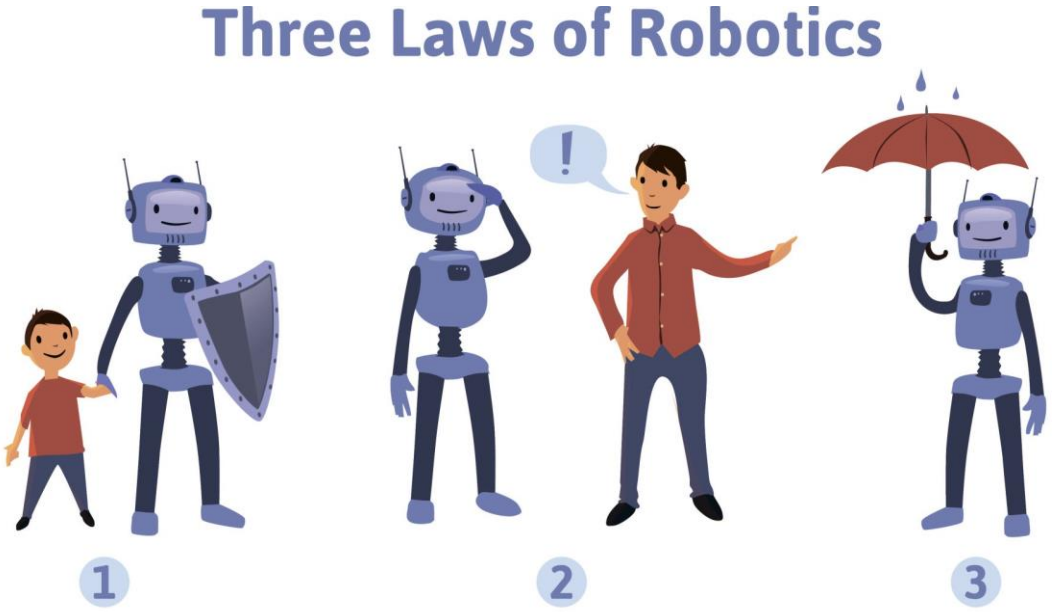


Figure 1 - Isaac Asimov's 3 Fundamental Laws of Robotics (Vachnadze, 2021)

Autonomous robots are of particular interest in the developing field of robotics due to their ability to act without recourse to human control. While old age robots are rigid operating machinery that carry out basic tasks blindly, emerging artificial intelligence looks into the implementation of smart robots that are able to operate in dynamic and unstructured environments as they interact seamlessly with humans (Choi et al., 2021). This is made possible due to the design of these robots with several sensors, as well as comprehensive control algorithms that allow the robot to make informed decisions based on the information it is receiving from sensors, as well as the function it has been designed for (Builtin, 2023). These new robots aim to be flexible and reconfigurable as they adapt to different environments as new robots are being developed to assume roles such as self-driving cars, tutoring, and remotely performing surgeries.

Despite the majority of robotics research being on physical robots that operate in real life, simulating robot behaviours virtually has been deemed a necessary research tool required to

model, visualise, plan, and develop these robots prior to physical deployment. One particularly interesting application of autonomous robots is the creation of a robot that can follow a human (Choi et al., 2021). Hence, the focus of this study will be in developing and analysing the performance of a human-following robot in a virtual environment. While this may seem like a relatively simple concept, the theory and design of this robot serves as a basis for the development and concepts used to create more complex robots that carry out technical tasks. Human-following robots provide a range of useful applications such as assisting people with disabilities, aiding in public places, or acting as a tour guide for tourists. Additionally, human-following robots have further use within virtual environments such as gaming as there is growing interest in virtual reality, as well as Web 3.0 and the Metaverse which aims to combine both the digital and physical world to create more immersive experiences in business, gaming, and daily activities.

Virtual environments and simulations such as Gazebo provide a range of advantages in robot development as it provides a cost-effective & safe method to test and develop robot algorithms prior to real world deployment where they will operate safely with improved performance. Simulations have the potential to push the field of robotics further as it can eliminate certain barriers that may have limited the development of robots (Žlajpah, 2008). Although physical testing of robots is inevitably mandatory, the design process can be accelerated through testing. The opportunities and limitations of designing robots in a virtual environment will be discussed thoroughly throughout the report.

1.2 Aims & Objectives

The objective of this research project is to design a human-following robot using ROS, within a virtual environment (Gazebo), that will follow a human in a simulated world. The robot and human will be tested in a several environments to see how the robot will respond to increasingly difficult navigation routes. While some scenarios may not give positive results, the results can still be analysed to investigate the limitations of the robot and provide further suggestions to improve the robot's performance.

The above task will be done by making use of the existing TurtleBot3 Waffle Pi robot due to its structure and function which will be discussed thoroughly, consisting of a camera and lidar sensor (Kononov, 2021). Gazebo is a widely used simulation environment that will work in collaboration with ROS to develop and test robotic algorithms without the need for physical hardware. Additionally, algorithms such as deep-learning, human detection, and human-following will be created using python codes, making use of relevant packages like OpenCV and PyTorch.

1.3 Scope and Limitations

This task presents a range of challenges that will be addressed. Firstly, a sophisticated code that properly communicates between sensors, the robot control system, and the simulation environment is required to ensure smooth operation between different nodes that are all working together. Secondly, creating an accurate algorithm for human detection and tracking can be complex in a dynamic environment (Tsarouchi et al., 2016). The difficulty of this task comes with the ability for the robot to continuously be able to keep track of the human and efficiently follow him/her in potentially challenging scenarios where there may be changes in speed or the presence of obstacles. A key challenge for the robot will be its ability to keep track of and correctly identify the human from a range of different angles and distances. Due to this, incorporating a machine learning algorithm where the robot is trained to identify a human based on a custom dataset will be necessary in order to improve the robot's ability to follow the human. Similar concerns also arise when developing a robust obstacle avoidance algorithm as this is crucial for the robot's safety and to ensure smooth navigation.

A successful project will aim to overcome these challenges and create a sophisticated human-following robot that can autonomously track a human, while simultaneously avoiding obstacles and navigating through the environment. In the case of an unsuccessful robot that cannot efficiently follow the human, a study will be carried out into what has caused these limitations and how they could be overcome.

2 Literature Review

The idea of human-following robots provide interesting principles and concepts that can be translated into more complex applications. There are several studies carried out that specifically look into the creation, application, and optimisation of human-following robots both virtually and in real life. While both virtual and real-life robots follow similar concepts and ideas, the methodologies differ as the virtual environment is typically tested in the early stages of robot development to ensure safe production.

The literature review will look into the development of a human-following robot from the ground up. Firstly, a study will be carried out on ROS and robot functionality in general as this is the core theory and concept of how the robot will be operate. Secondly, it is worth carrying out a study on the Gazebo simulation environment, specifically how it works and interacts with ROS. After this, an overview of human-following robots will be carried out which show the principles and applications of the robots, including techniques and algorithms used for human detection and tracking, reviewing different sensors and perception features the robot could have, as well as the challenges and considerations involved in the process. This will lead into existing seminal work on creating human-following robots with ROS and Gazebo to highlight the progress of this technology, as well as the challenges and opportunities present. After the main body of the research, further discussions will be carried out regarding further potential of HRC as it is a growing industry with immeasurable potential.

2.1 Overview of ROS and Robot Functionality

Robot Operating System (ROS) is a set of software libraries, and tools that help build robot applications. It is an open-source set of software frameworks that allow for efficient application of robotics packages (Constructism, 2019). Since first being developed in 2007 as a personal project of Keenan Wyrobek and Eric Berger at Stanford University, it has quickly become the standard for programming in robotics due to its comprehensive framework that allows it to abstract hardware from the software, allowing users to create robots without having to deal with hardware. Keenan Wyrobek and Eric Berger developed ROS with the aim of solving the most common problems within robotics at the time (Constructism, 2022). Being an open-source project, ROS allows users to choose and configure the tools and libraries as they please to fit their robot's requirements. Since the birth of ROS, several developments and new versions have been released as the framework is constantly evolving to keep up with the advancements in robotics. ROS offers a flexible architecture that allows for code reusability, simplification of the development process, and open collaboration (Ed, 2023).

Quigley et al. (2009) provides an overview of ROS as an open-source system designed to simplify the development of robot software. They discuss the motivation behind the creation of ROS as it aimed to target a specific set of challenges and philosophical such as: peer-to-peer, tools-based, multi-lingual, thin, free, & open-source. ROS provides a collection of software libraries and tools that provide various functionalities such as drivers, communication infrastructure, data visualization, and package management. The paper also highlights benefits of ROS such as the flexibility and community support of the project, both of which are key to its success. Despite providing a comprehensive introduction to ROS, Quigley et al. (2009) did not dive deeper into specific applications of ROS as it focused more on the technical aspects.

ROS processes can be represented as nodes in a graph structure, connected by topics. Nodes pass through topics and make service calls to other nodes, allowing interactions between different programs. The basic ROS tools can be used to understand how it allows programs to communicate:

Actions: Actions allow the user to give non time-sensitive directions that can carry out a specific task,

Topics: Topics are a medium that can relay information among nodes in order to transmit data.

Services: Services create a channel between nodes so that developers can request actions from the robots.

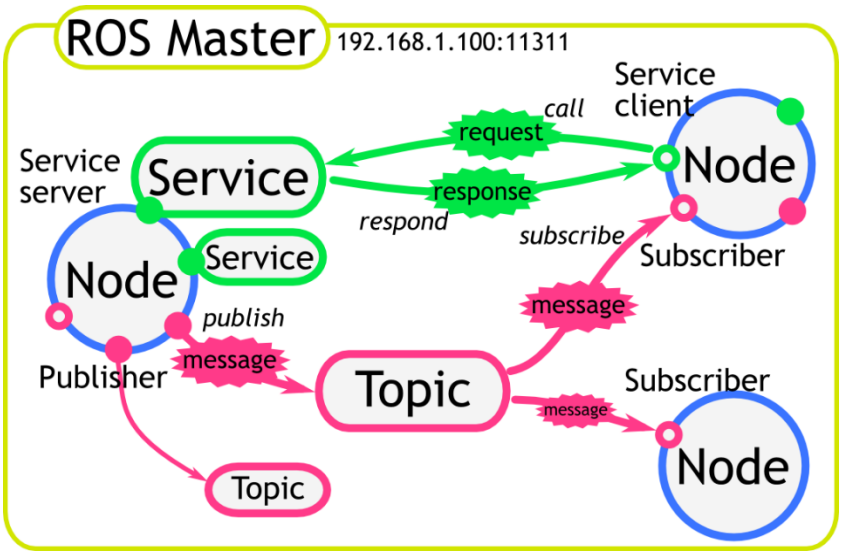


Figure 2 - ROS interactions (Akihiko, 2018)

Maruyama et al. (2016) explores the performance of ROS2 as it highlights all its key features and capabilities, as well as the advancements made from the original ROS. The authors observed that ROS2 introduced several improvements such as reduced communication as it highlights several tests which could not be carried out on ROS initially. ROS1 was not suitable for real-time embedded systems and was only operable on a few OSs. The paper aimed to explore not only the advantages of ROS2, but also the limitations and challenges ROS2 could face. However, the paper does not cover performance considerations for ROS on a large-scale to see how well ROS2 performs as the system and communication load increases.

Further discussion of ROS2 is carried out by Eros et al. (2019) as they discuss the development of collaborative robots becoming a larger part of intelligent automation systems. The authors propose a ROS architecture as a solution to challenges in communication of complex automation systems. Eros et al. investigate the usage of ROS2 and making use of the publish-subscribe model to exchange information between different components of the automation system. Despite being a comprehensive study, future work is required to further develop stateless nodes for increased reusability beyond this particular application they use.

Now that we have a strong understanding of the foundation of ROS, including its features, capabilities, and applications in robotics, it is worth learning about Gazebo and its place in virtual applications of ROS.

2.2 Gazebo Simulation Environment

While ROS acts as the middleware that provides a framework for developing the robot software, Gazebo is the base environment used to create and model simulations in collaboration with ROS. Gazebo is an open-source 3-D virtual robotics simulator that provides a realistic platform to create, develop, and test robotic behaviours. It allows developers to rapidly test algorithms and design robots virtually, even for users without access to robotics hardware (Koenig, 2004). Gazebo allows you to test robotic designs in a virtual simulation as

a fast, safe, cost-effective method which enables users to use multiple high-performance physics engines such as ODE, sensor models, and 3-D visualization environments to create high quality simulations.

Although this project will make use of Gazebo as the chosen virtual simulation environment, other suitable environments were also considered such as Unity3D which is a popular game engine that has gained traction in robot simulation due to its flexibility and powerful graphics capabilities. It allows developers to create 3D environments, import robot models, and simulate behaviours while integrating physics engines like NVIDIA PhysX (Platt et al., 2022). The challenges with Unity come with its integration with ROS as it is far more difficult to integrate in comparison to Gazebo which is ROS-native.

Another simulator considered is Webots which is a professional mobile robot simulation software that provides an integrated development environment for designing robots. Its advantages lie is its user-friendly interface as it is very intuitive, making it easy to design and test robot simulations (Farley et al., 2022). However, it has very limited resources as a result of a small community resulting in fewer online resources and available models. Also, like Unity, it is far more difficult to integrate with ROS compared to Gazebo.

Ultimately, Gazebo is the best choice for simulating robot behaviours in collaboration with ROS as Gazebo is ROS-native, providing seamless interactions between the ROS nodes and the Gazebo environment. Gazebo also boasts a large community of active users, resulting in countless resources, plugins, and community-contributed robots being released regularly.

2.3 Human-following robots: Principles & Applications

Human-following robots are autonomous robotics systems that are designed to track and follow human subjects. The more intricate details regarding the role of the robot can vary based on its requirements. The primary objective of the human-following robot is to maintain a safe distance from the human, while effectively keeping track of the human while navigating through an environment. This is done by employing a range of computer vision and machine learning techniques to interpret movements from the human and respond accordingly. Robot tracking has many uses including: care robots in nursing, carrying luggage in airports, walking assistance, aiding the disabled, and acting as guides for tourists in places like a museum.

An example of the necessity and implementation of human-following robots is described by Sonoura et al. (2008) as they developed a person following robot with vision-based and sensor fusion tracking algorithms. Based on demographics at the time, Japan was experiencing a combination of an aging population and declining birth rate. As such, symbiotic robots that could aid in daily life and take care of children and the elderly grew largely in interest. The target of the robots was to identify a human and provide real-life services to the human. The authors found that using a robot solely with cameras proved to be unsatisfactory in the event of the human moving at fast speed. Hence, they combined the camera vision with a LRF (Laser range finder) to accurately measure distance from the human and gauge the speed of the human. Based on this comprehensive robot design, the ‘ApriAttenda’ (robot) was able to track a specified human at his/her pace, while also avoiding obstacles by making use of ultrasonic sensors. Sonoura et al. discussed the problems they encountered during creating the tracking algorithm, specifically with vision-based and LRF tracking. When a robot is controlled by feedback using image processing, the frame rate and matching accuracy of the image data contribute greatly to the kinematic mobility of the robot. Any drop-off in computer performance could result in latency and information transfer delays, resulting in a slower responding robot. They also found that it was difficult to detect sets of legs using LRF when there were two or more humans walking adjacently.

Figure 3 - ApriAttenda human-helper robot (GadetsLatest, 2016)

While Sonoura et al. focused on developing a robot based on the integration of vision-based and laser-based tracking, Tripathi et al (2021) discuss a robot system that utilizes ultrasonic and IR sensors due their wide connection area, small size, reduced reliance on UV, and low memory and cost in comparison to LRF. While both robot modifications are satisfactory, it shows that the design of a human-following robot can be based on the capacity of resources and requirements for the robot, giving flexibility and room for improvement in any human-following robot.

More recent studies such as “Real-time smart lighting control using human motion tracking from depth camera” (Chun et al., 2014) proposed a new approach where multiple cameras are used so that the robot performs better even in more crowded environments by making use of a smart lighting control system using depth cameras.

Building further on computer vision algorithms, Katz (2016) discusses through a range of different tracking methods. Vision-based tracking is the most popular technique with algorithms such as particle filtering and mean-shift algorithm. Particle filters represent a distribution by a set of weighted samples called particles. The weighted distribution is updated over time using the set equations in the algorithm. Another tracking method was to make use of laser-based tracking using a LRF to provide more precise information, as well as being less influenced by lighting conditions (an issue that is experienced in vision-based tracking). It is common practice to integrate both these techniques to optimise the robot where each sensor makes up for the limitations of the other one (as done with the AppriAttenda earlier).

2.4 Human-following robots in Gazebo

Now that we have discussed physical human-following robots on a broad spectrum, it is useful to look at studies which implement human-following robots virtually. While human-following robots are typically designed for real-world applications, only making use of virtual environments as a cost & time effective tool to scale and develop robots in a safe manner, there are also applications for these robots in the virtual world. These robots can act as tour guides in VR, be used to tutor and help students in school, and interact with humans in videogames. Exploring human-following robots in a virtual world offers opportunities for research and testing that is beneficial in creating immersive experiences both virtually and in person.

Koenig & Howard (2004) discuss the design principles and usage paradigms of Gazebo as they create a human-following robot that can operate in a dynamic environment. They introduce Gazebo as a powerful simulation tool, making it integral to the R&D of robotics due to its capabilities in modelling and simulating robots in complex environments. The paper discusses the architecture of Gazebo, consisting of a server and client interface, and how Gazebo makes use of a range of sensors and physics engine to create a realistic environment. Gazebo provides sensor models such as kinetic depth sensors, RGB sensors, cameras, lidars, and range finders that are able to generate realistic sensor data that the human-following robot can utilise for perception and navigation tasks. Koenig & Howard also talk about the importance of ROS and its collaboration with Gazebo to allow for smooth communication between the two systems. Despite being a very sophisticated paper that clearly highlights the functionality of Gazebo and how it can be implemented with ROS to design robots, the authors highlight their limitations of Gazebo as it is almost impossible to replicate certain environments like soil, sand, grass, or any deformable or fluid surface. Hence, while maximizing the potential of Gazebo will greatly help in the development of a robot, real-world robots cannot be perfected for their intended use until they are tested physically rather than virtually.

The limitations mentioned by Koenig & Howard are further validated through a study by Sokolov et al. (2016) as they carried out 3D modelling and simulation of a crawler robot in Gazebo. The goal of this study was to attempt creating robots that could operate in unreachable and dangerous environments, such as search-and-rescue robots. They attempted to create a robot that could adapt to different environments using three basic motions: raising and lowering flippers, and locomotion through the right and left four-wheels set rotation. A difficulty they experienced was being unable to properly simulate uneven components of the unmanned ground vehicle (UGV) or simulating uneven environment conditions too.

The integration of ROS and Gazebo has become a standard approach for simulating and developing robotic systems. Another application of this can be shown through a study carried out by Takaya et al. (2016) as they investigate the development of a simulation environment for testing mobile robots. The authors highlight the importance of virtual simulations in robotics, as it is a reliable and efficient testing environment for mobile robots. The core contents of the paper are very similar to the study carried out by Koenig & Howard (2004) as they discuss the interactions between ROS & Gazebo, setting up a simulation environment consisting of a robot with sensors, and a testing world. Takaya et al. test the robot in various scenarios including navigation tasks and obstacle avoidance. Based on these tests, the accuracy of sensor models, and performance of the robotic algorithms for navigation & obstacle avoidance can be determined. The paper validated that a reliable environment for simulation and control of robots could be carried out virtually using Gazebo.

2.5 ML & RL for Robot Training

Machine Learning (ML) techniques have revolutionized the development of human-following robots as it has enabled them to learn and adapt their behaviours based on large amounts of data and training algorithms. These techniques & algorithms, particularly those used for object detection, play a crucial role in identifying and tracking specific objects in unique environments. YOLO is a notable algorithm popularly used for real-time object detection tasks (Fang, 2019). This algorithm can be used to train a robot to recognise and interact with humans using supervised machine learning, while also making use of custom image datasets. Typically, the robot has a trained backbone, performing to a high degree due to being exposed to large datasets consisting of a diverse set of images of many different objects in different poses, lighting, and backgrounds. The robot can then be trained further on custom datasets with objects of interest, so that it can further improve its performance for its unique task. Training robots with image sets allows them to acquire a visual understanding of their environment, allowing for smooth interactions. However, the development of the robot can be limited by biased or insufficient data which can lead to poor training and lead to generalisation.

Reinforcement learning (RL) is a machine learning approach used to train human-following robots. RL algorithms allow the robot to learn using a feedback control loop so that it can adapt and optimise over time based on its past experiences (Lin, 1992). The robot is able to interact with the environment, receiving feedback in the form of reward systems, or penalties (depending on the algorithm used), in order to update behavioural patterns. In human-following robots, RL is used to train the robot on images of the human including any distinguishing features, so that in a new environment from the training environment, the robot can carry out its function to the same degree. While the initial robot design mentioned previously, consisting of sensors and cameras, may be sufficient and work fairly well, RL allows the robot to autonomously adapt and improve its behaviour, making it a useful tool to maximise robot optimization. RL allows for robots that are more adaptable as they learn from experience, more optimised to improve their decision-making, operate better in complex and dynamic environments, and more generalised to apply their learned behaviour into new environments.

Some commonly used algorithms used for RL in robot training include Markov Decision Process, Q-Learning, Deep Q-Networks, Proximal Policy Optimization, Actor-critic methods, Imitation learning, Reward shaping. Being as DQN is the most popular algorithm, it is worth looking into literature that makes use of it. Sasaki et al. (2017) carried out a study on vision-based mobile robot learning using Deep Q-Networks (DQN). They discuss how robots have the ability to learn and acquire good behaviours. They also make use of Profit Sharing Method in addition to DQN to accelerate the learning process which can sometimes take long depending on mass of data and computer capabilities as Profit sharing is an exploitation-oriented RL method.

Agrawal et al. (2022) carry out a similar study where they implement a different ML/RL algorithm; YOLO into real time object detection and tracking. YOLO works on the basic idea of dividing images into smaller squares to create a small matrix within which class probabilities for each cell can be determined. For each grid cell in the matrix, YOLO predicts bounding boxes to produce a confidence score which represents the likelihood of the box containing the desired object. They explain how the key advantages of YOLO lie in its simplicity and speed as it produces the entire image at once, which proves to be advantageous for real time object detection scenarios. However, YOLO has its drawbacks too as it struggles to identify small objects or objects that are grouped together due to its fixed grid size and coarse spatial resolution.

Ultimately, implementing a complementary ML/RL algorithm into the robot's design will aid in designing a more intelligent robot that is able to make informed decisions in different scenarios.

2.6 Development of Robots in HRC

Robots have traditionally been used to carry out simple, repetitive tasks with minimal human interaction. With the advancement of technology, there is a demand for robots that are able to operate alongside humans within a collaborative environment. Goodrich & Schultz (2008) carried out a comprehensive survey on human-robot interaction, exploring the transition of robots from being standalone machines to interactive, socially aware systems capable of human collaboration to carry out complex tasks. The authors discussed the integration of sensing technologies such as vision and speech recognition to aid in HRC.

He et al. (2020) shows the implementation of HRC by designing a robot that aids humans in the manufacturing industry. The study takes advantage of both the flexibility of human workers, and the accuracy of robots while using simulation as a useful tool to test models in Gazebo. Gazebo is used to virtually test HRC due to security risks that a human operator could be exposed to. HRC provides an economical solution as it allows for higher flexibility and shorter production cycles within the manufacturing industry, as the robots are tasked with completing fast, accurate, and repetitive tasks, leaving the humans to carry out more sensitive tasks. This study highlights the limits of older robots that are limited to only completing simple repetitive tasks, as the manufacturing industry separates the human jobs and robot jobs to a certain extent. Further studies into a hybrid environment could aid in optimizing processes to reduce operational time.

Figure 4 - HRC in industry (Weber, 2014)

More complex applications of HRC in a virtual environment were tested during a study by Richert et al. (2016) as they created a robot that could alleviate physical limitations set by humans. Through the hybrid partnership and effective operation between the human and robot, they aimed to create a seamless process where the robot could aid the human within a range of activities such as assembling a product, high-tech surgeries, and a kitchen helper robot. The performance of the model was evaluated based on robot accuracy and success, and more importantly on the human wellbeing and feasibility of transferring this process into real-life.

2.7 Challenges & Future Directions

Simulation in robotics has a range of opportunities and challenges. Having discussed the opportunities and applications above, being aware of the barriers that need to be overcome is necessary for the development of simulations in Robotics. The biggest challenge is the inability of simulations to be able to create a perfect environment that can be directly translated into real-life. For most scenarios, the simulation can provide a base case, that can only be developed further once the physical robot has been tested. Perfecting the simulation

environment so that it can recreate unique environments is a difficult task that needs to be overcome so that simulations can be relied on more (Choi et al., 2021). Another large challenge with virtual simulations is the difficulty in implementing uncertainties on the robot such as wear and tear, or unexpected damages.

Additionally, the development of human-following robots poses its own sets of limitations that need to be overcome. One of the main limitations is having to create a sophisticated tracking algorithm that could track one particular human within a crowded environment consisting of many humans, or in instances where the human may be moving erratically or at different speeds (Inkulu et al., 2021). Since the human-following robot will be trained using images of a human, it may be over-trained to the image set given which may include the human in a single pair of clothes, without any distinguishing feature that could differentiate that particular human from a crowd of people. Similarly, if a robot is designed to track a pair of legs for example, a long coat or other pieces of clothing could hinder this tracking as mentioned by Sonoura et al.

Regarding HRC in particular, there are a range of new challenges and opportunities as creating socially intelligent robots that can seamlessly integrate into human-centric environments is an incredibly difficult task. The main challenge is definitely the creation of effective human-robot communication (Choi et al.,2021). However, as humans rely on natural language, non-verbal cues, and social context, developing such communication systems within a robot is highly difficult to carry out to perfection. Sensor integration is another challenge as the robot needs to be able to perceive and understand the environment they are in perfectly, so they can reliably detect any movements and respond accordingly. This can be particularly difficult as humans exhibit a range of different movements and behaviours, which are not happening repeatedly in the same order. This could create a slight lack of safety and trust in HRC in industries where a fault could lead to fatalities. Thirdly, human environments are highly dynamic, requiring flexibility and adaptability in order of operations and tasks being carried out. Designing a robot that can comprehend changes and respond with a solution rapidly by adjusting their behaviour is a complex task when accounting for how that response from the robot aligns with how the human responds which is not always predictable.

Beyond the extensive list of technical challenges HRC and human-following robots pose, it is also important to consider the ethical implications (Capurro et al., 2009) of this technology as the ultimate priority is to maintain human safety and standard of life. One of the biggest ethical considerations is privacy as human-following robots typically have a range of sensors and cameras to interact with humans. It is crucial that the data collected by the robot is kept confidential and handled appropriately. The human should also have the power to control how their data is being handled to prevent misuse or unauthorised access to sensitive information. Safety is another factor to consider as developers must design the robots in a way that there is no risk of physical harm to humans (Sheridan, 2016). The robots should have predictable, reliable behaviours to prevent any unexpected situations from occurring. Another ethical factor is the equity and fairness of robots, as they should be made openly accessibly and relatively affordable in order to prevent marginalization or exclusion of individuals. Building on the mentioned ethical factors, humans must always have the right of consent and power over the robots so that a human can instruct the robot to cease following them for example in order to keep control of their own boundaries and personal space. Finally, it is extremely important to address issues of accountability and liability by developing clear guidelines, frameworks, and laws in the case that a robot causes an accident.

Addressing and overcoming these challenges through further research and development of HRC and AI through ethical means, could result in a world where robots and humans safely,

and efficiently interact within close proximity to carry out tasks in a seamless manner, over a range of domains and human-centric environments.

3 Methodology, Design, and Implementation

3.1 System Overview

There are several methodologies that can be carried out in order to design a human-following robot using different sensors, algorithms, and training approaches. This methodology is one of many approaches to create a human-following robot from the ground up, discussing relevant theory of designs and processes where applicable, as well as giving an in-depth detail into each step of the process in creating the final virtual robot that is able to follow a human within Gazebo. This section highlights both the theory of project, and the design & implementation of different aspects within the project. The methodology will contain information regarding the research and experiments involved, as well as the systematic process involved in completing this project, while delving into high-level architectural and conceptual aspects of the system and the technical details of how they work.

Firstly, an explanation will be given into the creation of the conda virtual environment, justifying any relevant packages or versions which have been used. Secondly, since the chosen robot that will be used in this project is the TurtleBot3 Waffle Pi, a description of its key features will be listed to understand its functionality and capabilities prior to any training or application of the robot. After this, different Gazebo scenarios will be discussed, within which a human model walks in a particular pathway, as coded within the Gazebo custom worlds xml file.

Now that the base of the design is complete, custom human image data within Gazebo is collected, after which the images are augmented and modified in order to fit the compatible COCO format that can then be used in the model training process. PyTorch's MMDetection is used to apply the relevant algorithm (YOLOv3) after consideration between two different algorithms. After creating the YOLOv3 configuration file, a model can be trained based on the image dataset. An explanation is given into how the testing of the model is carried out, particularly what the results show and how they give an insight into the quality of the model. Finally, the trained detection model is implemented into an ROS code that applies into the TurtleBot3 robot and directs the robot to follow the human throughout a range of scenarios based on its human detection capabilities to see how well it performs in increasingly difficult scenarios.

3.2 Configuring the Virtual Environment

Several steps are carried out in the creation of the virtual environment in which the project will be carried out. Properly setting up the environment is essential to ensure compatibility and smooth execution of various components that will be involved in the project. Firstly, a 'bash' file is initialized using the code in Appendix A1. This configuration step is required to ensure that the Conda system will operate smoothly within the terminal environment. This is generally good practice as it will ensure that the base conda environment is not tampered with. By initializing Conda into the Bash shell, it allows you to automatically activate and deactivate Conda environments, as well as other commands so that they can immediately be carried out upon launching the conda environment (Verma, 2022).

The code in Appendix A2 is used to create the python environment. For compatibility reasons with ROS Foxy and PyTorch, Python 3.8.10 is used as it is able to carry out the ROS commands as intended, as well as run deep-learning algorithms on PyTorch.

The final line of code in Appendix A2 is used to install PyTorch with GPU support by utilizing CUDA 11.8 which is the latest CUDA version available on the PC being used. CUDA is a computing platform developed by NVIDIA which allows developers to harness the power of their GPUs to accelerate computational tasks which can typically be intensive on the computer (Ilievski, 2018). When using PyTorch for deep learning, enabling GPU support will significantly

increase the rate of training and inference, as the computational load will be offset onto the GPU for faster execution times and quicker model convergence.

Appendix A3 presents the code used to install MMDetection which is the deep learning framework that is used to apply object detection algorithms in this project (Chen et al., 2019). This toolbox is built on top of the PyTorch deep learning framework to provide an optimal, easy-to-use deep learning application that is highly customizable. Mmengine and mmcv are installed which allow MMDetection engine execution, and provides the essential libraries required for computer vision tasks such as object detection. The MMDetection repository can then be obtained from its GitHub repository, containing source code, models, configuration files, and several more codes to carry out tasks within MMDetection.

In addition to the conda initialization code present in the bashrc file, more commands as seen in Appendix A4 have also been echoed into the bashrc file. The first line instructs the terminal to instantly open the conda environment ‘openmmlab’ every time a new terminal window opens, as this environment has the relevant packages and dependencies for the project. The second command sources the setup script for ROS Foxy, so that the terminal can recognize and execute ROS commands. Similarly, the third command sources the Gazebo setup script which enables the use of Gazebo functionalities. The fourth and fifth commands are more relevant to the computer being operated on as it ensures that the environment is available on this computer without any interferences from other users. The final code appends the path to the custom ‘GazeboModels’ folder that contains important files such as the TurtleBot3 Waffle files, as well as the COLLADA files containing the human models which will be discussed in further detail later. With these additions, the bashrc file is complete with all necessary inputs in order for the environment to launch with the correct ROS framework and file directories.

The additional packages in Appendix A5 are installed as they are required in order to carry out the functionalities of this project. Catkin, lark and empy aid in handling, building, managing ROS packages, and parsing in ROS. The OpenCV package offers computer vision capabilities and is necessary in order to carry out any image processing tasks. Compatible versions of NumPy and Open-CV are automatically installed alongside the PyTorch installation carried out previously so that there are no dependency complications.

Appendix A6 provides code related to building ROS packages in the dev workspace consisting of important build and installation commands for TurtleBot3 packages. Running ‘colcon build’ compiles and builds the packages, creating executable binaries. Running the command for a second time performs an incremental build to ensure all changes and updates have been incorporated into the built packages.

Upon completion of all the steps in this section, the base environment for the project has been set up to ensure a seamless development experience moving forward.

3.3 TurtleBot3 Waffle Pi Features

TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot which has open access virtually for all. TurtleBot3 aims to increase robot capability and quality without the costly prices that may come with it. Waffle Pi is a specific model within the TurtleBot3 series which provides a compact design with various sensors, enabling the robot to perceive and interact with its surrounding environment (Amsters, 2020). Some features of TurtleBot3 that make it advantageous in comparison to competitors is that it is open source software which can be downloaded, modified, developed, and shared among users. Being the most popular open source robot, TurtleBot3 is also extremely affordable and flexible in design as it is easy to maintain and reconfigure. An online version of the Waffle Pi is available and can be imported into Gazebo for testing.

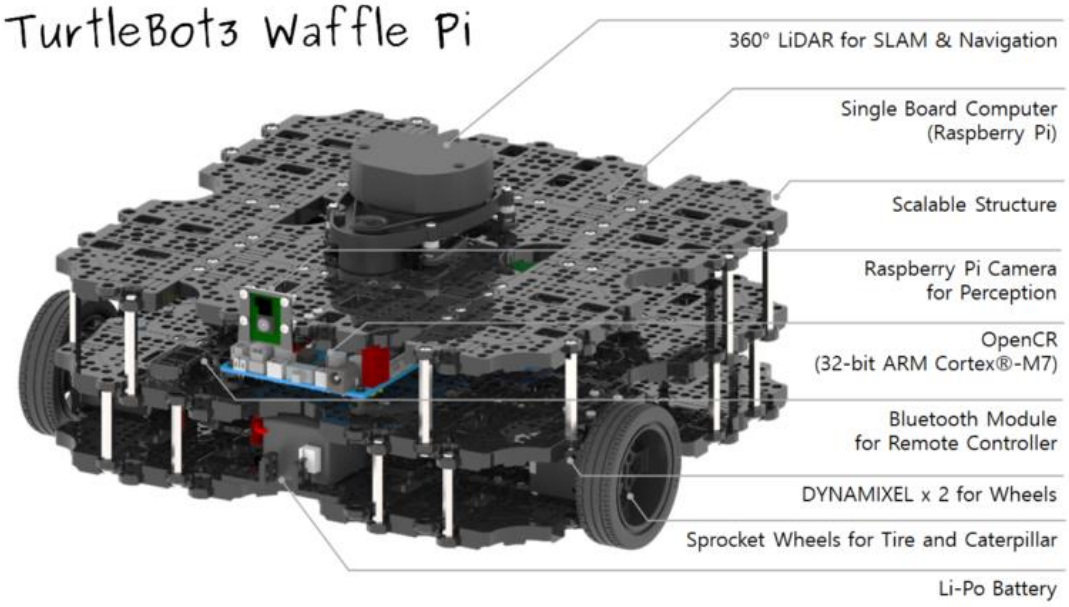


Figure 5 - Components of the Robot (Robotis, 2023)

Figure 5 shows the basic components of the Waffle Pi robot. The main features of interest in this project is the 360 degree LiDAR and the Camera for perception as these are what will be used in collaboration with the ROS human detection and tracking algorithms in order to carry out its purpose.

In order to create and apply the robot into a virtual environment, a configuration file and an SDF file are created for the Waffle Pi robot. Both of these are openly accessible and contain meaningful insights into the design and components of the robot as seen in Appendix B. The SDF file defines the structure and properties of the robot, whereas the configuration file is used to customize the behaviour and settings of various components within the simulation. Note that the SDF & Configuration file format is used for more than just the robot as it is the typical format through which the Gazebo world can also be designed, where physics and objects can also be specified and implemented. Both files work together in order to create a detailed and customizable simulation environment.

The TurtleBot3 Waffle Pi is a versatile robot designed to perform tasks and navigate environments effectively. The SDF file (refer to Appendix B2) offers deep insight into the robots' physical structure and key component, which may also overlap with Figure 5 above. The main features and components based on the SDF file are as follows:

Base Link: This is the foundation of the robot's structure as it contains the inertial properties, collision model, and visual representation. As the core of the robot, it is referenced in other components of the robot.

IMU Link: This sensor is able to capture angular velocity and linear acceleration data which contributes to the Waffle Pi's understanding of its orientation, movement, and position within the environment.

LiDAR Sensor (base_scan): The LiDAR sensor is able to generate a 360 degree point cloud representation of its surroundings, making it a vital component of the robot for mapping, navigation, and obstacle detection. The use of LRF also aids in gauging an idea of the distance between the robot and its surroundings, allowing the robot to navigate more effectively.

Camera Sensor (camera_rgb_optical_frame): The equipped camera captures visual information as it provides a wide horizontal field of view and captures images that can contribute to later tasks such as human detection, image processing, and human following.

Wheels & Caster Wheels: The two wheels on each side are responsible for the robot's differential drive motion, whereas the caster wheels provide stability and support to the robot so that it can move smoothly.

Joints: These connect robot components so that they can interact in order to carry out tasks.

Plugins: Several plugins have also been used to enhance the robot's capabilities.

While the Waffle Pi combines a range of different features, the Camera and LiDAR are of most interest as they gain information in order to perceive and interact with the simulation environment, allowing the robot to navigate based on informed decisions.

The camera provides visual information as it captures images that can be processed for tasks like object detection. The camera sensor's configuration includes parameters like field of view, image resolution, noise, etc. Noise is an interesting parameter as it simulates factors that could hinder the camera vision similar to real-world influences that may hinder vision (Rekleitis, 2023).

The LiDAR (Light Detection & Ranging) sensor allows the robot to perceive its environment in greater detail. It works by emitting laser beams and measuring the time it takes for these lasers to reflect off the surrounding and return to the sensor (Hutabarat et al., 2019). Through this method, it can gauge the distance of objects and gain a detailed understanding of its surroundings by creating a 360-degree cloud image of the environment.

Figure 6 shows the Waffle Pi in gazebo, where the command to run the world is given in Appendix B3. The blue lines indicate the LiDAR sensor obtaining information about its surroundings as discussed above. The camera can clearly be seen as it is able to see the human model in front of it.

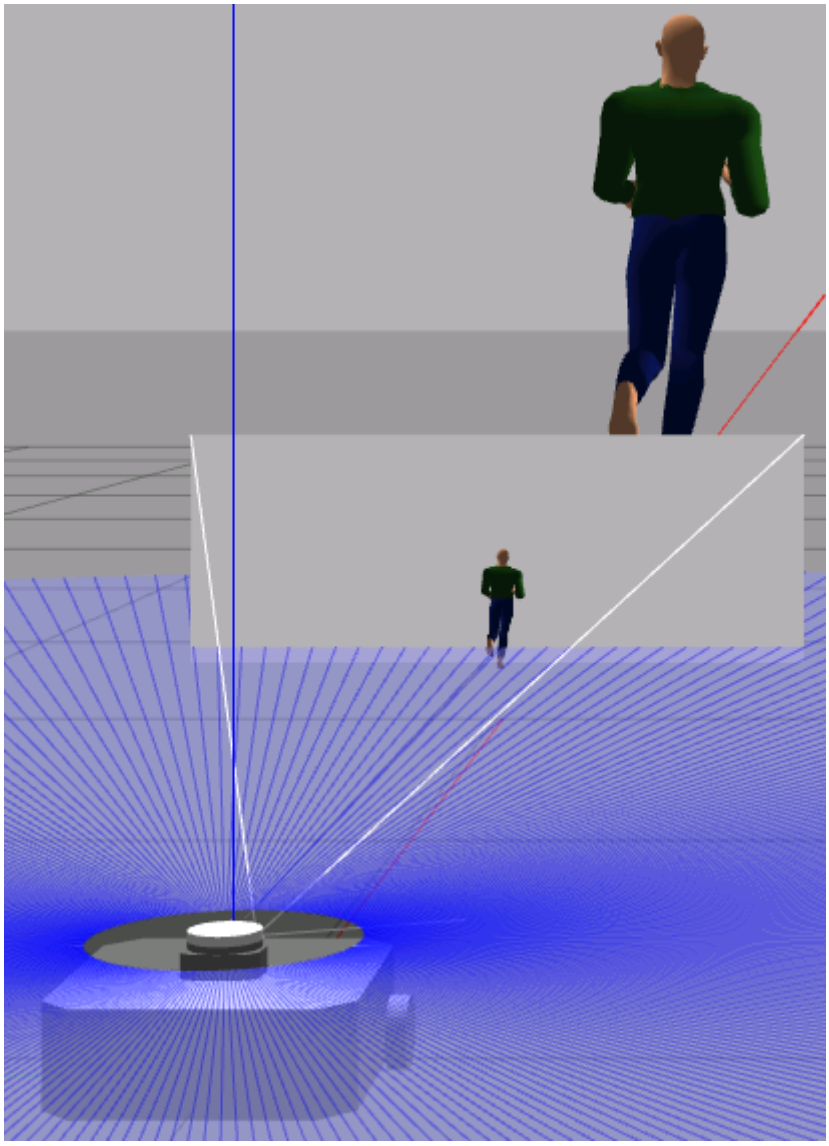


Figure 6 - Image of Waffle Pi in Gazebo

3.4 Creating the Virtual Scenarios

Now that a virtual environment has been configured and the designs and features of TurtleBot3 have been discussed in detail, the next stage of the methodology investigates importing a human model from COLLADA as a '.dae' file (GitHub, 2014). A .dae file or COLLADA file uses XML format and is used to describe and exchange 3D digital assets between applications. COLLADA plays a significant role in creating and simulating human models, including animations like walking, or running within the Gazebo environment. The human model itself is created using a 3D modelling software 'Blender' as it designs the human as a collection of meshes, bones, joints, and associated animations (refer to Appendix C1). COLLADA files are compatible with Gazebo and the 3D human models can be imported into the simulation environment where they conform to the Gazebo physics engine, allowing the human to interact and coexist within the environment as seen in Figure 7. Figure 7 shows the resulting human model that has been created in Gazebo with all its basic features and animation sequences activated as the model will simulate walking once waypoints are specified in the Gazebo world later on. In summary, COLLADA files bridge the gap between 3D modelling software, animation sequences, and simulation environments like Gazebo.

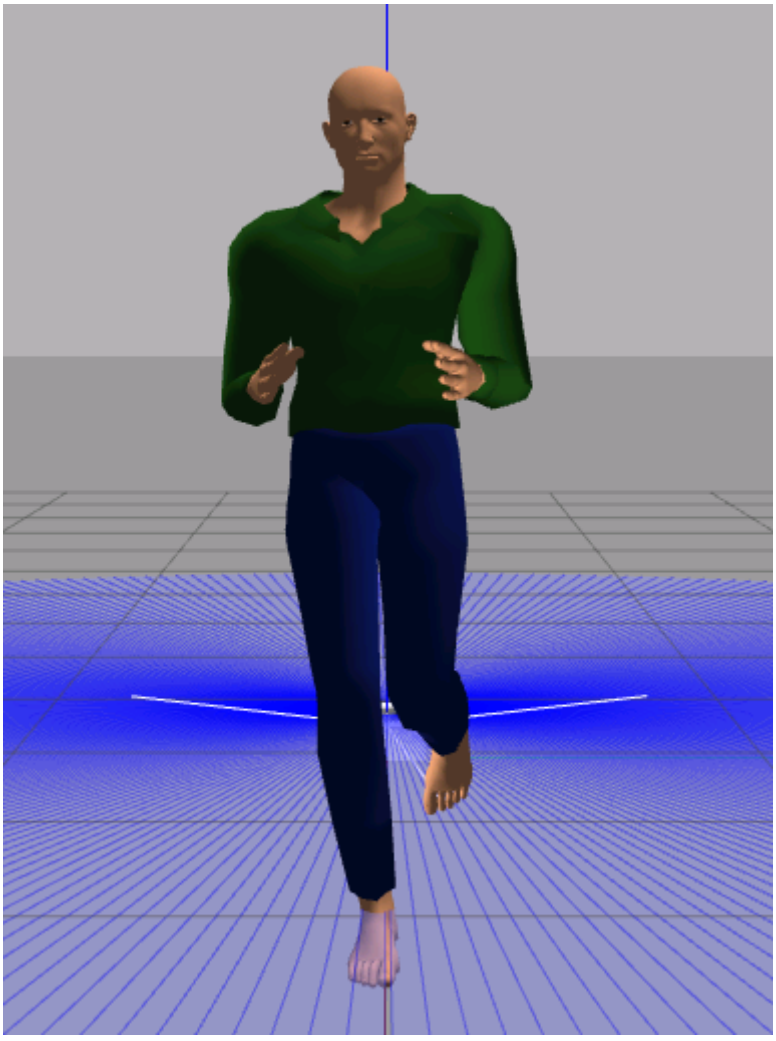


Figure 7 - Human Model in Gazebo

The goal of this project is to create a human-following robot that can detect and follow the human model in Figure 7 around the Gazebo Simulation Environment. In order to test the effectiveness of the robot, the human model is simulated in several different scenarios where the human follows different pathways at different speeds obstacles to see how the robot responds to these changes in simulation conditions.

The first scenario is a very simple simulation that ensures that the robot is functioning and is able to follow a human that is running in a straight line ahead. The human model will run 13 metres ahead in 15 seconds. Appendix C2 contains the basic configuration and properties of the empty gazebo world in the first scenario. This code will exist in every scenario as it essentially includes setup of the work including the physics, sunlight, and aspects of the environment which contribute to the overall behaviour and appearance.

The next section of the code in Appendix C3 is the command written in order to control the navigation of the human during the first scenario (walk_straight.world). This code defines an ‘actor’ within Gazebo which is scripted to perform a running animation along a specified trajectory. While most of this code is self-explanatory, the key control measure is the use of waypoints as they indicate the time and pose of each movement. Based on the time you specify between two waypoints; the human model will travel at a faster or slower speed. The pose determines position and orientation of the actor at each waypoint. Pose consists of 6 values (X, Y, Z, Roll, Pitch, Yaw) where X, Y, Z each control the displacement of the object

along different axes. Roll represents the rotation of an object around the X-axis (Svenstrup, 2009). A positive value indicates a clockwise rotation. Pitch represents rotation around the Y-axis, where a positive value will indicate clockwise rotation along the axis. Finally, Yaw represents the rotation around the Z-axis. Collectively, these 6 elements define the spatial position and orientation of an object in a 3D environment. A visual of this scenario as well as the other scenarios will be available through snippets in their respective sub-sections in Section C of the Appendix. These snippets will show the position of the human as the scenarios are activated, to show how it moves in accordance with the xml file used to run the Gazebo world.

The final snippet of the world SDF file code for Scenario 1 (as well as every scenario) can be seen in Appendix C4. This is used to call the TurtleBot3 Waffle Pi into the world. Again, the pose is specified at the centre of the simulation as it has all values as 0. Including this code into the world SDF file will ensure that the TurtleBot3 Waffle Pi will automatically be loaded into the Gazebo world as it launches.

Now that the format of the world file including trajectories, waypoints, and times has been explained, a few more environments with different scenarios are also created to progressively challenge the robot's performance. Due to the increased difficulty, it is highly likely the robot underperforms in many scenarios, upon which an investigation will be carried out to explain the robot's limitations and how they could be overcome.

Scenario 2 (walk_back.world) is a Gazebo world created to ensure that the human detection algorithm works properly. This world will not be used to test human tracking, rather it is solely used to test the human detection algorithm. The code for this can be seen in Appendix C5 as it essentially directs the human model to walk forward 13 metres, stop and turn around, and walk back to the starting point, after which the human turns around again to repeat this pathway. The Waffle Pi is able to view this entire simulation from a stand-still point, allowing it to be an ideal scenario in testing the human detection algorithm within which the human is travelling at various speeds, and seen from different distances and angles. This will allow the user to gauge an understanding into the accuracy and limitations of the trained robot's detection algorithm.

The next scenario 'walk_rect.world' (refer to Appendix C6) controls the pathway of the human model so that it walks in a 5m X 10m rectangle over 80 seconds. Two major changes that can be noticed in this code when compared to the first scenario is the change in direction of the human model, and the change in speed of the human model as it is now making use of the walk.dae file and has a slightly different animation. This scenario aims to see how the robot will perform when trying to detect and follow the human that is 'walking' rather than 'running' as the body movements differ between the two animations. On top of this, the human walks in a large rectangle relatively slower to scenario 1 and 2 to see if the robot can also navigate this pathway and keep track of the human as it changes directions. By changing the 'Yaw' pose (in radians), the orientation of the human model is rotated by 90 degrees at each corner of the rectangle.

Scenario 4 (walk_obstacles.world) in Appendix C7 aims to challenge the robot as much as possible. This scenario is by far the most difficult and it is likely the Waffle Pi will not be able to follow the human accurately and efficiently around the environment due to the several obstacles and moments the human is out of camera range. It shows the human model walk in a straight line between two obstacles, until an obstacle appears directly in front of the human. The human walks around the obstacle and continues forward. This scenario is used to test the robot's capability in avoiding obstacles while following the human around the simulation.

With this, the different scenarios have been formed and are ready for testing the robot's performance in following the human in a range of different scenarios. The intention behind each scenario was to progressively increase the difficulty of the human's movements and surrounding environment to see how the robot performs when challenged with such changes like obstacles, restricted spaces, different angles of the human it is tracking, different speeds the human travels at, and different animation sequences between 'walking' and 'running' to see if these factors have any bearing on the performance of the robot.

3.5 Data Pre-Processing

The next stage of the methodology consists of gathering images of the human model in Gazebo and preparing the dataset ready for the training and testing processes ahead. Good data cleaning and handling ensures that the data is kept in a suitable format that is compatible with the PyTorch algorithms used ahead in order to train the model. In order to obtain images of the actor in the simulation, a simple Gazebo world consisting of the human walking in different trajectories in the TurtleBot house was loaded into. Once in the world, several images of the human model were taken where the human is seen from different heights, angles, and distances. These photos are saved in jpg format as seen in Figure 8.

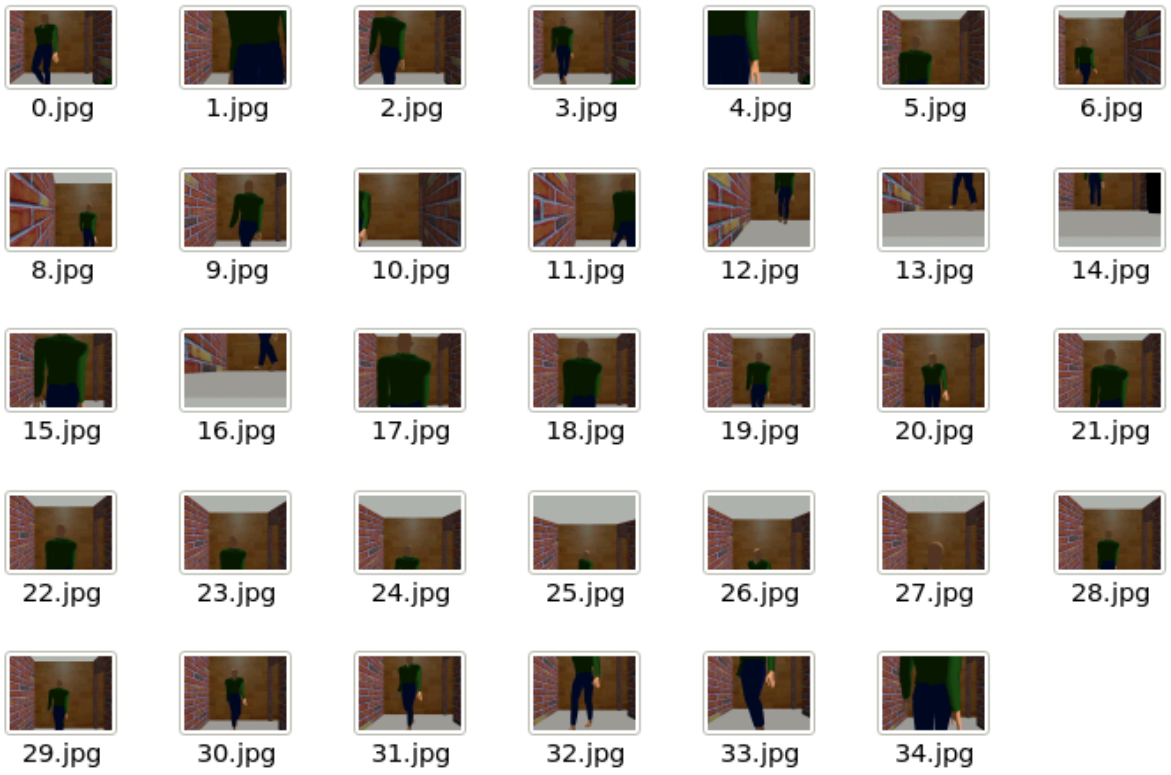


Figure 8 - JPG images of human model

A key stage in this process is the labelling of images which is done using the LabelImg software (Mustamo, 2018). When this software is launched in the terminal, it will open a window where you can access images in your files. By drawing a rectangle box around the object within the image and then assigning a label to this object, an xml file is created that provides information of the object within the rectangular box as shown in Appendix D1.

Once each image has been labelled with the 'person' encapsulated in a rectangular box, a dataset containing a pairing of a jpg file and xml file is created where the jpg file contains the image, and the xml file contains the annotation details. However, when using PyTorch's

MMDetection training process, datasets need to be in the COCO format to be compatible. Additionally, only 34 images are present in the original dataset, hence ideally the dataset needs to be augmented in order to increase the size of the dataset for training. Both these tasks are carried out with the aid of roboflow (Roboflow, 2023) which is an openly accessible website that aids in turning images into information. By inserting the folder containing both the jpg files and xml annotations into the roboflow project space, an annotated version of the image can be seen like in Figure 9.

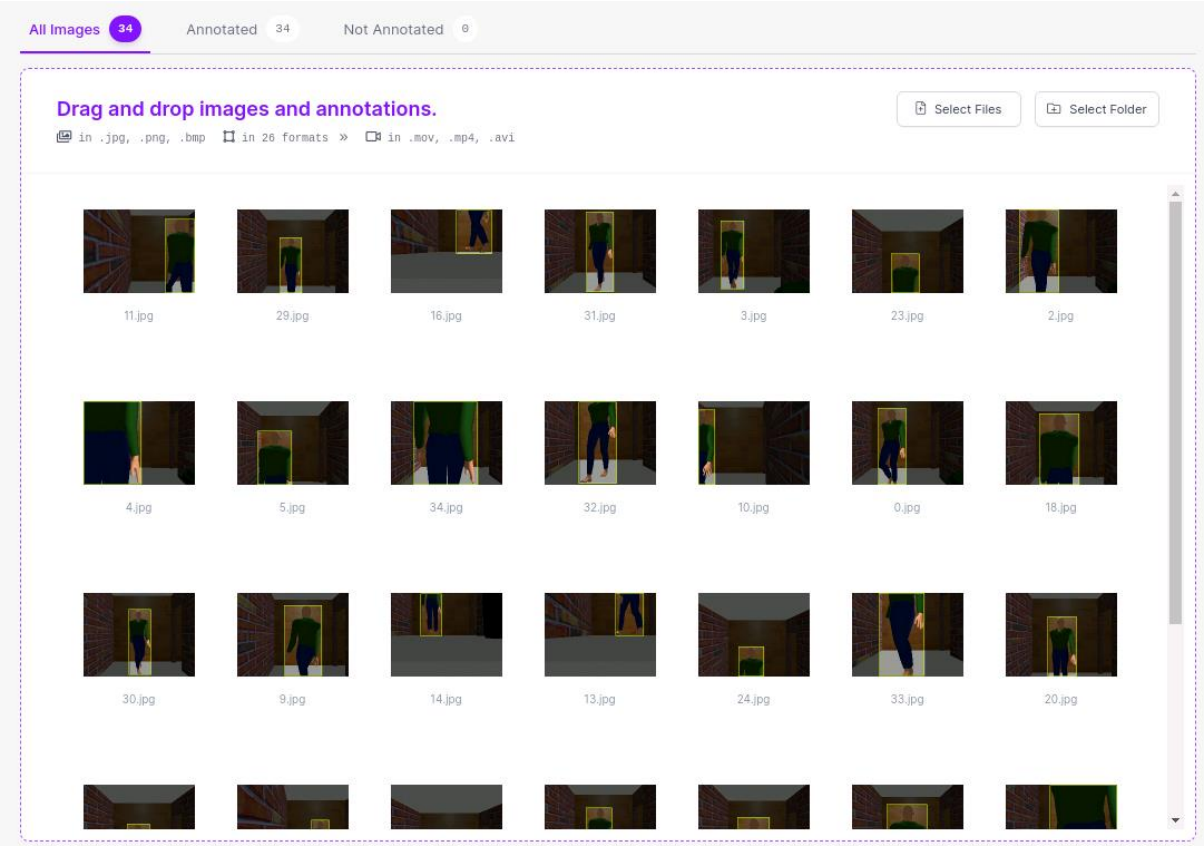


Figure 9 - Roboflow insertion of raw imageset

The images are then split into a training set, validation set, and testing set as is necessary for the machine learning algorithms configuration file further ahead. This split is done randomly so that there is no bias of which pictures are assigned to each category. Additionally, during the pre-processing of the images, data augmentation is carried out as images are randomly rotated by 15 degrees in either direction to create a larger dataset of approximately 81 images with 71/7/3 images split into Training/Validation/Testing respectively. This newly created dataset can then be downloaded back into the file directory containing these meaningful changes as the dataset has been augmented to increase the dataset size, allowing for more in-depth training and increased model accuracy. Additionally, ensuring that the data format has been converted from the jpg/xml format to the COCO format is essential for training the model. This can be confirmed by viewing the JSON file assigned in each folder that provides information about each image within the file in COCO format. COCO is widely used in object detection tasks as it provides a comprehensive and standardized dataset for training and evaluating models. The COCO dataset combines the two main components which are the image data, and the annotations including the bounding box and object category, making it compatible with PyTorch to ensure data can be seamlessly used with machine learning algorithms ahead.

A greater understanding of the COCO format can be given by understanding the snippet of code from the testing datasets 'annotations.json' file in Figure 10. The COCO file below shows that there are 3 images with their image_id, as well as bbox values representing the bounding box coordinates of the detected persons location. The code also shows that all images have had the object within the bounding boxes categorised as 'person'. Similar annotation files are present for both the Training and validation image sets with the file directory looking similar to that in Figure 11 below. The annotated JSON files for the training and validation image sets can be seen in Appendices D2 & D3.


```
1  {"info":{"year":"2023","version":"1","description":"Exported from roboflow.ai"
   ,"contributor":"","url":"https://public.roboflow.ai/object-detection/undefined"
   ,"date_created":"2023-07-20T14:04:04+00:00"},"licenses":[{"id":1,"url":"https
   ://creativecommons.org/licenses/by/4.0/","name":"CC BY 4.0"}],"categories":[{"id":0
   ,"name":"human","supercategory":"none"},{"id":1,"name":"person","supercategory"
   :"human"}],"images":[{"id":0,"license":1,"file_name":"27_jpg.rf
   .0612c7fd2e5f2675d49f33c35d3238ac.jpg","height":640,"width":640,"date_captured":"2023-07
   -20T14:04:04+00:00"},{"id":1,"license":1,"file_name":"16_jpg.rf
   .e1e4c1eac5b6a615437f86f0b90df00a.jpg","height":640,"width":640,"date_captured":"2023-07
   -20T14:04:04+00:00"},{"id":2,"license":1,"file_name":"12_jpg.rf
   .a4e39776b28ce603e2d2813514c0f8cc.jpg","height":640,"width":640,"date_captured":"2023-07
   -20T14:04:04+00:00"}],"annotations":[{"id":0,"image_id":0,"category_id":1,"bbox":[176
   ,387,307.5,252.5],"area":77643.75,"segmentation":[],"iscrowd":0},{"id":1,"image_id":1
   ,"category_id":1,"bbox":[372,2,205.5,338],"area":69459,"segmentation":[],"iscrowd":0}
   ,{"id":2,"image_id":2,"category_id":1,"bbox":[402,2,121.5,436.5],"area":53034.75
   ,"segmentation":[],"iscrowd":0}]}
```


Figure 10 - COCO annotations for Testing Images

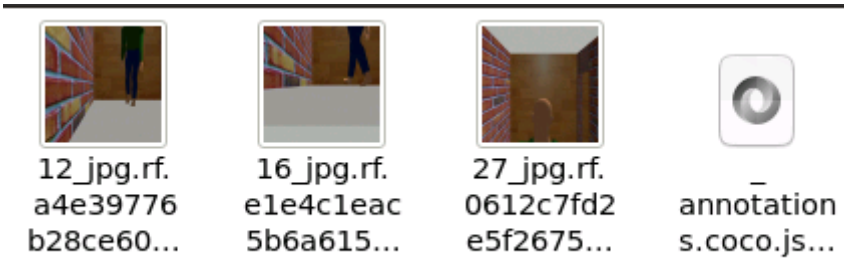


Figure 11 - File contents of the Testing Dataset

To summarise this stage of the methodology, a custom dataset is created consisting of human images within the Gazebo world. The images are labelled, and the data is converted into COCO format and augmented to create the final dataset with more images and the correct annotation format. The images are also split into 3 folders containing Training, Testing, and Validation datasets so that the dataset is ready to be applied into training a model in the next stage.

3.6 PyTorch & MMDetection

MMDetection is an open source object detection toolbox based on PyTorch and part of the OpenMMLab project (MMDetection, 2023). It allows for easy modular design and application of customized object detection frameworks making it the perfect tool for modifying and applying an algorithm from their model zoo of vast architectures. All models in the 'Model Zoo' have been trained on the 'coco_2017_train' dataset and tested on the 'coco_2017_val' dataset

which are large image sets. Additionally, many algorithms have pretrained backbones on ImageNet. ImageNet is a large scale dataset containing millions of images across thousands of categories, hence plays a pivotal role in advancing computer vision and machine learning research.

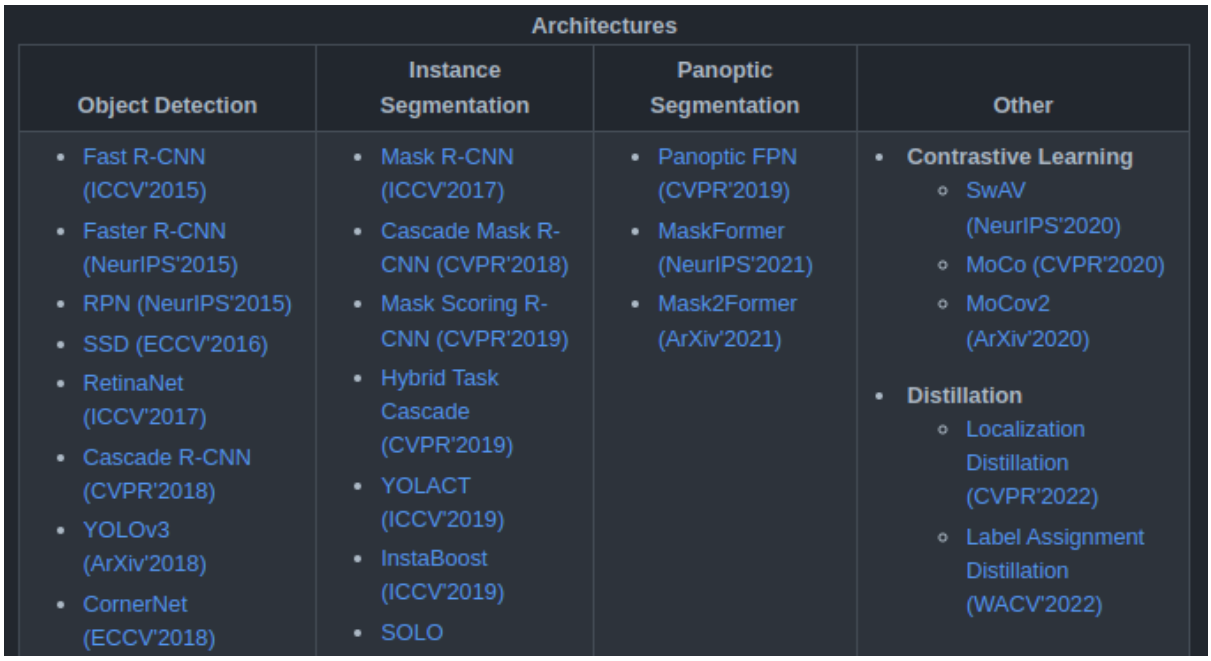


Figure 12 - MMDetection Model Zoo Snippet (Rath, 2022)

MMDetection is built on top of PyTorch and offers a wide range of state-of-the-art object detection models along with their configuration files, architectures, loss function, post-processing, and evaluation. The typical modular architecture MMDetection adopts contains several key components such as backbones, necks, heads, loss functions, post-processing, and evaluation metrics.

Overall, MMDetection is a valuable tool to train and test an object detection model for the TurtleBot3 Waffle Pi. The next section will discuss two key architectures that are typically used in object detection tasks, as the optimal algorithm is decided for this project based on several pros and cons of each algorithm, how they operate, and whether any key features make them more suitable. Note that although the optimal method to determine the best algorithm is to test a range of algorithms and see which performs the best, this is proven to be a difficult and tedious task to implement due to the difficulty in modifying several configuration files in order to merge seamlessly into the customised dataset and file directories, hence the best algorithm is decided prior to applying any model based on the goal of the project and the strengths of each algorithm.

3.7 YOLOv3 vs Fast R-CNN

From the large pool of different architectures available in the MMDetection Model Zoo, two key algorithms were identified due to their stature in object detection tasks as they were provenly some of the best performing models. Fast R-CNN and YOLOv3 were chosen and a choice for the optimal algorithm for a human-following task is carried out. First, it is worth looking into both algorithms, how they work, and where their strengths and weaknesses lie.

Fast R-CNN is a classic object detection framework that builds upon the R-CNN architecture as it aims to address its limitations and improve the object detection speed. This was made possible by several changes to the R-CNN architecture (Gad, 2021). The first component is

the RPN (Region Proposal Network) where Fast R-CNN replaces the selective search method using in R-CNN for generating region proposals. The second and main feature is the RoI (Region of Interest) pooling which extracts equal-length feature vectors from all proposals in the same image. The third component is the shared convolutional features as the backbone on the network is the pre-trained CNN model VGG16. This backbone is used to extract feature maps from the input image. Finally, the RoI are fed into a classifier and regressor where the classifier assigns class labels to the proposals, and the regressor refines the bounding box coordinates for accurate localization. This is a simplified approach to Fast R-CNN as it is a significant improvement from R-CNN due to its increased speed and accuracy.

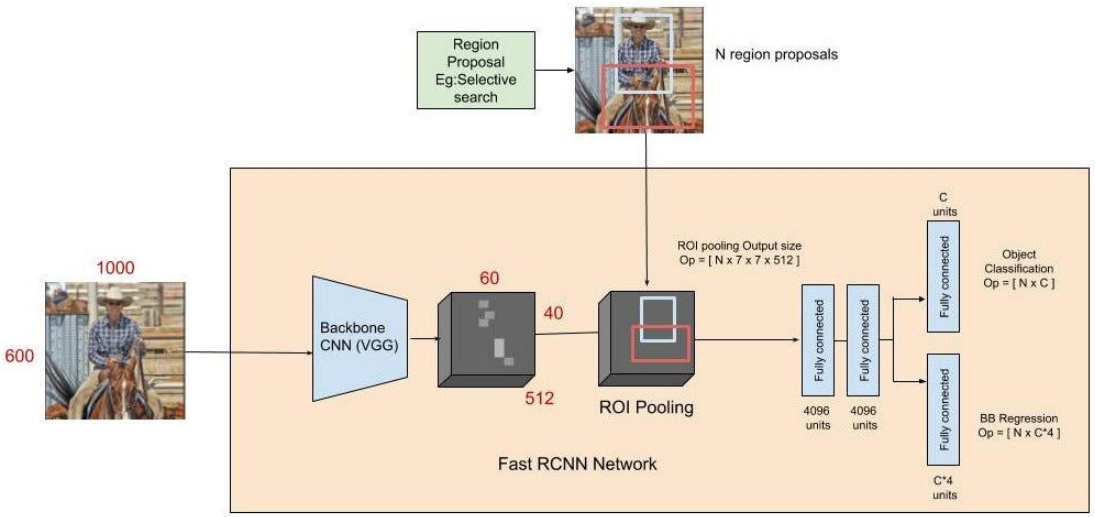


Figure 13 - Fast R-CNN Architecture (Ananth, 2019)

However, Fast R-CNN also comes with its fair share of limitations that make it fall short for this human-following tasks. The first complication which was encountered is the complexity of the algorithm as it has an extreme learning curve to modify, implement, and train this model as it is not very intuitive. Secondly, while being faster that R-CNN, it is still not optimal or suitable for the human-following task where real-time application of the model is required in order to identify the human (Maity, 2021). Finally, cases of proposal limitations were found as the RPN sometimes misses proposals leading to missed object detections. Due to these limitations in terms of complexity and speed compared to other specialized real-time object detection models, YOLOv3 is looked into as it is considered one of the best real-time object detection algorithms making it ideal for this project.

YOLOv3 (You Only Look Once version 3) is a state-of-the-art object detection algorithm known for its real-time performance and accuracy in identifying specific objects in videos, live feeds, or images (Keita, 2022). The YOLOv3 architecture can be simply explained as the algorithm first separates an image into a grid where each grid cell predicts a number of boundary boxes around objects that score highly for the desired object class that has been predefined. Each boundary box has a respective confidence score of how accurate it assumes that prediction is. Boundary boxes are generated by clustering the dimensions of the ground truth boxes from the original dataset. To capture features at multiple scales, YOLOv3 uses a feature pyramid network that combines features from different layers of the neural network (Masurekar, 2020). A key advantage this has over similar algorithms like Fast R-CNN is that it is able to carry out classification and bounding box regression at the same time, making it significantly faster.

YOLOv3 is known for its remarkable real-time object detection capabilities as it can process images and make predictions in single pass. While it is known to have a slightly lower accuracy than R-CNN, the accuracy is still considered to be quite high and satisfactory considering the response time it has.

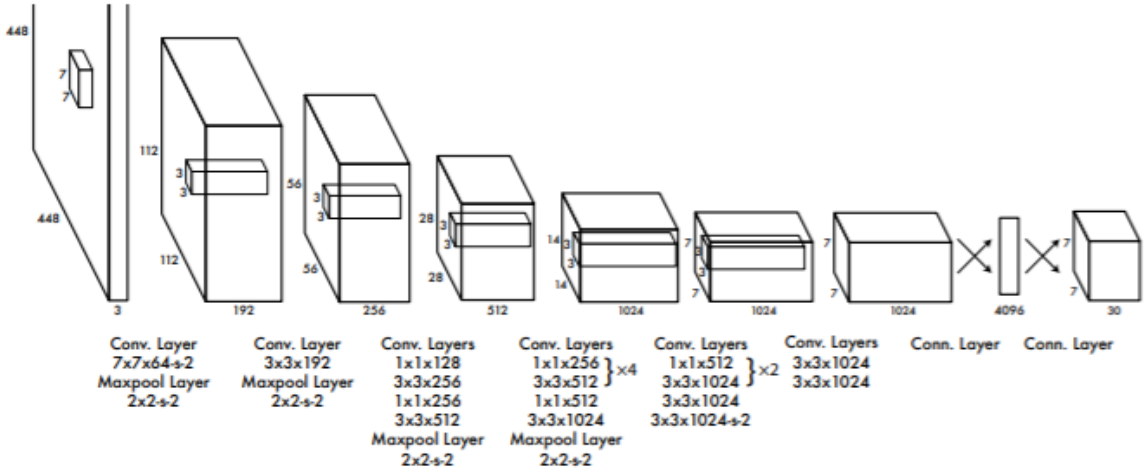


Figure 14 - YOLOv3 Architecture (Keita, 2022)

Despite this comprehensive architecture YOLOv3 boasts, it has its fair share of limitations too as it struggles in detecting small objects due to its coarse grid cell structure. It is also a complex architecture due to its depth and requiring a large dataset. While Fast R-CNN has a slightly higher accuracy, YOLOv3 is the obvious choice as it will still perform to a high standard in terms of accuracy and in real-time. Note that while there are newer versions of the YOLO algorithm, YOLOv3 provides a good balance between speed and accuracy, as well as being a relatively easier algorithm to implement due to its availability on MMDetection's Model Zoo.

Within YOLOv3, there are several different models that work with different backbones, scales, and learning rates. Due to the nature of this project, a heavyweight model taking up memory and having a large comprehensive backbone seems overkill as the project only aims to identify a single class (person) in relatively simple scenarios. In the case of real-life robots or a robot navigating in more difficult environments with several humans present, perhaps a larger model will be preferred. Therefore, the two choices for the backbone of the YOLOv3 algorithm are DarkNet-53 and MobileNetV2.

DarkNet-53 is a deep CNN (consisting of 53 convolutional layers) which is designed to be lightweight and efficient while achieving impressive performance in computer vision tasks. MobileNetv2 is a more lightweight CNN designed specifically for mobile devices (like a robot) as the architecture reduces the number of parameters and computations required while maintaining a good performance. MobileNetV2 finds a good balance between model size and accuracy, making it suitable for real-time applications on devices with limited computational resources.

Ultimately, a Darknet backbone should be used when the user is looking to a high-accuracy, robust model that can tackle complex detection tasks by maximizing access to powerful hardware like high-end GPUs and ample amounts of memory (Reddy, 2021). The MobileNetV2 backbone should be used in cases where efficiency, faster inference, and lower computational demands are key factors such as in the application of real-time mobile applications where the user is perhaps willing to sacrifice some accuracy for increased speed (Handoko, 2022). Therefore, the YOLOv3 algorithm will make use of the lightweight model

which wields the MobileNetV2 architecture as this will be more than sufficient in carrying out human-detection at a fast and accurate rate.

3.8 YOLOv3 Configuration File

Prior to training a model using the YOLOv3 algorithm, the configuration file needs to be modified so that it is compatible with the human model dataset and all its file directories, image sizes, and other details. The 'yolov3.py' file contains the code required to initiate the YOLOv3 training process. However, this code refers to the base code 'yolov3_base.py' which contains the core configuration settings for the YOLOv3 model with the MobileNetV2 backbone. This dual code structure (as seen in Appendix E1 & E2) allows for easy configuration and modification of files. This code defines the base configuration, data preprocessor configuration, and model configuration including information regarding the model backbone (MobileNetV2), neck (YOLOv3Neck), head, as well as how the training and testing configuration will look like. The algorithm makes use of loss functions during the training process as they apply Cross Entropy Loss which measures the discrepancy between predicted class scores and the ground truth class labels.

The YOLOv3 code continues as it defines the test and test pipeline, dataloader, and evaluator. In addition to the previous data augmentation carried out using roboflow, the YOLOv3 architecture also applies data augmentation in its pipeline as it loads images and annotations, increases the size of the input image while maintaining aspect ratio, randomly crops, flips, and resizes images while also applying colour and intensity distortions to the images to simulate changes in lighting conditions. These techniques helps the model generalize better to create a more robust algorithm by exposing it to various inputs in different appearances and conditions. The dataloaders are relatively straightforward as they call upon the custom human model dataset split into a training, testing, and validation set.

Finally, the model requires the specification of the number of epochs as well as configuration of the optimizer. The number of epochs can vary as a range will be used in the training process to see how the results may vary. An SGD (Stochastic Gradient Descent) optimizer is used as it aims to minimize the loss function by iteratively adjusting model parameters based on the loss gradients. The learning rate is also specified here which is a key hyperparameter in determining the step size at which the model trains, affecting how quick or slow the model converges. Typically, it is worth finding a good balance between speed and accuracy by using a middle-ground value, as a learning rate too low would result in an extremely long training process which you may not have the time nor computer capabilities for. On the other hand, a high learning rate will result in a model that is not completely optimal as it has not been given enough time to train thoroughly, causing the optimization to be unstable and overshoot/oscillate the optimal parameter values during training.

Once the configuration file has been adjusted to the user's requirements, the training process can begin using the 'train.py' code used in the MMDetection folder.

3.9 Model Training

The 'train.py' script is used to train a detector using the MMDetection framework. It handles several stages of the training process as it loads the configuration file, enables mixed-precision training, and saves epoch checkpoints. In order to carry out the training process on the coco human model dataset, the following code needs to be entered into the Linux terminal from the MMDetection folder: 'python tools/train.py configs/yolo/yolov3.py'. This command will initiate the training process as it calls upon both the training code and the configuration file.

Figure 15 below shows what the training process looks like over one epoch. Once an epoch is complete, it saves into a 'work_dirs' folder where it can then be called during the testing

process. As you can see, the epoch training process has 178 images despite the original dataset after augmentation containing just 71 images. These additional images are due to further augmentation and will significantly aid in improving the model performance. Once an epoch is completed, a checkpoint is automatically created, so that the performance of the model at each epoch can be evaluated.

Figure 15 also provides a few useful values to indicate the performance of the model such as the current learning rate which is the rate at which the model updates its weights during training. It also provides a value for the gradient norm of the model's parameters which is a measure of how much the parameters should be updated based on the calculated gradients. Finally, there are several loss values each representing different types of loss. 'loss' is the total loss value for the current batch, 'loss_cls' is the classification loss between probabilities and the ground truth, 'loss_conf' is the confidence loss which measures how well the model predicts the confidence score of an object in the anchor box, 'loss_xy' is the loss associated with the predicted centre coordinates of the bounding box, and 'loss_wh' is the loss related to the predicted width and height of the bounding boxes.

```
08/17 16:17:38 - mmengine - INFO - Epoch(train)  [1][ 50/178]  lr: 3.7056e-05  e
ta: 5:51:55  time: 2.3860  data_time: 0.0275  grad_norm: 381.8559  loss: 352.374
9  loss_cls: 21.0127  loss_conf: 314.9809  loss_xy: 12.6462  loss_wh: 3.7351
08/17 16:19:28 - mmengine - INFO - Epoch(train)  [1][100/178]  lr: 7.4561e-05  e
ta: 5:36:42  time: 2.2056  data_time: 0.0149  grad_norm: 268.6752  loss: 222.404
2  loss_cls: 20.3682  loss_conf: 187.2288  loss_xy: 12.4103  loss_wh: 2.3969
08/17 16:21:19 - mmengine - INFO - Epoch(train)  [1][150/178]  lr: 1.1207e-04  e
ta: 5:30:57  time: 2.2167  data_time: 0.0218  grad_norm: 211.2681  loss: 125.381
4  loss_cls: 19.5230  loss_conf: 91.5854  loss_xy: 12.4133  loss_wh: 1.8596
08/17 16:22:19 - mmengine - INFO - Exp name: yolov3_20230817_161533
08/17 16:22:19 - mmengine - INFO - Saving checkpoint at 1 epochs
```

Figure 15 - Training process using YOLOv3

In this project the training process is carried out over 300 epochs to see the effect this has on the results. The performance of the model is monitored at several key checkpoints of 25 ,50, 100, 150, 200, and 300 epochs to see when the model converges to its optimal performance.

3.10 Model Testing

The 'test.py' script available in MMDetection/tools loads a model checkpoint and test configuration that can evaluate the model's performance on the test dataset. The test.py code can be called and applied to any epoch saved in the checkpoints using the following example code for epoch 10 in the training set:

'python tools/test.py configs/yolo/yolov3.py work_dirs/yolo300/epoch_10.pth'

This command calls the test code to carry out the testing process and present relevant values based on the given configuration file which specifies the results it would like to present, as well as the data within the epoch checkpoint to see the model's performance. The format of the results can be seen in Figure 16 to help understand each metric.

Intersection over Union (IoU) is a common evaluation metric used in object detection as it measures the overlap between a predicted bounding box and the ground truth bounding box to quantify how accurate the predicted region is compared to the actual object (Zhao, 2020). IoU is calculated by dividing the area of the intersection between the predicted and ground truth boxes by the area of their union, resulting in an IoU value ranging from 0 to 1, where 0 indicates no overlap, and 1 indicates a perfect match between the regions. Multiple thresholds are used to determine whether a prediction is considered a True Positive, False Positive, or

False Negative. An IoU threshold of 0.5 will determine an image to have an accurate bounding box if it has at least a 0.5 overlap with the ground truth box.

mAP (Mean Average Precision) is a metric used to evaluate overall precision performance of an object detection model across different classes and IoU thresholds (Gad, 2021b). It provides a single value that summarizes the model's ability to detect the human at different levels of accuracy. To calculate the overall mAP, the AP (Average Precision) and AR (Average Recall) for all classes and IoU thresholds are averaged to provide this single value of mAP across various detection scenarios. A higher mAP value indicates a better overall detection performance, while a random model would have a value of 0.

```
Accumulating evaluation results...
DONE (t=0.00s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.341
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.730
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.191
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.00
0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.00
0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.341
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.457
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.457
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.457
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.00
0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.00
0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.457
08/17 17:00:57 - mmengine - INFO - bbox_mAP_copypaste: 0.341 0.730 0.191 -1.000
-1.000 0.341
08/17 17:00:57 - mmengine - INFO - Epoch(test) [2/2]    coco/bbox_mAP: 0.3410  c
oco/bbox_mAP_50: 0.7300  coco/bbox_mAP_75: 0.1910  coco/bbox_mAP_s: -1.0000  coc
o/bbox_mAP_m: -1.0000  coco/bbox_mAP_l: 0.3410  data_time: 0.1113  time: 0.6594
```

Figure 16 - Results from the YOLOv3 training process (Epoch 10 checkpoint)

3.11 Human Detection Algorithm

The purpose of the human detection algorithm is to identify the human 'person' within images obtained from the TurtleBot3 Waffle Pi camera sensor. The algorithm must be able to process each incoming image, detect the presence of the person, and provide their estimated position. The algorithm will operate based on the YOLOv3 deep learning model trained in the previous section.

The human detection algorithm (refer to Appendix F1) is implemented as an ROS node with the following functions that carry out their respective tasks:

1. __init__: This algorithm initializes as a ROS node, setting up necessary parameters and data structures. It creates a subscription to the camera topic, so that it is able to receive images from the robot's camera sensor. This function also creates a publisher for the 'detected_persons' topic, where it publishes a Pose Array message containing a detailed script of the persons position within the simulation. Finally, the function calls the trained YOLOv3 object detection model that will be used to identify the person.
2. Image_callback: When a new image is received on the camera, this function is called, and a message is printed into the terminal to confirm that a new image is received. The primary purpose of this function is to convert the image from the ROS message format into a format suitable for processing using the OpenCV library.

3. Draw_boxes: This function overlays bounding boxes on the original image to visually identify the detected person based on the YOLO algorithm. It iterates through the detected persons and draws a bounding box, as well as labelling the box ‘person’.
4. Detect_persons: This function takes the image as an input and uses the YOLOv3 object detection model to identify the person within the image. It processes the models output to obtain important information about the person, including their label (person), confidence scores, and bounding box coordinates.
5. Create_pose_array: The function converts the information from the detect_persons function into a Pose Array message, where the detected person is assumed to be at the centre of their bounding box. The publish pose array function then publishes the data on to the detect_persons topic.
6. Main: Finally, the main function initializes the ROS node, and enters a spin loop to process incoming images and provide a live visual output of the human with the bounding box around it.

While this code runs fine and carries out its function, the draw_boxes function can also be modified for reasons explained in Section 4.2. The modified function can be seen in Appendix F2 as it essentially makes it so that a bounding box is only drawn around the detected person if there is a confidence value greater than 0.1. This improves the reliability of the robot as it can filter out unreliable detections which are likely to be false positives or incorrect detections which could be a result of noise or occlusion that could confuse the robot and lead to erratic behaviours. The new function also sets the ‘person_detected’ flag to ‘True’ if at least one confidence score is above 0.1 so the robot will only act and respond to the person if it is reasonably certain that there is actually a person in its field of view.

This node acts as the initial step in the human tracking process, as the information collected here is crucial for the subsequent human tracking algorithm, which makes use of the detected persons position to track and interact with the person.

3.12 Human Tracking Algorithm

The purpose of the human tracking algorithm is to implement a basic human-following algorithm for the robot. In this case, the robot subscribes to the ‘detected_persons’ topic created in the detection algorithm previously, providing important information on the position of the person in the simulation environment. When a person is detected, along with their position, the human tracking algorithm calculates and publishes control commands to direct the robot to follow the person.

A simple human tracking algorithm (refer to Appendix G1) is created to ensure that proper communication can be carried out between the two algorithms in order to allow the robot to accurately detect and follow the human using Scenario 1 (refer to Section 3.4) where the human walks forward in a straight line. The functions within the node can be explained as follows:

1. __init__: This is the constructor method for the node which initializes the node and sets up the necessary publishers and subscribers. Specifically, it creates a subscription to the ‘detected_persons’ topic which is where positional information of the human is received. It also creates a publisher for the control commands to the ‘cmd_vel’ topic which is used to control the robot’s movements.
2. Pose_array_callback: This function is a callback that is executed whenever a Pose Array message is received on the ‘detected_persons’ topic. The logic of the code works so that if a detected pose message is received, it proceeds to the control command. A fixed linear velocity is set to make the robot move forward at that speed, and an angular

velocity is set to zero for now which means the robot does not turn at all while following. Finally, the twist message is used to publish the control commands.

3. Main: The main function is used to initialize the rclpy framework and create an instance of the node. The node starts spinning and continues to run until keyboard interrupted.

While this is a very simple tracking logic with a fixed linear velocity and no angular velocity, it is useful in creating the groundworks of the algorithm and ensure that both the detection and tracking nodes can communicate with each other to control the robot.

Many different tracking logics can be applied to the robot to ensure it follows the human efficiently regardless of its route. A more sophisticated algorithm which takes in a larger range of considerations for smooth robot operation is created (refer to Appendix G2). While the '__init__' and 'main' functions will stay relatively similar in 'ht1.py', the tracking logic is optimised in the 'pose_array_callback' and new 'follow_human' functions allowing the human to detect a human at various distances and vary the linear velocity of the robot based on its distance from the human so that it is always within range of the human and does not get too far or too close. 'ht2.py' (refer to Appendix G3) implements another interesting logic as it makes use of two separate fixed linear velocities. If the robot is above 0.5m away from the human, it travels at max speed, else if the robot is too close it halts completely. However, this will result in a lot of stop and start motions of the robot, making the previous variable linear speed a smoother interaction.

Appendix G4 shows the 'follow_human' function for 'ht3.py' which is the most sophisticated code as it accounts for all human-robot interactions. Similar to 'ht1.py' it applies a variable linear and angular velocity based on distance so that the robot is always centred behind the human and is never too close or too far from the human. Additionally, it provides a safety logic that if the robot is too close to the human or the human is not visible then the robot will stop moving.

With the creation of the human tracking algorithm, the setup for the project is complete, where the results and performance of the robot as well as seeing if the project runs as expected is carried out in Section 4 ahead.

4 Results & Evaluation

4.1 YOLOv3 Analysis

By following the methodology through Sections 3.8 – 3.10, the following results are produced from the trained YOLOv3 model.

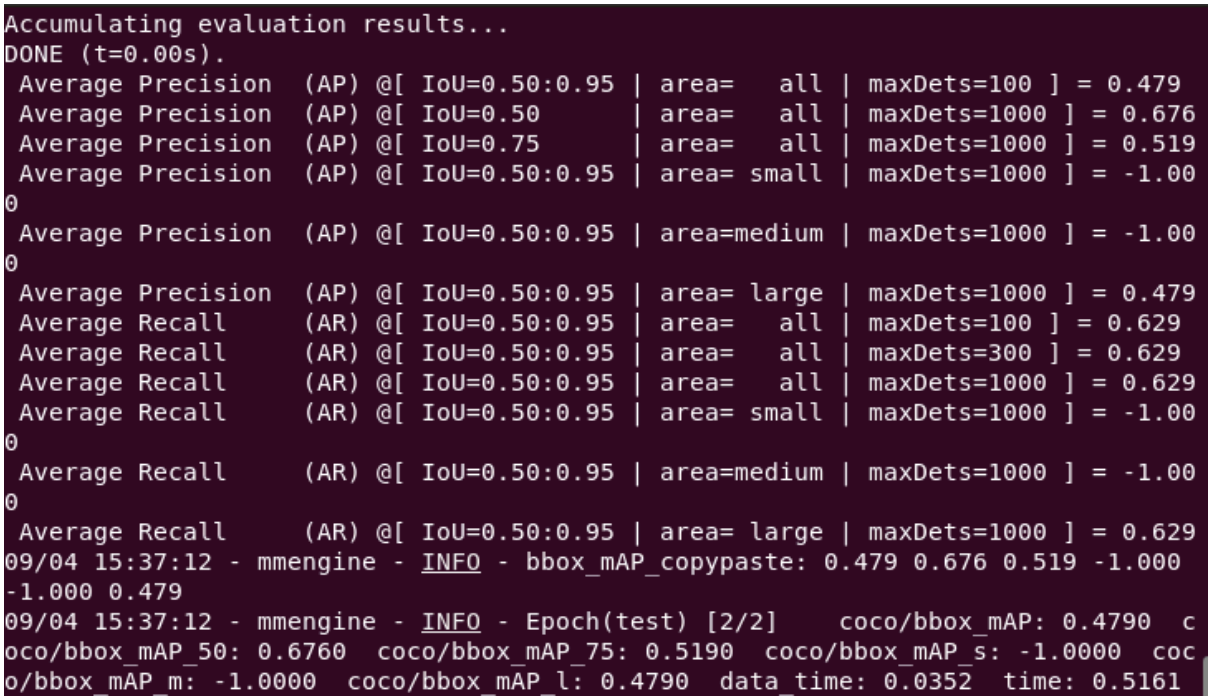


Figure 17 - Epoch 25 Result

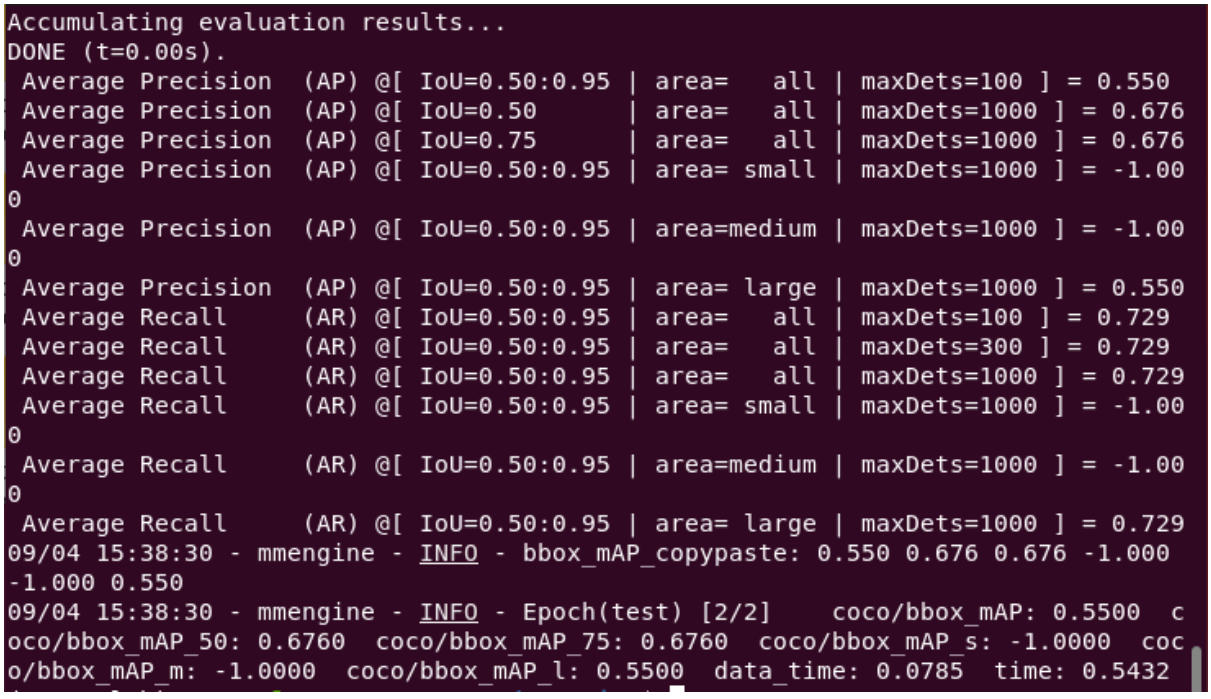


Figure 18 - Epoch 50 Result

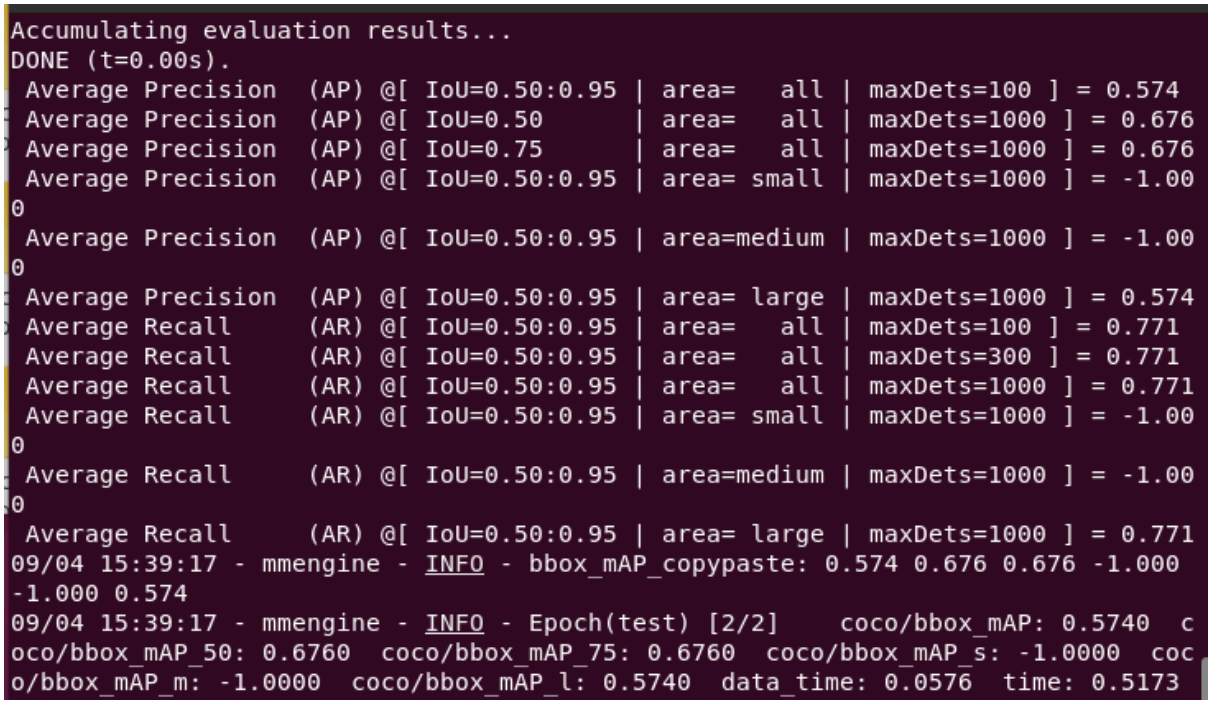


Figure 19 - Epoch 100 Result

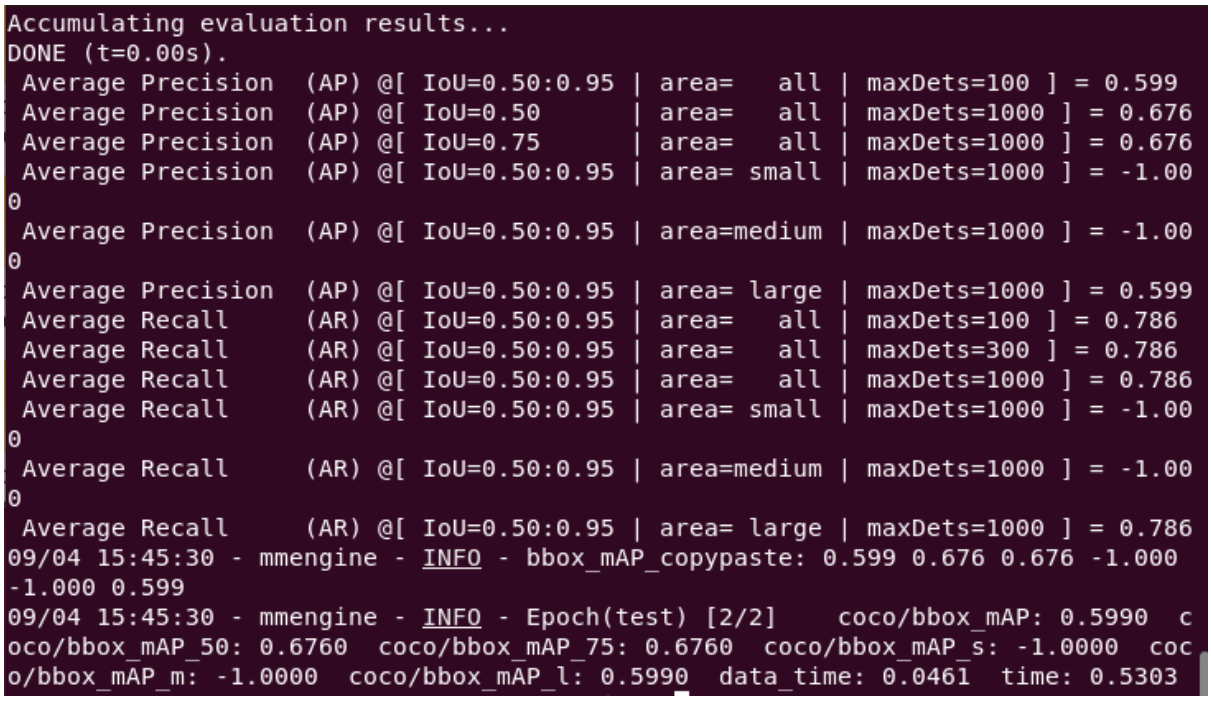


Figure 20 - Epoch 150 Result

```
Accumulating evaluation results...
DONE (t=0.00s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.596
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.676
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.676
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.00
0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.00
0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.596
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.771
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.771
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.771
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.00
0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.00
0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.771
09/04 15:46:43 - mmengine - INFO - bbox_mAP_copypaste: 0.596 0.676 0.676 -1.000
-1.000 0.596
09/04 15:46:43 - mmengine - INFO - Epoch(test) [2/2]    coco/bbox_mAP: 0.5960  c
oco/bbox_mAP_50: 0.6760  coco/bbox_mAP_75: 0.6760  coco/bbox_mAP_s: -1.0000  coc
o/bbox_mAP_m: -1.0000  coco/bbox_mAP_l: 0.5960  data_time: 0.0577  time: 0.5254
```

Figure 21 - Epoch 200 Result

Figures 17-21 above show the performance results of the YOLOv3 model at 5 different checkpoints. By monitoring the progress, you can see where the model converges, after which the performance of the model deteriorates.

At 25 epochs, the model reaches a low mAP of 0.479 for the IoU threshold 0.5:0.95. This is to be expected as an insufficient number of epochs will not allow the model to converge. At the next checkpoint (epoch 50), the AP at IoU thresholds of 0.5 and 0.75 reach its maximum potential of 0.676. Any further increase in epoch numbers does not increase these AP values. There is also a significant increase in mAP as it is now 0.55 due to an increase in AR.

While epoch 50 provides a good model and could be sufficient to implement into the robot, an increase to 100 epochs shows a further increase in the performance of the model. The mAP at IoU 0.5:0.95 increases to 0.574, resulting in better performance over a range of IoU thresholds despite the AP at IoU thresholds of 0.5 and 0.75 remaining the same (at max value). This is because the AR value over different thresholds is continuously increasing, showing that the models detection capability is increasing as the model trains for a longer period. While the AP values have converged to an optimal level, the average recall over different thresholds seems to continuously increase, indicating an increase in overall model performance as seen by the increase in mAP too. In order to maximize the model's performance, the model is run until a drop is seen in AR, resulting in a drop of the mAP value.

Figure 20 shows the performance of the YOLOv3 model at 150 epochs. The AP value at IoU thresholds of 0.5 and 0.75 remain the same (at 0.676), however the AR values increase up to 0.786. This results in the maximum performing model with a mAP value of 0.599, making it the optimal checkpoint for the model to operate at when being implemented into the robot's algorithm. This is because any further increase beyond this epoch results in a decrease in performance as seen in Figure 21 which runs the model at 200 epochs as the mAP decreases to 0.596, and the AR value decrease to 0.771. This suggests that the model have converged to its optimal performance around Epoch 150 after which model performance declines. Hence, it is not worth even testing Epoch 300 or more as the model will inevitably get worse due to overtraining and less generalization.

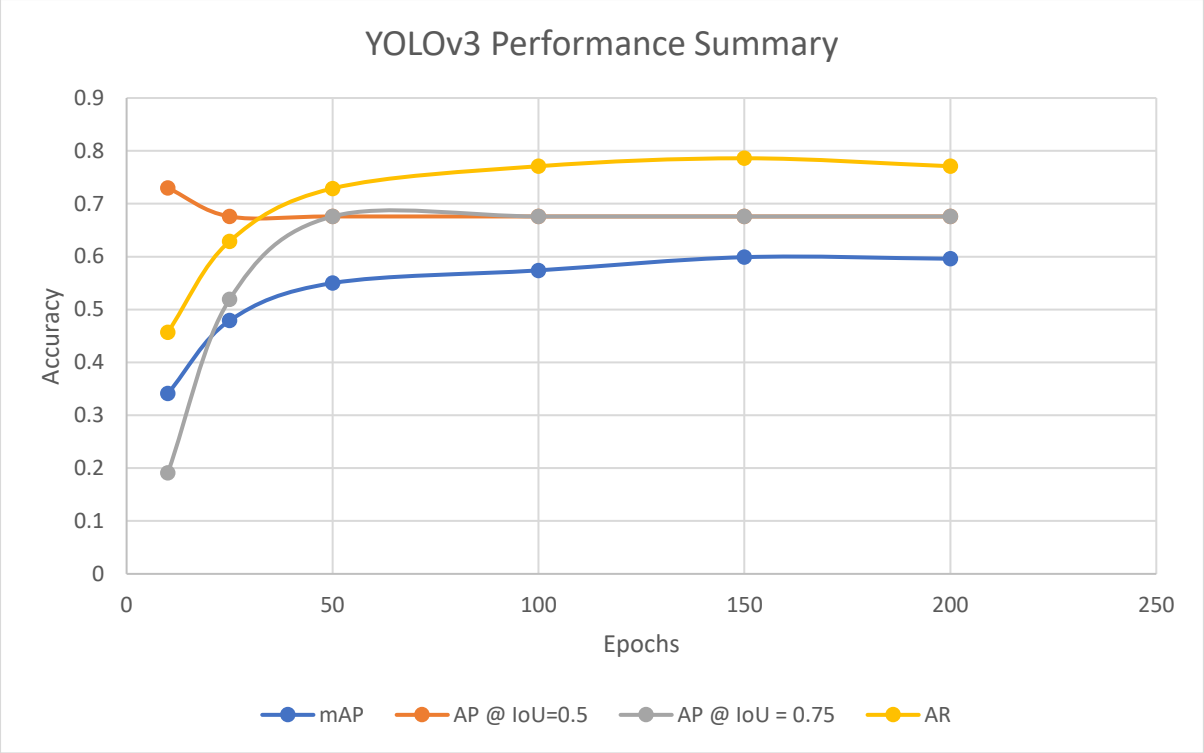


Figure 22 – Summary of YOLOv3 Results

Figure 22 shows the summary of the results through the different epoch checkpoints, as the AP at IoU values of 0.5 and 0.75 converge to 0.676, the AR value (0.786) continuously increases with a peak at 150 epochs, after which is drops, and the mAP following suit with a max value (0.599) at epoch 150 after which it drops slightly (0.596).

To conclude, the optimal epoch number for the YOLOv3 model is 150 epochs, at which the model can best detect a human over a range of different IoU thresholds. While several other hyperparameters can be modified and optimised, the performance of the model (approximately a mAP of 60%) is sufficient for this project. A further increase in epochs in addition to other hyperparameter tests like changing the optimizer or decreasing the learning rate will result in a longer training process and require more time and a greater computational demand, both of which are not necessary in this case.

4.2 Human Detection Algorithm Analysis

The human detection algorithm discussed in Section 3.11 applies the trained YOLOv3 150 Epoch checkpoint to the Waffle Pi robot, allowing for human detection within the Gazebo world. To test the performance of the human detection algorithm, scenario 2 (refer for Section 3.4) is used. Since, the robot will not be moving for this test, the performance of the robot can be judged for different human angles and distances. By running the Gazebo world 'walk_back.world' as well as running the human detection algorithm 'hd.py', the test begins.

At the start of the test, the human is extremely close to the robot, meaning the robot can only see parts of the human's legs, making it slightly difficult to identify the human. This is shown since there is a larger bounding box around the human due to a lower confidence score. While this is still sufficient and the algorithm performs incredibly well, 'hd_conf.py' is used as the human detection algorithm as it will only detect the human if the confidence score is greater than 10% increasing its reliability. Once this model runs, the human can be identified more confidently even at close proximities.

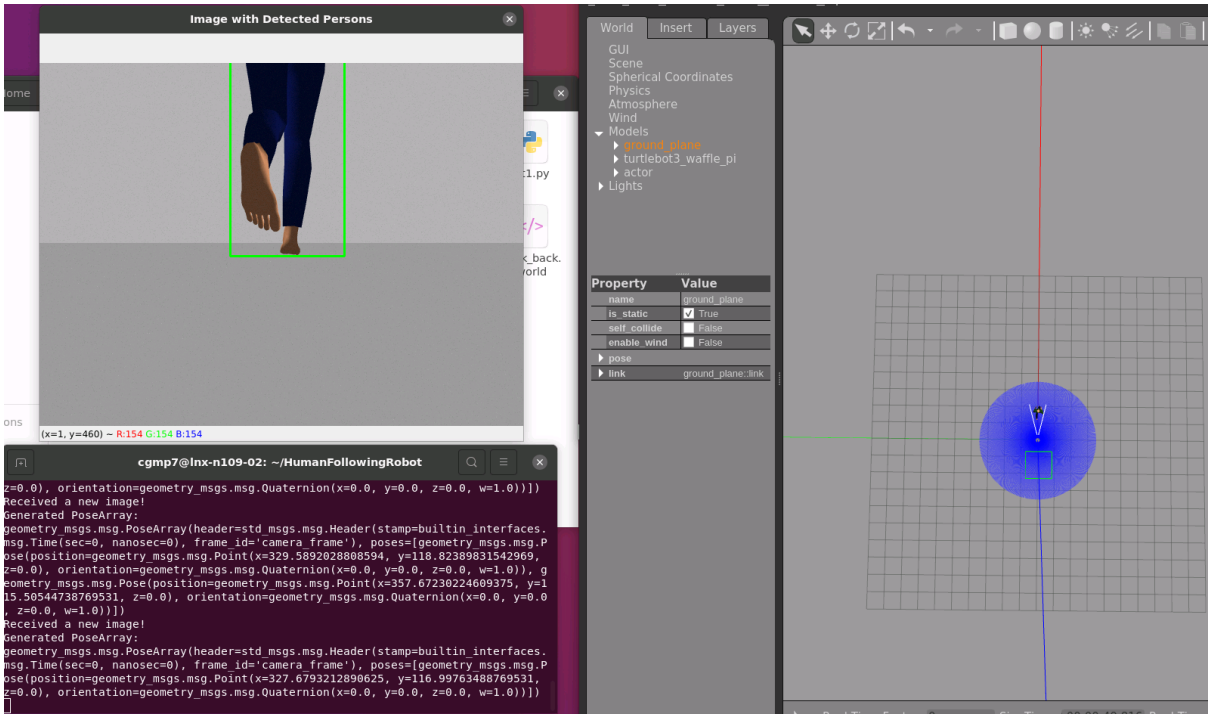


Figure 23 - Human Model at close proximity

Once the human is approximately 0.5 metres ahead of the robot, its camera is able to accurately identify the human model until it is greater than approximately 8-10 metres ahead of the robot after which the human can no longer be identified (as seen in Figures 24 & 25). Hence, it is fair to conclude that the robot performs best when it is within 0.5-10 metres range of the human at all times.

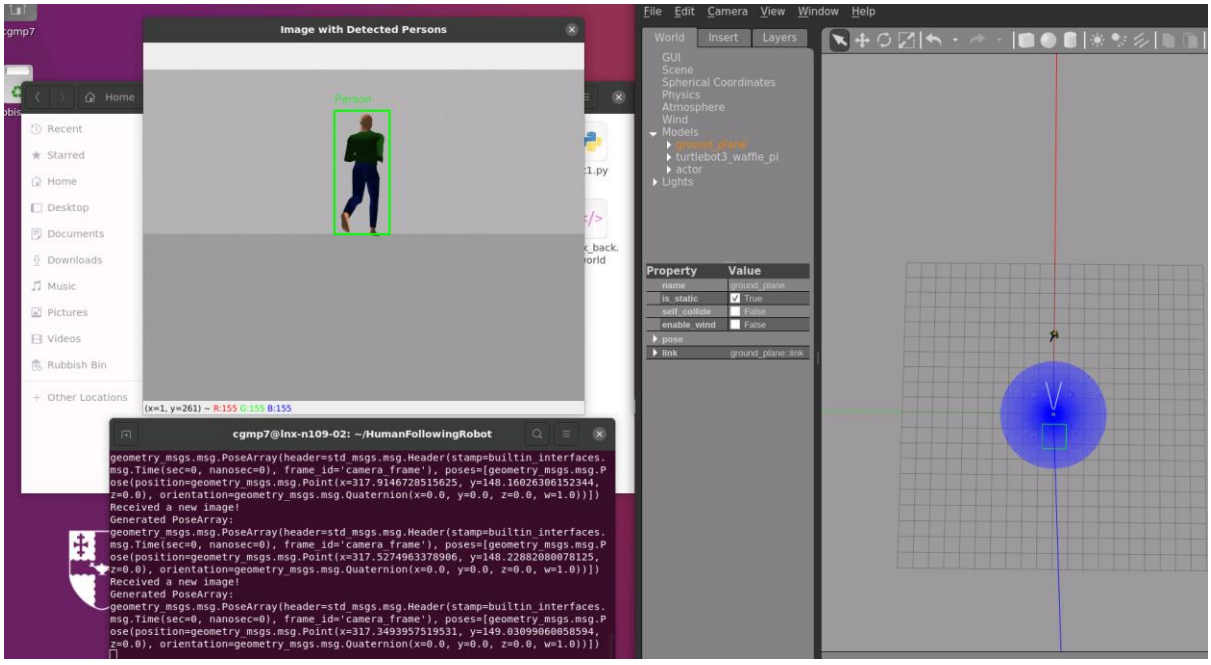


Figure 24 - Person in 0.5-10 metre distance from the robot

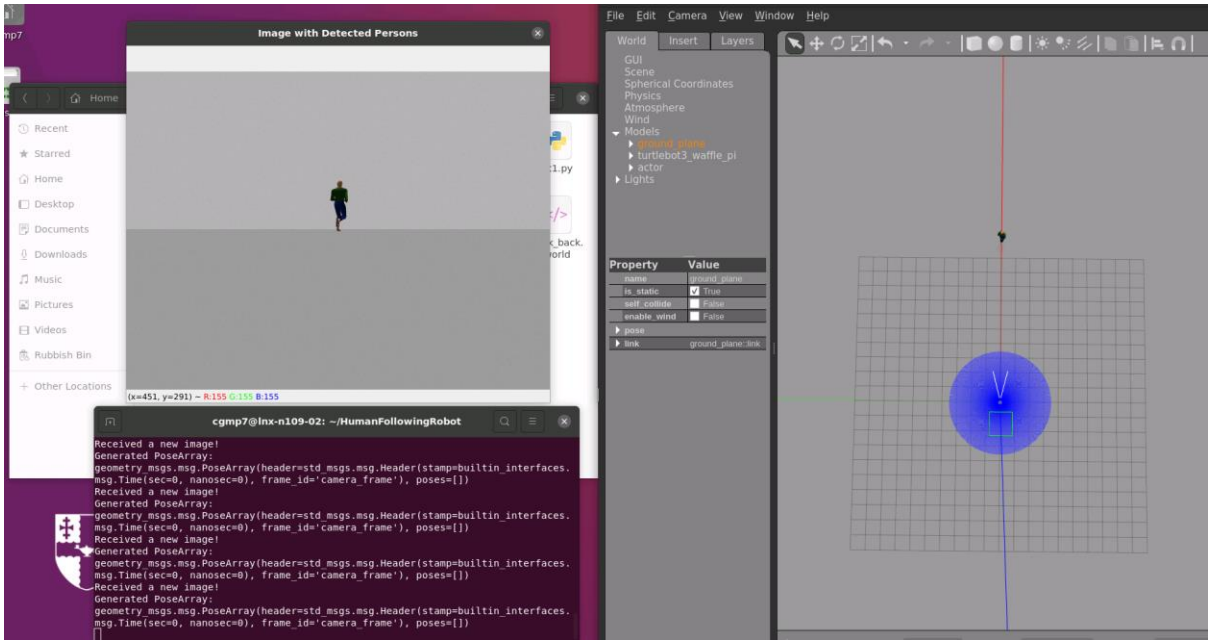


Figure 25 - Human model 10+ metres from the robot (No bounding box)

Once the human turns around and returns to the start point, the robot is now able to identify the human (at all distances) based on its features from the front rather than the back. This shows that the robot is able to accurately identify the human from a range of different distances and angles as seen in Figure 26.

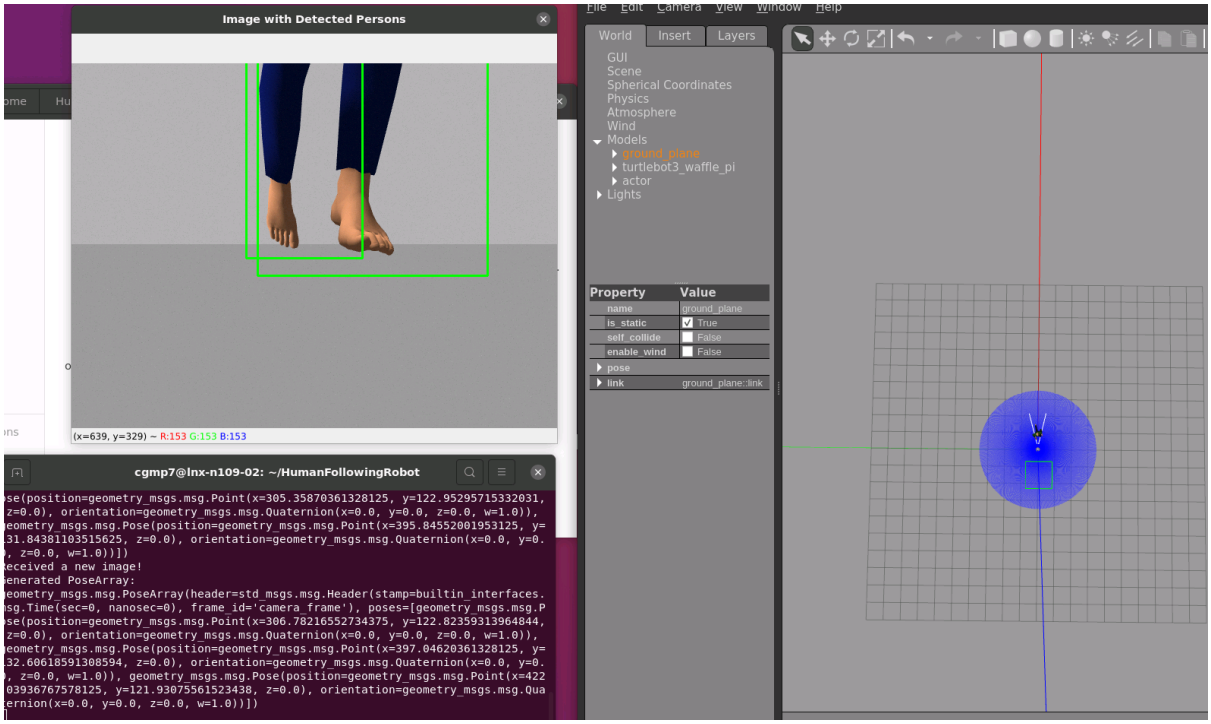


Figure 26 - Human model from the front (with bounding boxes)

To summarise, this test is largely a success and confirms that the human detection algorithm produced is a comprehensive design that is able to apply the YOLOv3 model and accurately identify the human model within the virtual environment.

4.3 Human Tracking Algorithm Analysis

The human tracking algorithms discussed in Section 3.12 are used to implement a robot-following logic onto the Waffle Pi so that it can follow the detected person based on the previous human detection algorithm. To ensure that both algorithms can interact, the Gazebo world for Scenario 1 (walk_straight.world) is loaded. After loading this world in, running ‘hd.py’ and running ‘ht0.py’, the test can proceed.

Figures 27 and 28 show snippets of what happens when the human tracking algorithm is applied to the simulation. Once the robot has detected the person, its control commands activate in order to move forward directly behind the human. This works perfectly fine, although the robot seems to veer to the left after 8-10 metres as seen in Figure 27, likely due to no angular velocity application that will stabilise and centre the robot behind the person.

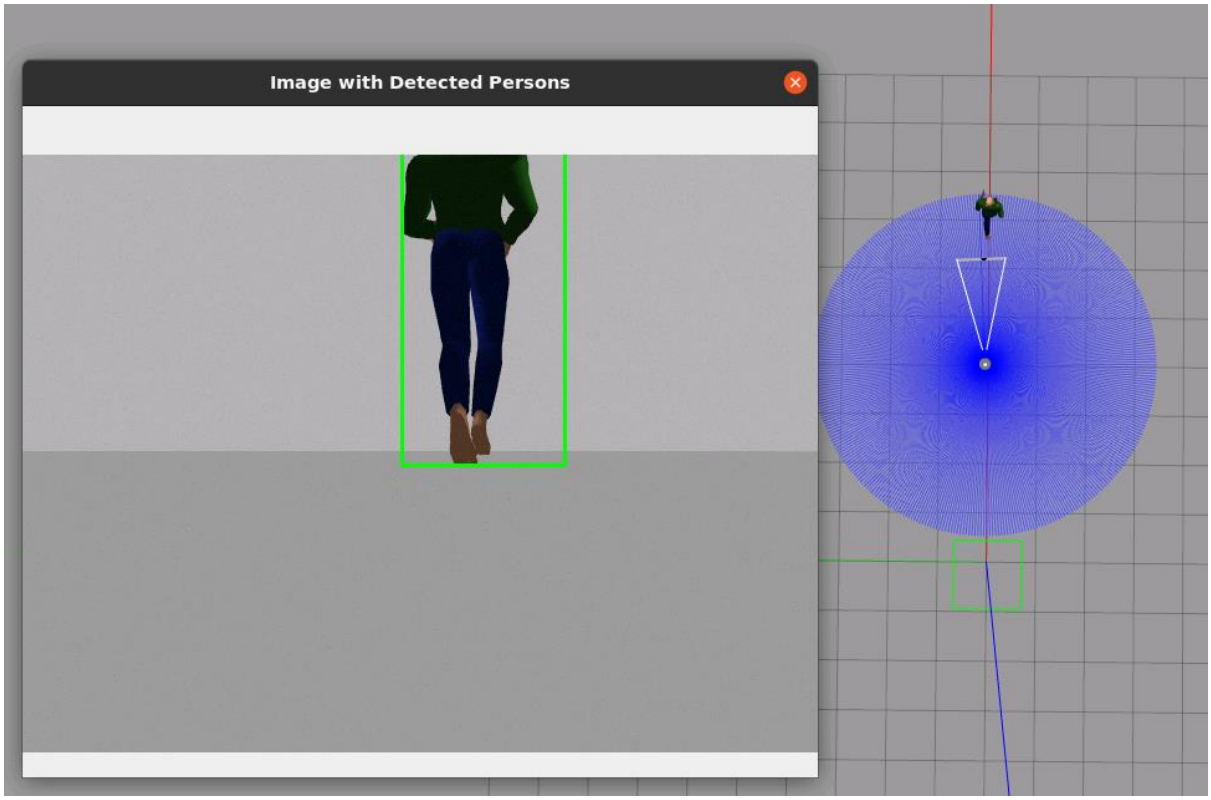


Figure 27 - Human-following robot

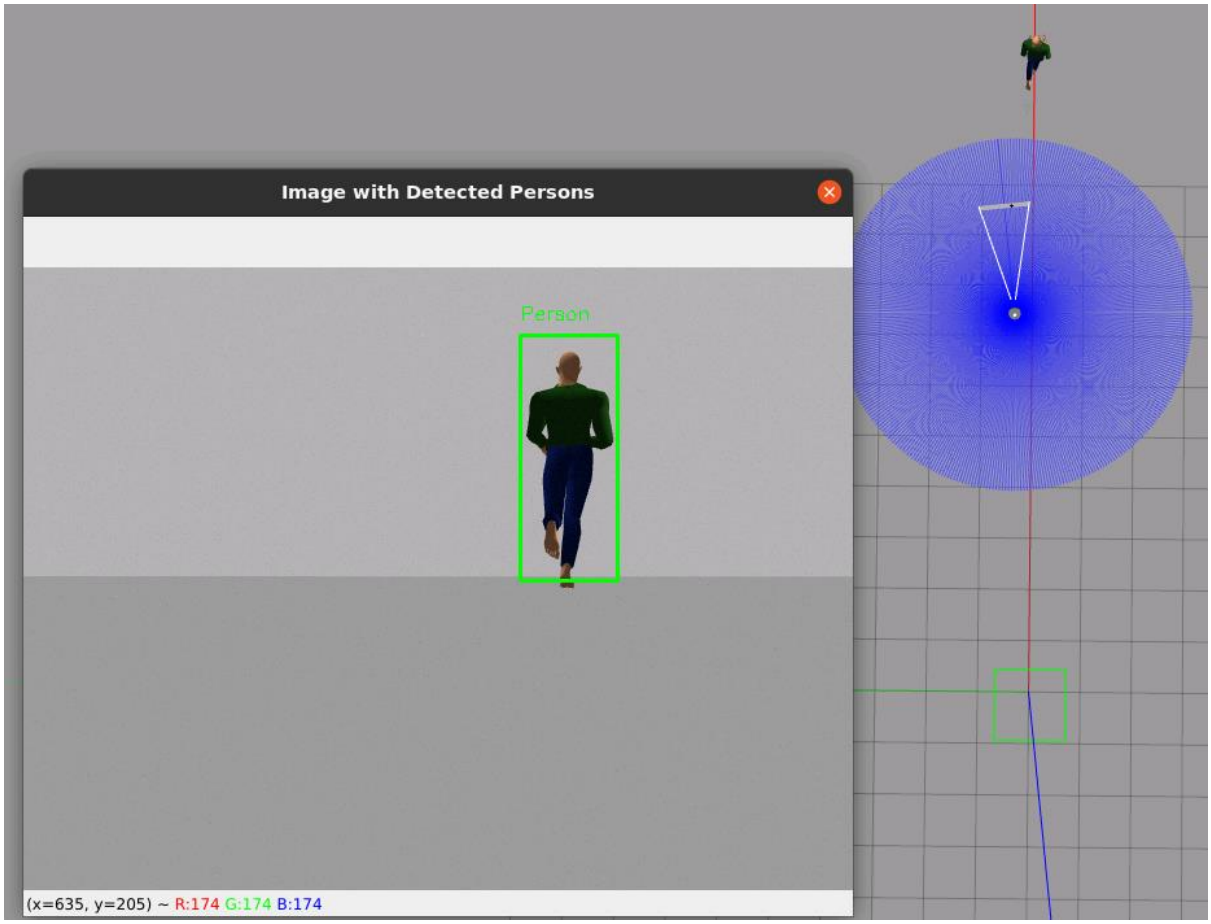


Figure 28 - Robot veering to the left

The human-following robot is operational when tested in Scenario 1 as it is able to accurately follow behind the human as it walks in a straight line for up to 10-15 metres. Since Scenarios 3 and 4 are scenarios where the human changes directions and walks around obstacles, an attempt to use a more sophisticated tracking algorithm (such as ht3.py) is made as this has directions for angular velocity and turning the robot.

Figures 29 & 30 below show the application of 'ht3.py' in Scenarios 3 (walk_rect.world) & 4 (walk_obstacles.world). It clearly shows the robot is completely unresponsive and unable to move as it stays static at the starting point. This suggests that there are several errors with the human tracking algorithm that need to be addressed in the future to create a more robust and operating tracking algorithm that operates. Many tests have been carried out to ensure that the message types (Pose Array, Twist, Bool) work correctly, to ensure that 'detected_persons' is subscribed to successfully, as well as ensuring 'cmd_vel' is published properly. This means that the reason the robot is unresponsive must come down to the 'follow_human' function and the logic implemented. Firstly, the method used to calculate distance between the robot and the human may not be correct, resulting in a null desired linear velocity value. The code extracts the humans coordinates from the Pose Array message; however, these may not be properly translated into the algorithm. Additionally, any implementation of angular velocity also seems to either cause the robot to be stationary, or just travel in a circle indefinitely. To solve this, the algorithm needs to be modified such that the angular velocity applies situationally only when necessary. Figure 30 also shows that the robot is likely overwhelmed from the 3 different boxes of different colours as it creates bounding boxes around them suggesting it has identified them as people. This suggests there are limitations in the YOLOv3 model that must be addressed in the future.

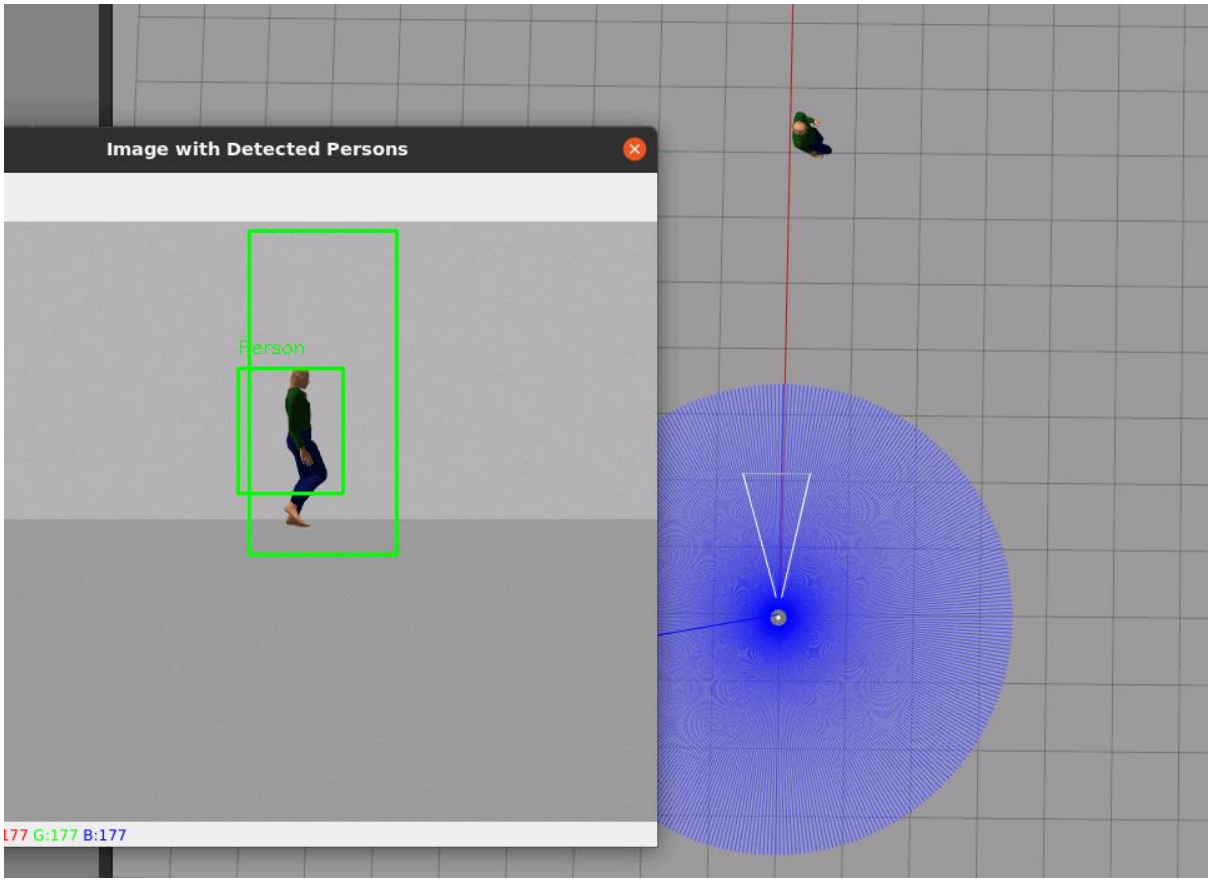


Figure 29 - Scenario 3 with ht3.py

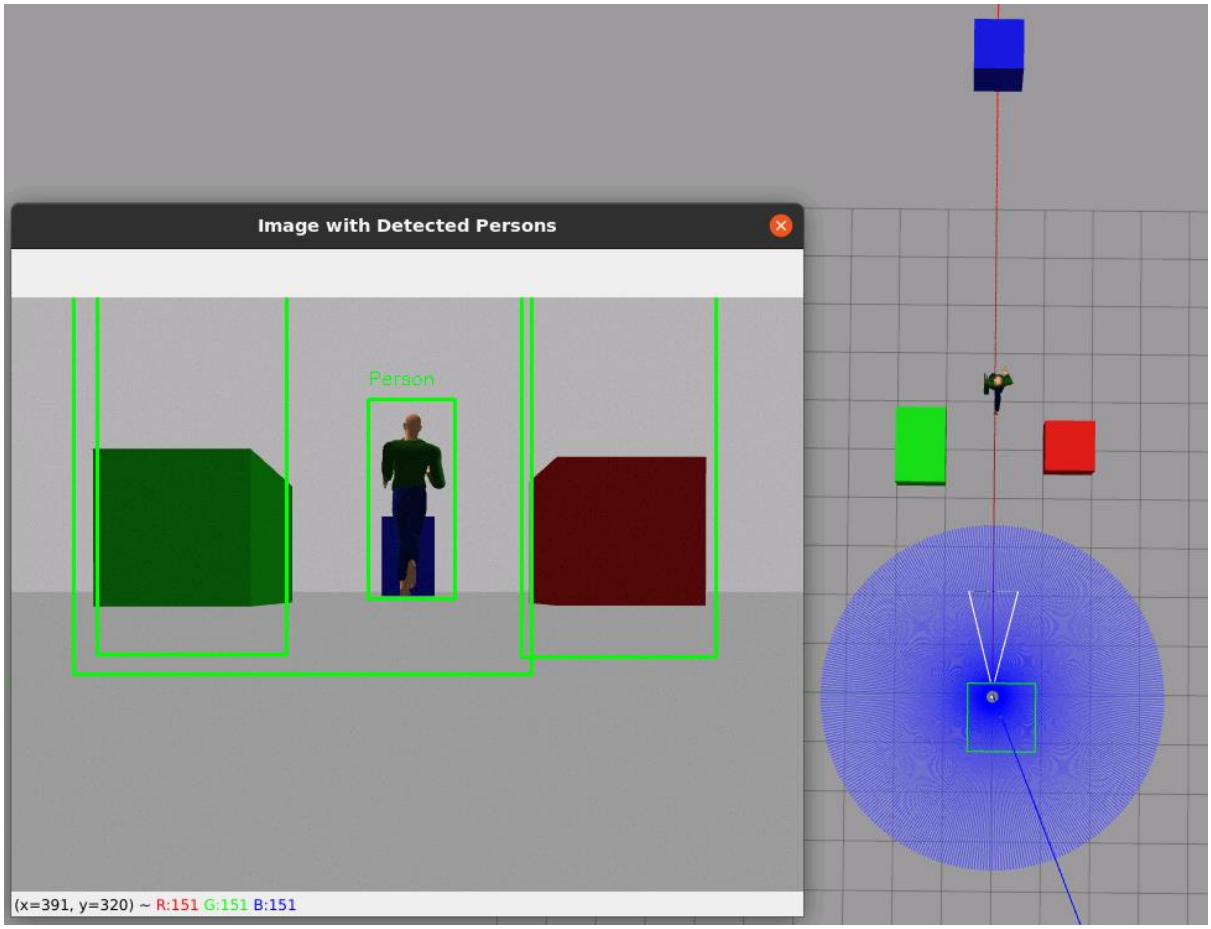


Figure 30 - Scenario 4 with ht3.py

In summary, the human-following robot was a success when applying a simple human tracking algorithm (ht0.py) which is able to follow a human in a straight line (and no angular velocity). However, the algorithm (ht3.py) caused the robot to fail in following the human in a rectangle (where there are changes in trajectory) or around obstacles. Reasons for this error are discussed and suggestions have been given as to why this error is likely occurring as well as what needs to be done to create a better robot. Overall, the project can be considered a partial success as a detailed methodology in developing a human-following robot is discussed, and a range of detection and tracking algorithms are implemented into different simulation environments. The current algorithms can be built upon in order to create a more flexible and smarter robot that can follow the human in more difficult environments.

4.4 Limitations & Challenges

Based on the methodology implemented as well as the results discussed, there are a range of limitations encountered which prevented the creation of a perfect human-following robot, based on which several suggestions for future development can be made.

Firstly, collecting a larger dataset of the human images from a wider range of angles and distances can increase the performance of the robot. This should also include a greater range of data augmentation such as flipping images or changing the contrast of colours as both of these can lead to creating a more robust model with increased generalization.

Beyond this, the YOLOv3 model is a sub-par YOLO model with new and improved algorithms existing such as YOLOv8 which is superior in every way due to increased speed and accuracy of the model. Applying better algorithms will result in a better robot. In addition to an improved

YOLO algorithm being used, a more thorough investigation into hyperparameter tuning is also ideal in order to optimize the algorithms performance by testing a range of parameters, optimizers, learning rates, and loss functions. This can be quite time consuming and computationally demanding, hence why a trade-off between computational demand and accuracy is typically met based on the users limits in time and computational capacity.

Thirdly, a lot of difficulties were encountered in perfecting the human tracking algorithm as discussed in Section 4.3 due to the algorithm being unable to properly follow the human when it would turn directions or move at differing speeds. Implementing an algorithm logic that can modify the robots linear and angular velocity to move based on the position of the human and its movements, as well as creating an improved robot logic that applies different decisions based on the data it receives will result in a much better performance of the robot. In collaboration with the existing human detection and tracking algorithms, an obstacle avoidance algorithm should also be implemented so that the robot has an increased sense of safety in the environment rather than blindly trusting the human.

Finally, an attempt into applying a reinforcement learning algorithm in order to train a robot based on reward learning methods to iteratively improve the performance of the robot is a highly interesting way in optimizing the performance of the robot, hence making it worth looking into after this project.

5 Conclusion & Future Directions

To conclude, this project successfully researched into virtual human-following robots as it is a staple concept that can be developed and implemented into more complex technologies. A discussion is carried out into the importance of human-following robots as well as HRC, as relevant literature is referenced to track the progress of this technology. These concepts are linked to virtual robot development as this is a safe and cost-effective manner to optimize robots. The methodology gives a detailed description of how to create a human-following robot in a virtual environment by making use of ROS, Gazebo, and Python, as well as data pre-processing, and application of deep learning algorithms such as the real-time object detection algorithm YOLO.

Overall, this project was mostly successful as each stage of the methodology was carried out optimally to create a virtual human-following robot with a 60% detection accuracy, and a safety logic so that the robot does not collide with the human. Despite this, the performance of the robot was not perfect, largely due to a range of limitations and difficulties in the human tracking algorithm that should be investigated in order to perfect this project in the future. In terms of project-specific suggestions that should be investigated in the future, refer to Section 4.4 as it mentions increasing the size of the dataset, improving the human-tracking algorithm logic, implementing an obstacle avoidance algorithm, and applying RL. Implementing these suggestions could result in a more robust human-following robot that can operate better in a range of different challenging environments.

Finally, refer to Section 2.7 as it provides a detailed account into more general challenges and future directions of human-following robots and HRC as an industry, as detail is given into each aspect of simulation in robotics and what needs to be addressed to progress both technically and ethically in the field of robotics. Addressing and overcoming these challenges through further research and development of HRC and AI through ethical means, could result in a world where robots and humans safely, and efficiently interact within close proximity to carry out tasks in a seamless manner, over a range of domains and human-centric environments (Morioka, 2004).

6 References

1. Agrawal, P., Jain, G., Shukla, S., Gupta, S., Kothari, D., Jain, R., & Malviya, N. (2022, July). YOLO Algorithm Implementation for Real Time Object Detection and Tracking. In *2022 IEEE Students Conference on Engineering and Systems (SCES)* (pp. 01-06). IEEE.
2. Akihiko. (2018). *Ros: A minimum overview*. text/ROS/MinOverview - Akihiko's Tech Note. http://akihikoy.net/notes/?text%2FROS%2FMinOverview
3. Amsters, R., & Slaets, P. (2020). Turtlebot 3 as a robotics education platform. In *Robotics in Education: Current Research and Innovations 10* (pp. 170-181). Springer International Publishing.
4. Ananth, S. (2019, August 30). *Fast R-CNN for object detection*. Medium. https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022
5. Anderson, S. L. (2008). Asimov's "three laws of robotics" and machine metaethics. *Ai & Society*, *22*, 477-493.
6. Builtin. (2023) *Robotics.* Robotics: What Are Robots? Robotics Definition & Uses. | Built In. https://builtin.com/robotics
7. Capurro, R., Nagenborg, M., & Tamburinni, G. (2009). Ethics and robotics. *Heidelberg/Amsterdam*.
8. Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., ... & Lin, D. (2019). MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*.
9. Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., ... & Trinkle, J. (2021). On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, *118*(1), e1907856118.
10. Chun, S., Lee, C. S., & Jang, J. S. (2015). Real-time smart lighting control using human motion tracking from depth camera. Journal of Real-Time Image Processing, 10, 805-820.
11. Constructism. (2022) *A history of ROS (robot operating system)*. The Construct. (2022, November 18). https://www.theconstructsim.com/history-ros/
12. Constructism. (2019) *What is ROS?*. The Construct. (2022). https://www.theconstructsim.com/what-is-ros/
13. Ed. (2023, February 1). *What is Ros? - the robotics back*. End. https://roboticsbackend.com/what-is-ros/#ROS_core_and_communication_tools
14. Erős, E., Dahl, M., Bengtsson, K., Hanna, A. and Falkman, P., 2019. A ROS2 based communication architecture for control in collaborative and intelligent automation systems. *Procedia Manufacturing*, *38*, pp.349-357.
15. Fang, W., Wang, L., & Ren, P. (2019). Tinier-YOLO: A real-time object detection method for constrained environments. *IEEE Access*, *8*, 1935-1944.
16. Farley, A., Wang, J., & Marshall, J. A. (2022). How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion. *Simulation Modelling Practice and Theory*, *120*, 102629.
17. Gad, A. F. (2021, April 9). *Faster R-CNN explained for Object Detection Tasks*. Paperspace Blog. https://blog.paperspace.com/faster-r-cnn-explained-object-detection/
18. Gad, A. F. (2021b, April 9). *Mean average precision (MAP) explained*. Paperspace Blog. https://blog.paperspace.com/mean-average-precision/
19. GadgetsLatest. (2016), Gadgetslatest, Gadgetsgear, Entertainmentgadgets, & Gadgets. (2016, November 17). *Meet your domestic servant of the future –*

apriattenda. CoolThings.com | Cool Gadgets, Gifts & Stuff |. https://www.coolthings.com/meet-your-domestic-servant-of-the-future-apriattenda/
20. GitHub. (2014). https://github.com/arpg/Gazebo/tree/master/media/models
21. Goodrich, M. A., & Schultz, A. C. (2008). Human–robot interaction: a survey. *Foundations and Trends® in Human–Computer Interaction*, *1*(3), 203-275.
22. Handoko, A. B., Putra, V. C., Setyawan, I., Utomo, D., Lee, J., & Timotius, I. K. (2022, October). Evaluation of YOLO-X and MobileNetV2 as Face Mask Detection Algorithms. In *2022 IEEE Industrial Electronics and Applications Conference (IEACon)* (pp. 105-110). IEEE.
23. He, L., Glogowski, P., Lemmerz, K., Kuhlenkötter, B., & Zhang, W. (2020, April). Method to integrate human simulation into gazebo for human-robot collaboration. In IOP Conference Series: Materials Science and Engineering (Vol. 825, No. 1, p. 012006). IOP Publishing.
24. Hutabarat, D., Rivai, M., Purwanto, D., & Hutomo, H. (2019, July). Lidar-based obstacle avoidance for the autonomous mobile robot. In *2019 12th International Conference on Information & Communication Technology and System (ICTS)* (pp. 197-202). IEEE.
25. Ilievski, A., Zdraveski, V., & Gusev, M. (2018, November). How CUDA powers the machine learning revolution. In *2018 26th Telecommunications Forum (TELFOR)* (pp. 420-425). IEEE.
26. Inkulu, A. K., Bahubalendruni, M. R., Dara, A., & SankaranarayanaSamy, K. J. I. R. (2021). Challenges and opportunities in human robot collaboration context of Industry 4.0-a state of the art review. *Industrial Robot: the international journal of robotics research and application*, *49*(2), 226-239.
27. Katz, D. (2016) Development of algorithms for a human-following robot equipped with ... (n.d.). https://in.bgu.ac.il/en/robotics/thesis/DrorKatz.pdf
28. Keita, Z. (2022, September 28). *Yolo Object Detection explained: A beginner's guide*. DataCamp. https://www.datacamp.com/blog/yolo-object-detection-explained
29. Koenig, N., & Howard, A. (2004, September). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)* (Vol. 3, pp. 2149-2154). IEEE.
30. Kononov, K., Lavrenov, R., Tsoy, T., Martínez-García, E. A., & Magid, E. (2021, October). Virtual experiments on mobile robot localization with external smart RGB-D camera using ROS. In *2021 IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE)* (pp. 656-659). IEEE.
31. Lin, L. J. (1992). Reinforcement learning for robots using neural networks. Carnegie Mellon University.
32. Maity, M., Banerjee, S., & Chaudhuri, S. S. (2021, April). Faster r-cnn and yolo based vehicle detection: A survey. In *2021 5th international conference on computing methodologies and communication (ICCMC)* (pp. 1442-1447). IEEE.
33. Maruyama, Y., Kato, S. and Azumi, T., 2016, October. Exploring the performance of ROS2. In *Proceedings of the 13th International Conference on Embedded Software* (pp. 1-10).
34. Masurekar, O., Jadhav, O., Kulkarni, P., & Patil, S. (2020). Real time object detection using YOLOv3. *International Research Journal of Engineering and Technology (IRJET)*, *7*(03), 3764-3768.
35. MMDetection 3.1.0 documentation. (2023). https://mmdetection.readthedocs.io/en/latest/overview.html

36. Morioka, K., Lee, J. H., & Hashimoto, H. (2004). Human-following mobile robot in a distributed intelligent sensor network. *IEEE Transactions on Industrial Electronics*, *51*(1), 229-237.
37. Mustamo, P. (2018). Object detection in sports: TensorFlow Object Detection API case study (Bachelor's thesis, P. Mustamo).
38. Platt, J., & Ricks, K. (2022). Comparative Analysis of ROS-Unity3D and ROS-Gazebo for Mobile Ground Robot Simulation. *Journal of Intelligent & Robotic Systems*, *106*(4), 80.
39. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).
40. Rath. (2022) Image and Video Inference using MMDetection. https://debuggercafe.com/image-and-video-inference-using-mmdetection/
41. Reddy, B. K., Bano, S., Reddy, G. G., Kommineni, R., & Reddy, P. Y. (2021, January). Convolutional network based animal recognition using YOLO and Darknet. In *2021 6th international conference on inventive computation technologies (ICICT)* (pp. 1198-1203). IEEE.
42. Rekleitis, I., Meger, D., & Dudek, G. (2006). Simultaneous planning, localization, and mapping in a camera sensor network. *Robotics and Autonomous Systems*, *54*(11), 921-932.
43. Roboflow. (2023). https://roboflow.com/
44. Robotis. (2023). *Robotis e*. Manual. https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
45. Richert, A., Shehadeh, M.A., Müller, S.L., Schröder, S. and Jeschke, S., 2016, July. Socializing with robots: Human-robot interactions within a virtual environment. In *2016 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)* (pp. 49-54). IEEE.
46. Sasaki, H., Horiuchi, T., & Kato, S. (2017, September). A study on vision-based mobile robot learning by deep Q-network. In *2017 56th annual conference of the society of instrument and control engineers of Japan (SICE)* (pp. 799-804). IEEE.
47. Sheridan, T. B. (2016). Human–robot interaction: status and challenges. *Human factors*, *58*(4), 525-532.
48. Singh, B., Sellappan, N., & Kumaradhas, P. (2013). Evolution of industrial robots and their applications. *International Journal of emerging technology and advanced engineering*, *3*(5), 763-768.
49. Sokolov, M., Lavrenov, R., Gabdullin, A., Afanasyev, I. and Magid, E., 2016, December. 3D modelling and simulation of a crawler robot in ROS/Gazebo. In *Proceedings of the 4th International Conference on Control, Mechatronics and Automation* (pp. 61-65).
50. Sonoura, T., Yoshimi, T., Nishiyama, M., Nakamoto, H., Tokura, S., & Matsuhira, N. (2008). Person following robot with vision-based and sensor fusion tracking algorithm. *Computer vision*, 519-538.
51. Svenstrup, M., Tranberg, S., Andersen, H. J., & Bak, T. (2009, May). Pose estimation and adaptive robot behaviour for human-robot interaction. In *2009 IEEE International Conference on Robotics and Automation* (pp. 3571-3576). IEEE.
52. Takaya, K., Asai, T., Kroumov, V. and Smarandache, F., 2016, October. Simulation environment for mobile robots testing using ROS and Gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)* (pp. 96-101). IEEE.
53. Tripathi, A., Khan, M. A., Pandey, A., Yadav, P., & Sharma, A. K. (2021, December). Human Following Robot using Ultrasonic Sensor. In *2021 3rd International Conference*

on Advances in Computing, Communication Control and Networking (ICAC3N) (pp. 764-770). IEEE.
54. Tsarouchi, P., Makris, S., & Chryssolouris, G. (2016). Human–robot interaction review and challenges on task planning and programming. *International Journal of Computer Integrated Manufacturing*, *29*(8), 916-931.
55. Vachnadze, G. (2021, February 4). *Isaac Asimov's Three fundamental laws of robotics (and the zeroth law)*. Medium. https://giorgivachnadze.medium.com/isaac-asimovs-three-fundamental-laws-of-robotics-and-the-zeroth-law-ded9a781491b
56. Verma, J. (2022, August 3). *What is .BASHRC file in linux?*. DigitalOcean. https://www.digitalocean.com/community/tutorials/bashrc-file-in-linux
57. Weber, A. (2014, February 10). *Human-robot collaboration comes of age*. ASSEMBLY RSS. https://www.assemblymag.com/articles/91862-human-robot-collaboration-comes-of-age
58. Yasar, K., & Hanna, K. T. (2023, July 12). *What is robotics?: Definition from Whatis*. WhatIs.com. https://www.techtarget.com/whatis/definition/robotics
59. Zhao, L., & Li, S. (2020). Object detection algorithm based on improved YOLOv3. *Electronics*, *9*(3), 537.
60. Žlajpah, L. (2008). Simulation in robotics. *Mathematics and Computers in Simulation*, *79*(4), 879-897.

7 Appendix

Appendix A – Configuring the virtual environment

A1 – Configure Bash File

conda init bash

exit

A2 – Set up a compatible Python version and PyTorch Version

conda create --name openmmlab python=3.8.10 -y

conda activate openmmlab

conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia

A3 – Install MMDetection

pip install -U openmim

mim install mmengine

mim install "mmcv>=2.0.0"

git clone https://github.com/open-mmlab/mmdetection.git

cd mmdetection

pip install -v -e .

A4 – Update bash file

echo "conda activate openmmlab" >> ~/.bashrc

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

echo 'source /usr/share/gazebo/setup.sh' >> ~/.bashrc

echo 'export ROS_DOMAIN_ID=81' >> ~/.bashrc

echo 'export GAZEBO_MASTER_URI=http://localhost:8181' >> ~/.bashrc

echo 'export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:~/GazeboModels/' >> ~/.bashrc

A5 – Install additional packages

pip install -U catkin_pkg

pip install empy

pip install lark

pip3 install labelImg

A6 – Build ROS packages for TurtleBot3

cd ~/dev_ws

colcon build

colcon build

Appendix B – Features of the TurtleBot3 Waffle Pi

B1 – Waffle Pi Configuration File

```
<?xml version="1.0"?>

<model>
  <name>TurtleBot3(Waffle Pi)</name>
  <version>2.0</version>
  <sdf version="1.4">model-1_4.sdf</sdf>
  <sdf version="1.5">model.sdf</sdf>

  <author>
    <name>Taehun Lim(Darby)</name>
    <email>thlim@robotis.com</email>
  </author>

  <description>
    TurtleBot3 Waffle Pi
  </description>
</model>
```

B2 – Code snippet for the Waffle Pi SDF file – Camera

```
<link name="camera_link"/>

<link name="camera_rgb_optical_frame">
  <inertial>
    <pose>0.076 0.0 0.093 0 0 0</pose>
    <inertia>
      <ixx>0.001</ixx>
      <ixy>0.000</ixy>
      <ixz>0.000</ixz>
      <iyy>0.001</iyy>
      <iyz>0.000</iyz>
      <izz>0.001</izz>
    </inertia>
    <mass>0.035</mass>
  </inertial>

  <collision name="collision">
    <pose>0.076 0.0 0.093 0 0 0</pose>
    <geometry>
      <box>
        <size>0.008 0.130 0.022</size>
      </box>
    </geometry>
  </collision>
  <pose>0.076 0.0 0.093 0 0 0</pose>
  <sensor name="camera" type="camera">
    <always_on>true</always_on>
    <visualize>true</visualize>
    <update_rate>30</update_rate>
    <camera name="picam">
      <horizontal_fov>1.085595</horizontal_fov>
      <image>
          <width>640</width>
          <height>480</height>
          <format>R8G8B8</format>
      </image>
      <clip>
          <near>0.03</near>
          <far>100</far>
      </clip>
      <noise>
        <type>gaussian</type>
        <!-- Noise is sampled independently per pixel on each frame.
              That pixel's noise value is added to each of its color
              channels, which at that point lie in the range [0,1]. -->
        <mean>0.0</mean>
        <stddev>0.007</stddev>
      </noise>
    </camera>
      <plugin name="camera_driver" filename="libgazebo_ros_camera.so">
        <ros>
          <!-- <namespace>test_cam</namespace> -->
          <!-- <remapping>image_raw:=image_demo</remapping> -->
          <!-- <remapping>camera_info:=camera_info_demo</remapping> -->
        </ros>
        <!-- camera_name>omit so it defaults to sensor name</camera_name-->
        <!-- frame_name>omit so it defaults to link name</frameName-->
        <!-- <hack_baseline>0.07</hack_baseline> -->
      </plugin>
  </sensor>
</link>
```

B3 – Command used to run Gazebo world alongside the TurtleBot3 Waffle Pi

gazebo --verbose -s libgazebo_ros_init.so -s libgazebo_ros_factory.so ./{gazebo_world_name}

example:
gazebo --verbose -s libgazebo_ros_init.so -s libgazebo_ros_factory.so ./walk_straight.world

Appendix C – Creating Gazebo Scenarios

C1 – Snippet of the dae file for the human model

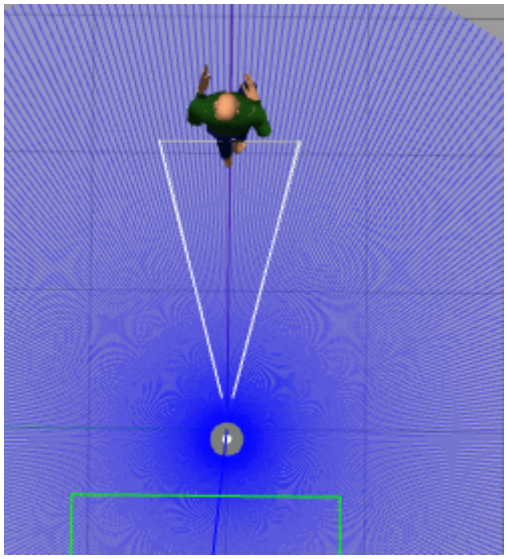
```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema" version="1.4.1">
  <asset>
    <contributor>
      <author>Blender User</author>
      <authoring_tool>Blender 2.63.5 r46744</authoring_tool>
    </contributor>
    <created>2012-05-19T18:37:55</created>
    <modified>2012-05-19T18:37:55</modified>
    <unit name="meter" meter="1"/>
    <up_axis>Z_UP</up_axis>
  </asset>
  <library_effects>
    <effect id="eyes-white-effect">
      <profile_COMMON>
        <technique sid="common">
          <phong>
            <emission>
              <color sid="emission">0 0 0 1</color>
            </emission>
            <ambient>
              <color sid="ambient">0.6220113 0.6209189 0.64 1</color>
            </ambient>
            <diffuse>
              <color sid="diffuse">0.6220113 0.6209189 0.64 1</color>
            </diffuse>
            <specular>
              <color sid="specular">1 1 1 1</color>
            </specular>
            <shininess>
              <float sid="shininess">23</float>
            </shininess>
            <index_of_refraction>
              <float sid="index_of_refraction">1</float>
            </index_of_refraction>
          </phong>
        </technique>
      </profile_COMMON>
    </effect>
    <effect id="eyes-brown-effect">
      <profile_COMMON>
        <technique sid="common">
          <phong>
            <emission>
              <color sid="emission">0 0 0 1</color>
            </emission>
            <ambient>
              <color sid="ambient">0.1969495 0.07701492 0.03641987 1</color>
            </ambient>
            <diffuse>
              <color sid="diffuse">0.1969495 0.07701492 0.03641987 1</color>
            </diffuse>
            <specular>
```

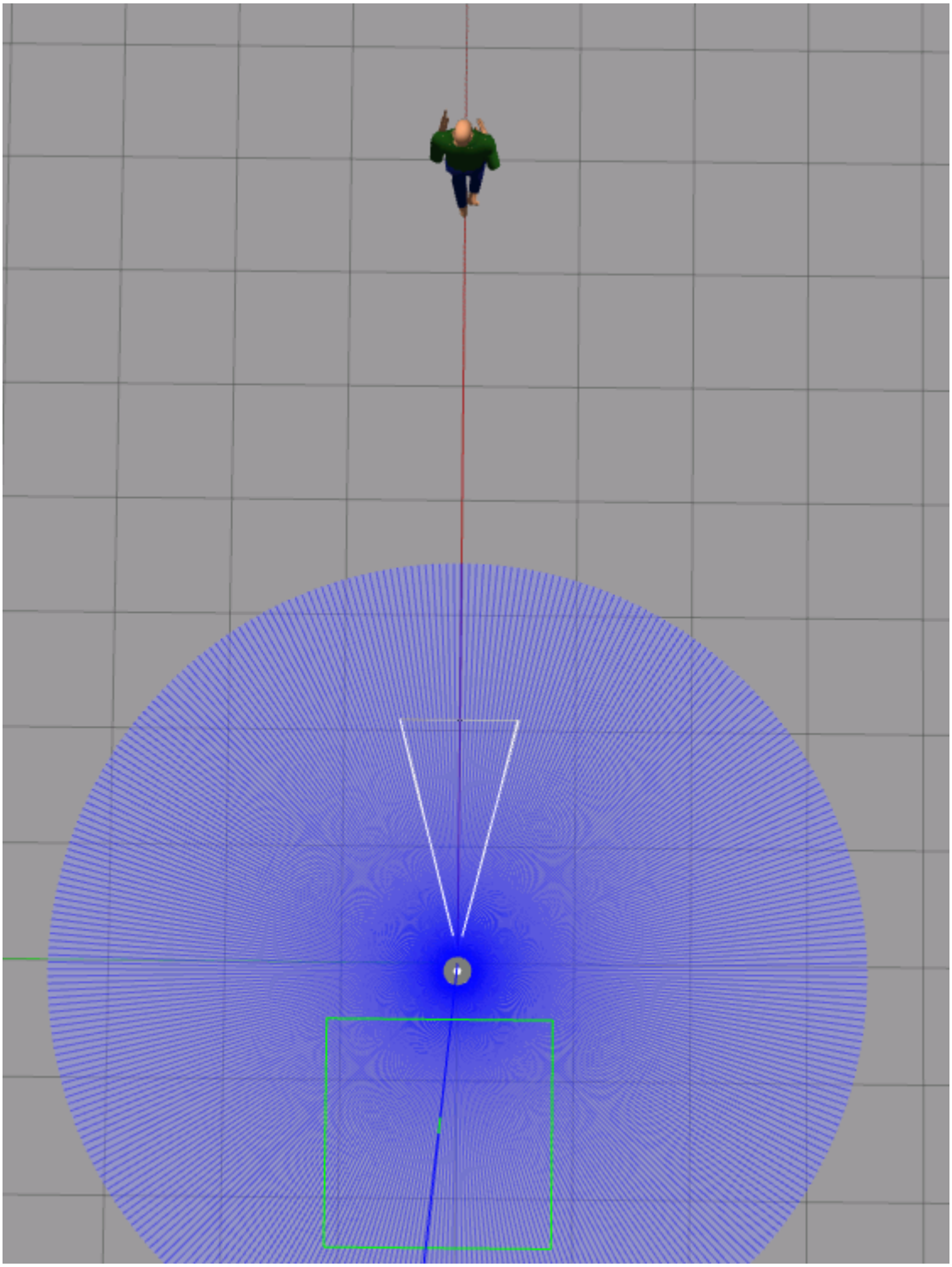
C2 – Basic Configuration present in all scenarios

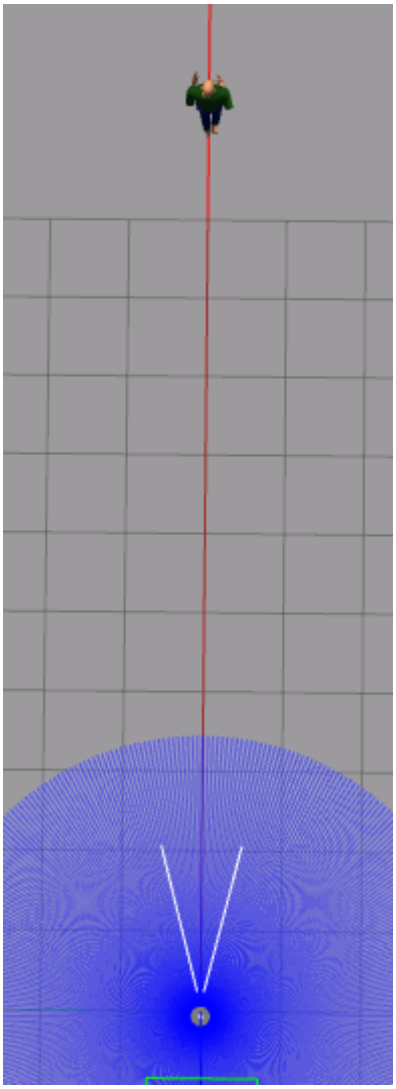
```xml
<?xml version="1.0"?>
<sdf version="1.6">
  <world name="default">

    <include>
      <uri>model://ground_plane</uri>
    </include>

    <include>
      <uri>model://sun</uri>
    </include>

    <scene>
      <shadows>false</shadows>
    </scene>

    <gui fullscreen='0'>
      <camera name='user_camera'>
        <pose frame=''>0.319654 -0.235002 9.29441 0 1.5138 0.009599</pose>
        <view_controller>orbit</view_controller>
        <projection_type>perspective</projection_type>
      </camera>
    </gui>

    <physics type="ode">
      <real_time_update_rate>1000.0</real_time_update_rate>
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <ode>
        <solver>
          <type>quick</type>
          <iters>150</iters>
          <precon_iters>0</precon_iters>
          <sor>1.400000</sor>
          <use_dynamic_moi_rescaling>1</use_dynamic_moi_rescaling>
        </solver>
        <constraints>
          <cfm>0.00001</cfm>
          <erp>0.2</erp>
          <contact_max_correcting_vel>2000.000000</contact_max_correcting_vel>
          <contact_surface_layer>0.01000</contact_surface_layer>
        </constraints>
      </ode>
    </physics>
```

C3 – Scenario 1 (Walking in a straight line)

```
<actor name="actor">
  <skin>
    <filename>run.dae</filename>
  </skin>
  <animation name="running">
    <filename>run.dae</filename>
    <interpolate_x>true</interpolate_x>
  </animation>
  <script>
    <trajectory id="0" type="running">
      <!-- Adjust the starting position to be directly in front of the robot -->
      <waypoint>
        <time>0</time>
        <pose>2 0 0 0 0 0</pose>
      </waypoint>
      <!-- Adjust the ending position according to your desired distance -->
      <waypoint>
        <time>20</time>
        <pose>15 0 0 0 0 0</pose>
      </waypoint>
    </trajectory>
  </script>
</actor>
```

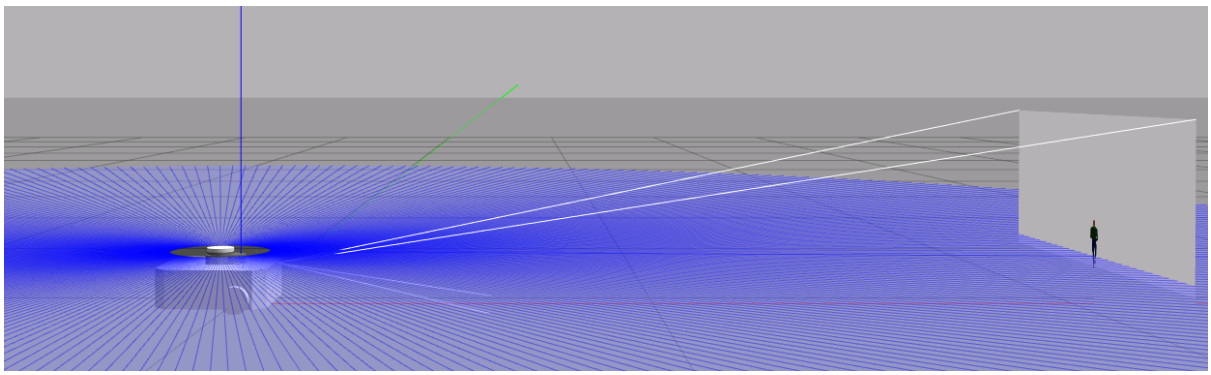






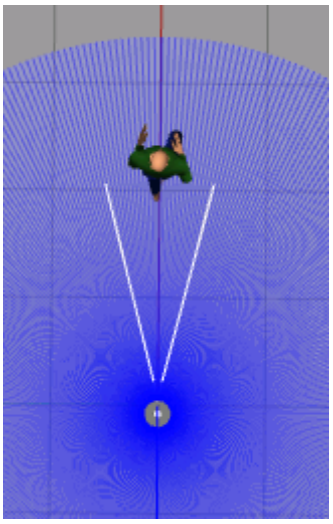
C4 – Calling the Waffle Pi Robot into the Gazebo Simulation

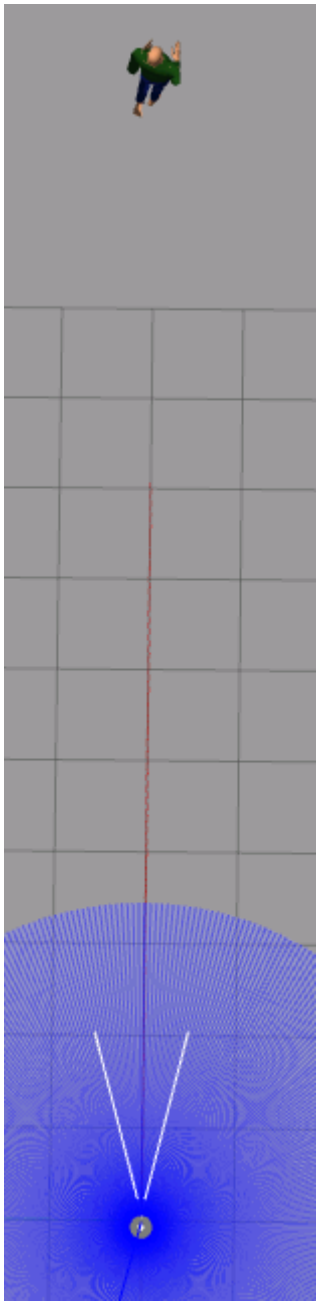
```
<include>
  <pose>0.0 0.0 0.0 0.0 0.0 0.0</pose>
  <uri>model://turtlebot3_waffle_pi</uri>
</include>
```

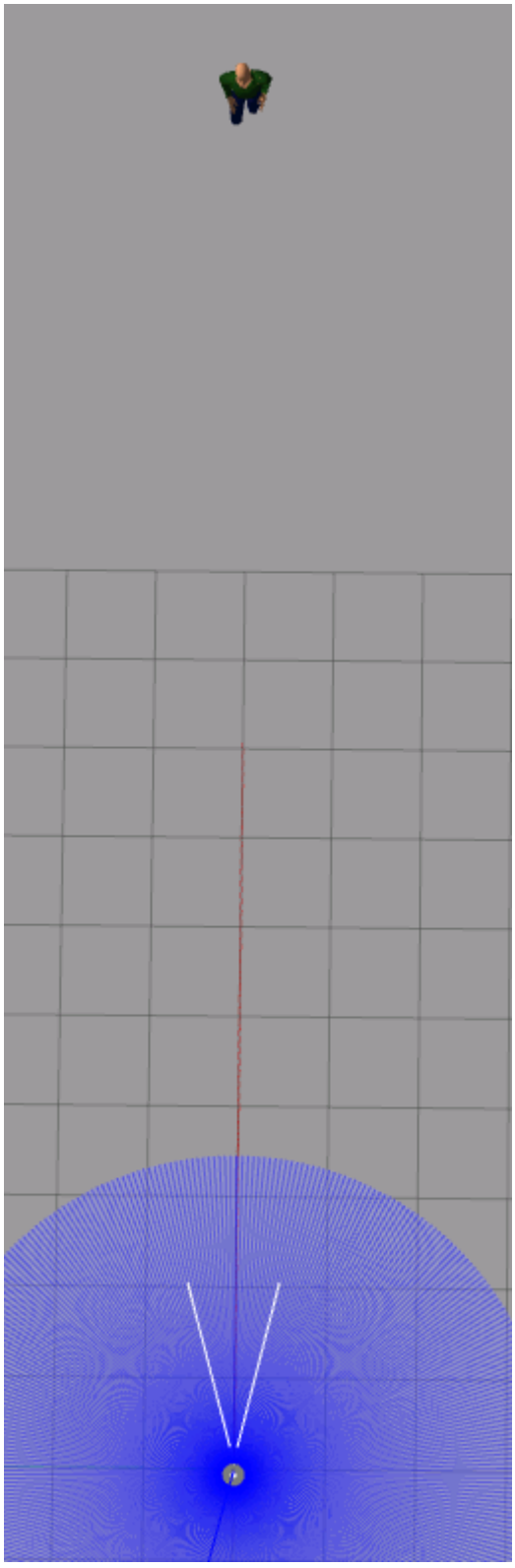


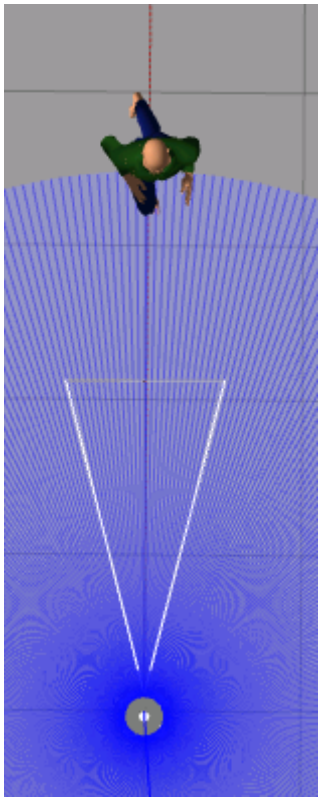
C5 – Scenario 2 (Walking back and forth)

```
<actor name="actor">
  <skin>
    <filename>run.dae</filename>
  </skin>
  <animation name="running">
    <filename>run.dae</filename>
    <interpolate_x>true</interpolate_x>
  </animation>
  <script>
    <trajectory id="0" type="running">
      <!-- Adjust the starting position to be directly in front of the robot -->
      <waypoint>
        <time>0</time>
        <pose>2 0 0 0 0 0</pose>
      </waypoint>
      <!-- Walk forward 13 metres -->
      <waypoint>
        <time>20</time>
        <pose>15 0 0 0 0 0</pose>
      </waypoint>
      <!-- Turn around 180 degrees -->
      <waypoint>
        <time>25</time>
        <pose>15 0 0 0 0 3.14159</pose>
      </waypoint>
      <!-- Walk back 13 metres -->
      <waypoint>
        <time>45</time>
        <pose>0 0 0 0 0 3.14159</pose>
      </waypoint>
      <!-- Turn around 180 degrees -->
      <waypoint>
        <time>50</time>
        <pose>2 0 0 0 0 0</pose>
      </waypoint>
    </trajectory>
  </script>
</actor>
```



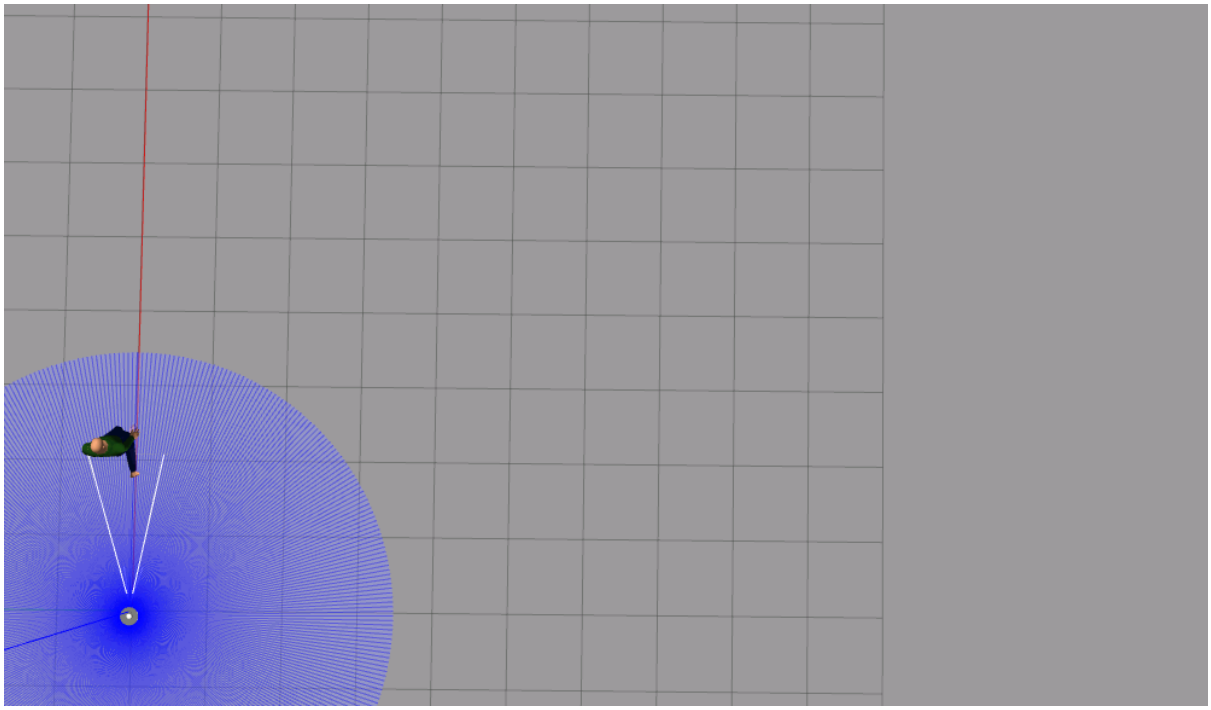


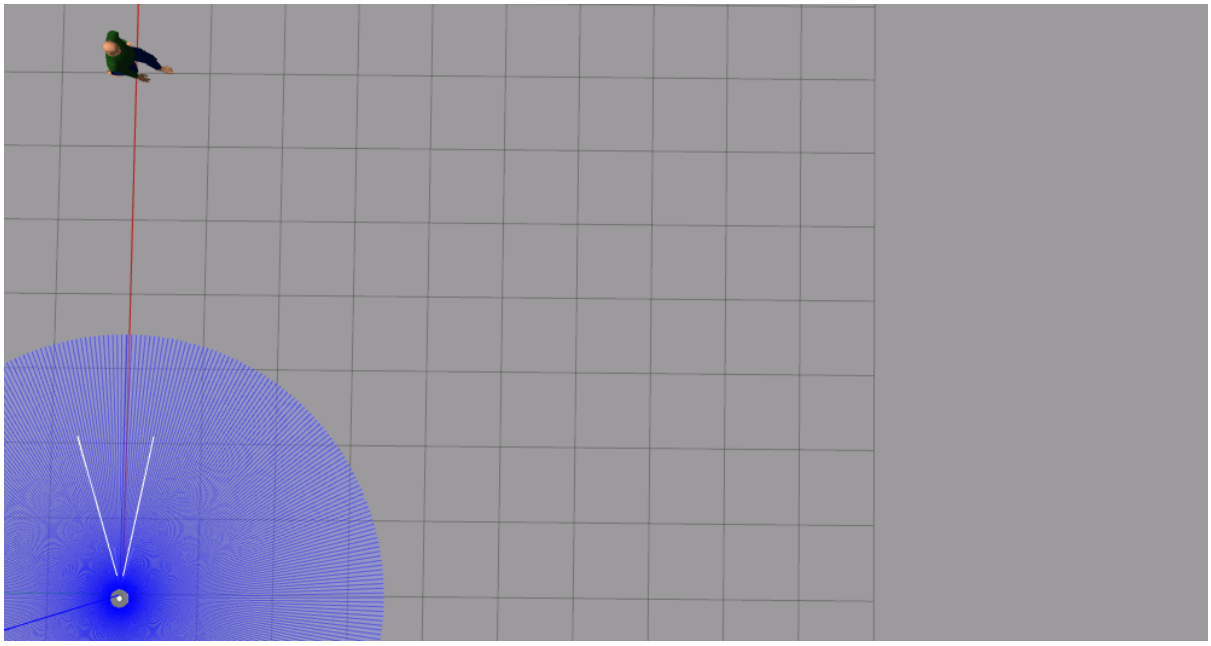


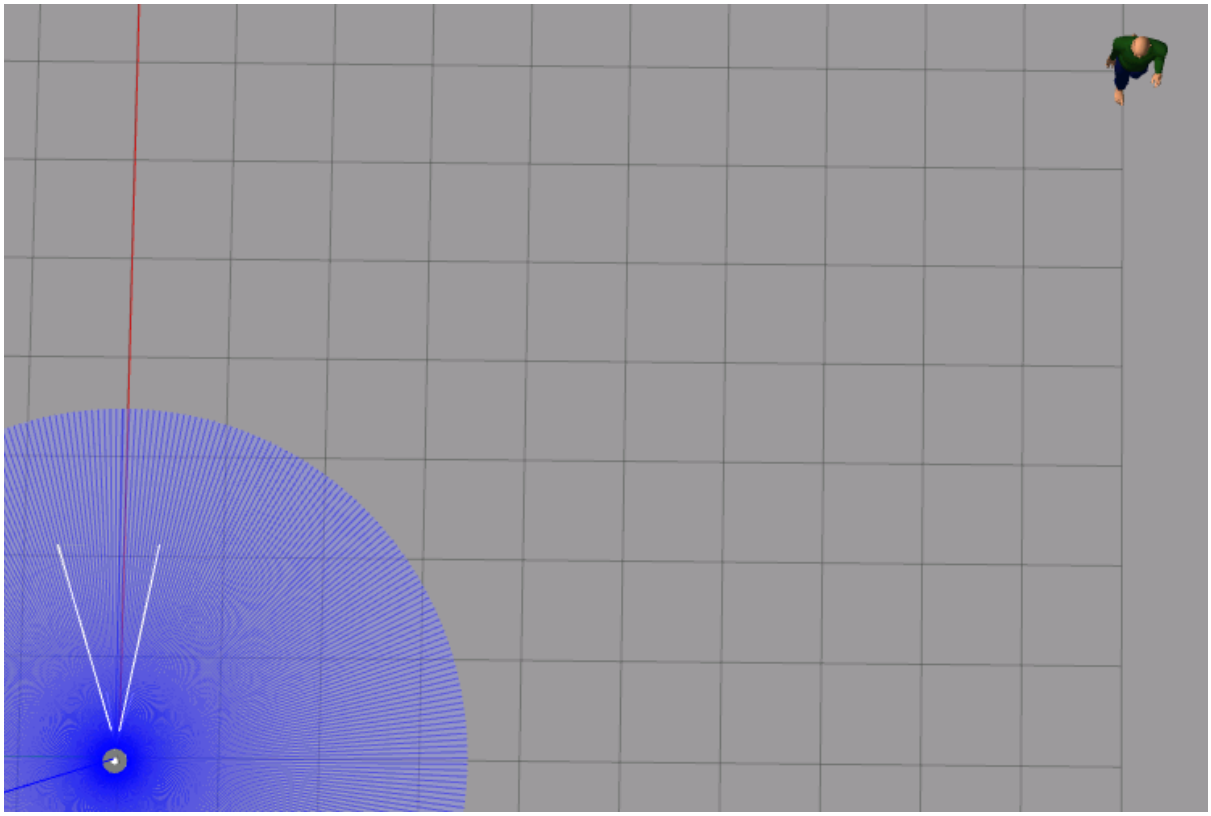


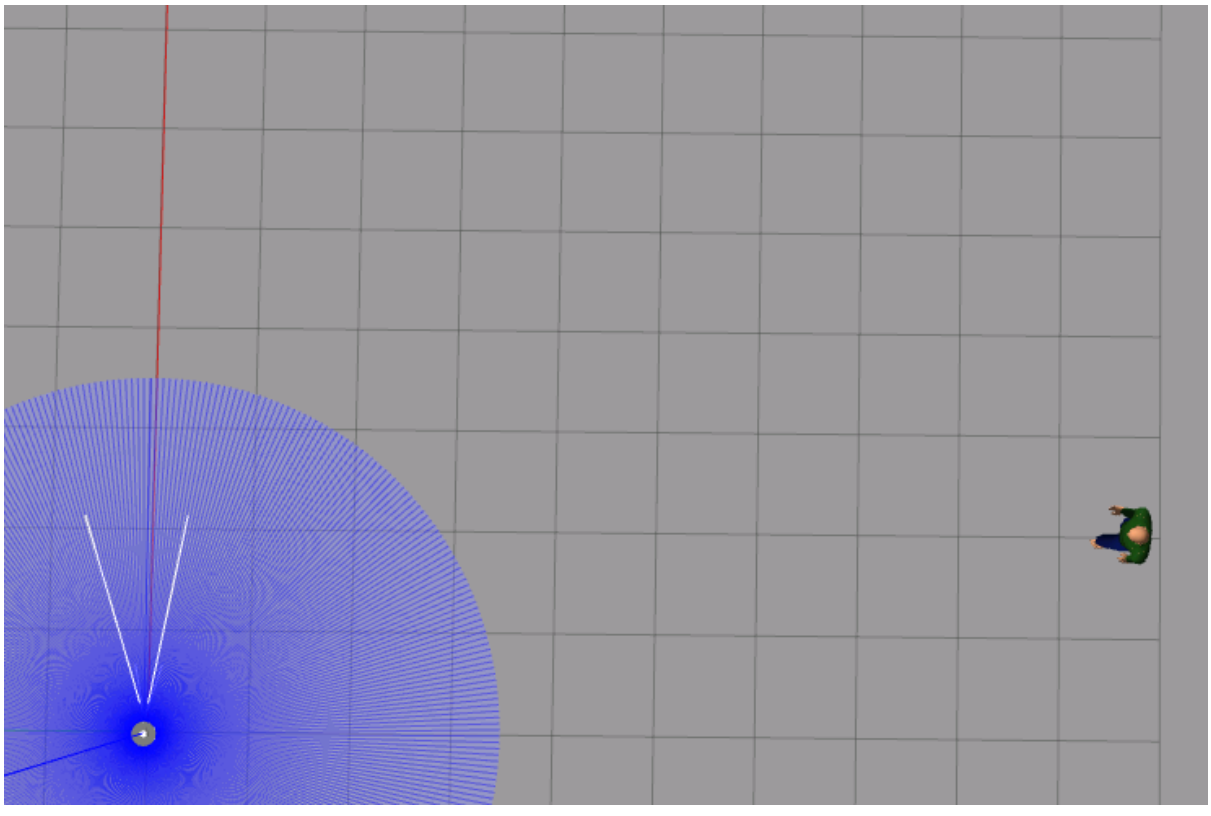
C6 – Scenario 3 (Walking in a rectangle)

```
<actor name="actor">
  <skin>
    <filename>walk.dae</filename>
  </skin>
  <animation name="walking">
    <filename>walk.dae</filename>
    <interpolate_x>true</interpolate_x>
  </animation>
  <script>
    <trajectory id="0" type="walking">
      <!-- Adjust the starting position to be directly in front of the robot -->
      <waypoint>
        <time>0</time>
        <pose>2 0 0 0 0 0</pose>
      </waypoint>
      <!-- Walk forward for 5 meters -->
      <waypoint>
        <time>10</time>
        <pose>7 0 0 0 0 0</pose>
      </waypoint>
      <!-- Change orientation to look to the right -->
      <waypoint>
        <time>15</time>
        <pose>7 0 0 0 0 -1.57</pose>
      </waypoint>
      <!-- Walk in a straight line for 10 meters -->
      <waypoint>
        <time>35</time>
        <pose>7 -10 0 0 0 -1.57</pose>
      </waypoint>
      <!-- Change orientation to look to the right -->
      <waypoint>
        <time>40</time>
        <pose>7 -10 0 0 0 -3.14</pose>
      </waypoint>
      <!-- Walk in a straight line for 5 meters -->
      <waypoint>
        <time>50</time>
        <pose>2 -10 0 0 0 -3.14</pose>
      </waypoint>
      <!-- Change orientation to look to the right -->
      <waypoint>
        <time>55</time>
        <pose>2 -10 0 0 0 -4.71</pose>
      </waypoint>
      <!-- Walk in a straight line for 10 meters to return to the starting point -->
      <waypoint>
        <time>75</time>
        <pose>2 0 0 0 0 -4.71</pose>
      </waypoint>
      <!-- Change orientation to look to the right -->
      <waypoint>
        <time>80</time>
        <pose>2 0 0 0 0 -6.28</pose>
      </waypoint>
    </trajectory>
  </script>
</actor>
```







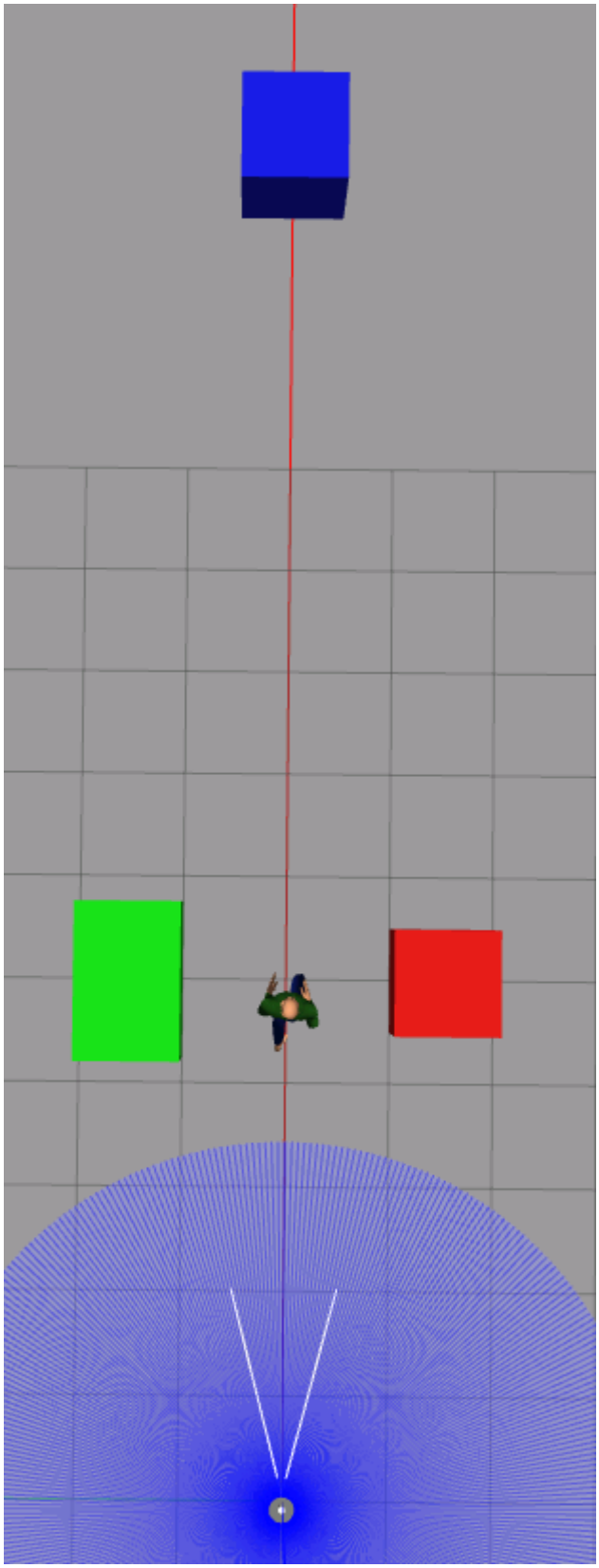


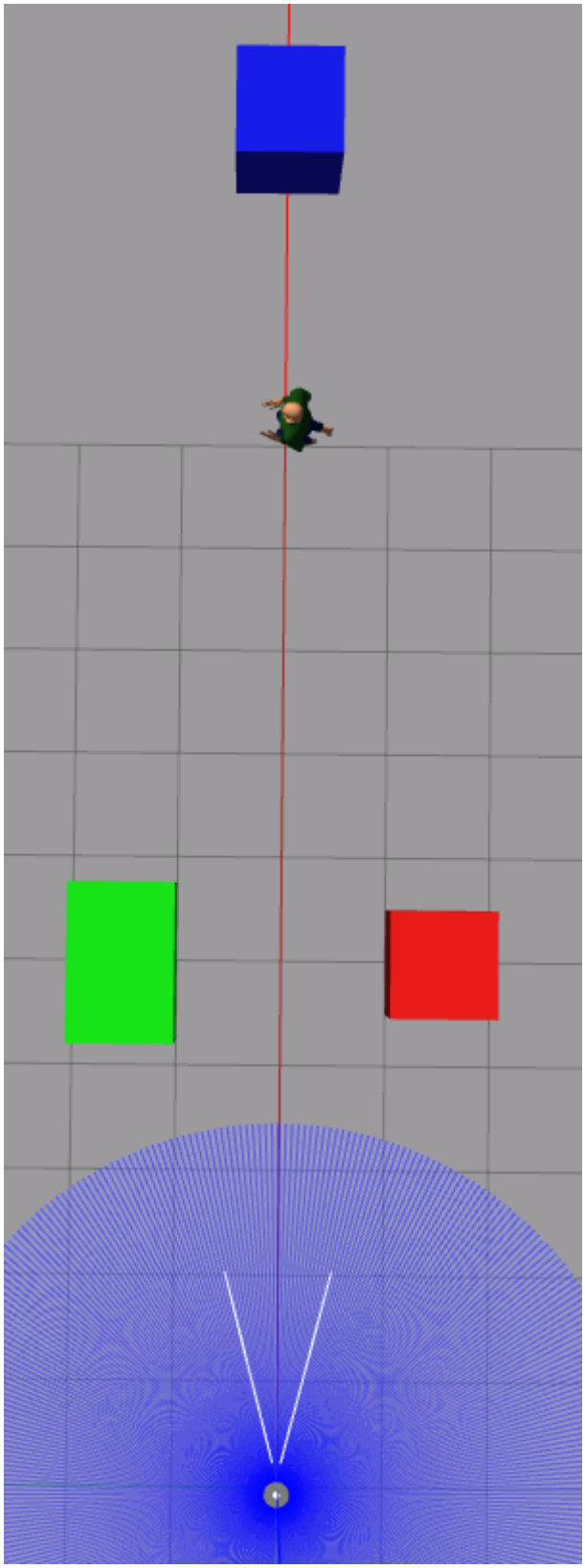
C7 – Scenario 4 (Walking around obstacles)

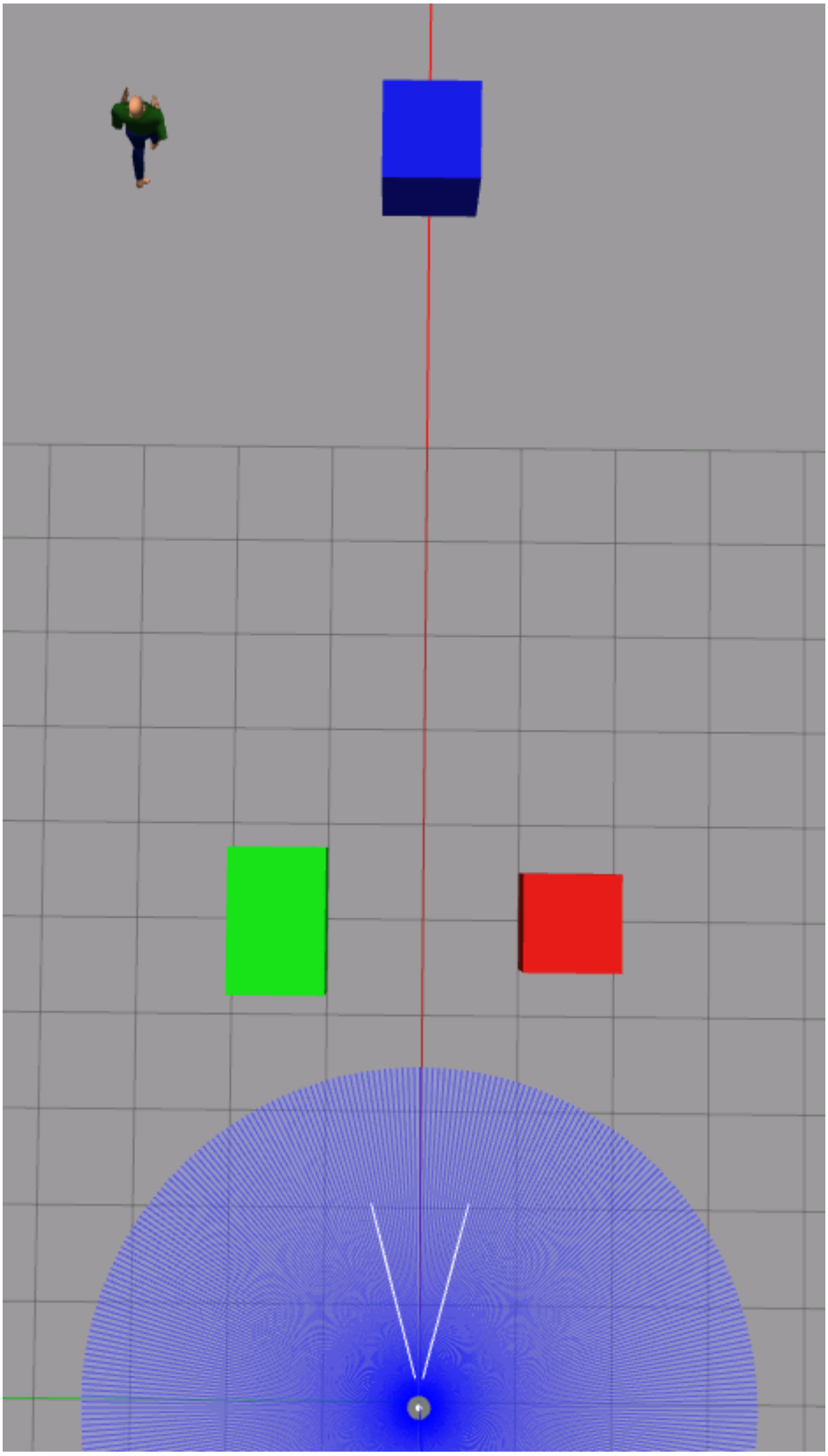
```
<actor name="actor">
  <skin>
    <filename>run.dae</filename>
  </skin>
  <animation name="running">
    <filename>run.dae</filename>
    <interpolate_x>true</interpolate_x>
  </animation>
  <script>
    <trajectory id="0" type="running">
      <!-- Adjust the starting position to be directly in front of the robot -->
      <waypoint>
        <time>0</time>
        <pose>2 0 0 0 0 0</pose>
      </waypoint>
      <!-- Adjust the ending position according to your desired distance -->
      <waypoint>
        <time>15</time>
        <pose>10 0 0 0 0 0</pose>
      </waypoint>
      <!-- Stop in front of box, turn left -->
      <waypoint>
        <time>18</time>
        <pose>10 0 0 0 0 1.571</pose>
      </waypoint>
      <!-- Go straight 3 metres -->
      <waypoint>
        <time>25</time>
        <pose>10 3 0 0 0 1.571</pose>
      </waypoint>
      <!-- Turn right -->
      <waypoint>
        <time>28</time>
        <pose>10 3 0 0 0 0</pose>
      </waypoint>
      <!-- Go straight 5 metres -->
      <waypoint>
        <time>35</time>
        <pose>15 3 0 0 0 0</pose>
      </waypoint>
    </trajectory>
  </script>
</actor>

<include>
  <pose>0.0 0.0 0.0 0.0 0.0 0.0</pose>
  <uri>model://turtlebot3_waffle_pi</uri>
</include>
```

```
<!-- Obstacle Box 1 -->
<model name="obstacle_box_1">
  <static>true</static>
  <link name="box_link_1">
    <collision name="box_collision_1">
      <geometry>
        <box>
          <size>1.0 1.0 1.0</size> <!-- Customize the size as needed -->
        </box>
      </geometry>
    </collision>
    <visual name="box_visual_1">
      <geometry>
        <box>
          <size>1.0 1.0 1.0</size> <!-- Customize the size as needed -->
        </box>
      </geometry>
      <material>
        <ambient>0.8 0.1 0.1 1</ambient>
        <diffuse>0.8 0.1 0.1 1</diffuse>
        <specular>0.8 0.1 0.1 1</specular>
        <emissive>0 0 0 1</emissive>
      </material>
    </visual>
  </link>
  <pose>5.0 -1.5 0.5 0 0 0</pose> <!-- Customize the position (x, y, z) as needed -->
</model>


<!-- Obstacle Box 2 -->
<model name="obstacle_box_2">
  <static>true</static>
  <link name="box_link_2">
    <collision name="box_collision_2">
      <geometry>
        <box>
          <size>1.5 1.0 1.0</size> <!-- Customize the size as needed -->
        </box>
      </geometry>
    </collision>
    <visual name="box_visual_2">
      <geometry>
        <box>
          <size>1.5 1.0 1.0</size> <!-- Customize the size as needed -->
        </box>
      </geometry>
      <material>
        <ambient>0.1 0.8 0.1 1</ambient>
        <diffuse>0.1 0.8 0.1 1</diffuse>
        <specular>0.1 0.8 0.1 1</specular>
        <emissive>0 0 0 1</emissive>
      </material>
    </visual>
  </link>
  <pose>5.0 1.5 0.5 0 0 0</pose> <!-- Customize the position (x, y, z) as needed -->
</model>
```

```
<!-- Obstacle Box 3 -->
<model name="obstacle_box_3">
  <static>true</static>
  <link name="box_link_3">
    <collision name="box_collision_3">
      <geometry>
        <box>
          <size>1.0 1.0 1.5</size> <!-- Customize the size as needed -->
        </box>
      </geometry>
    </collision>
    <visual name="box_visual_3">
      <geometry>
        <box>
          <size>1.0 1.0 1.5</size> <!-- Customize the size as needed -->
        </box>
      </geometry>
      <material>
        <ambient>0.1 0.1 0.8 1</ambient>
        <diffuse>0.1 0.1 0.8 1</diffuse>
        <specular>0.1 0.1 0.8 1</specular>
        <emissive>0 0 0 1</emissive>
      </material>
    </visual>
  </link>
  <pose>13.0 0.0 0.75 0 0 0</pose> <!-- Customize the position (x, y, z) as needed -->
</model>
```







Appendix D – Data Pre-Processing

D1 – Using LabelImg

D2 – JSON annotation for Training dataset in COCO Format


```
{"info":{"year":"2023","version":"1","description":"Exported from roboflow.ai","contributor":"","url":"https://public.roboflow.ai/object-detection/undefined","date_created":"2023-07-20T14:04:04+00:00"},"licenses":[{"id":1,"url":"https://creativecommons.org/licenses/by/4.0/","name":"CC BY 4.0"}],"categories":[{"id":0,"name":"human","supercategory":"none"},{"id":1,"name":"person","supercategory":"human"}],"images":[{"id":0,"license":1,"file_name":"23_jpg.rf.01b136de39735a35fff9799faac7827d.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":1,"license":1,"file_name":"3_jpg.rf.08cf8e17d2eb3bc6b4f47a28eed207d9.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":2,"license":1,"file_name":"5_jpg.rf.09ae7aa80ce5bd7dd48008ef4bb3910b.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":3,"license":1,"file_name":"4_jpg.rf.1f495054b917be2e7a4b037ffbed22c7.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":4,"license":1,"file_name":"5_jpg.rf.18a59be2b7818204a1cf751ef6bf1ff3.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":5,"license":1,"file_name":"22_jpg.rf.1d381c6096b855e0a2443ed153346a6e.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":6,"license":1,"file_name":"19_jpg.rf.2cbd65885c3c63aa115904e632109457.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":7,"license":1,"file_name":"31_jpg.rf.242a2a70d047aaff887ea913c932759b.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":8,"license":1,"file_name":"23_jpg.rf.01e4f3cff77ab8dd4792e4de7612e74f.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":9,"license":1,"file_name":"20_jpg.rf.37a579c942a3257c1a7bab102887df5d.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":10,"license":1,"file_name":"8_jpg.rf.269e536ede85b248b612f54a179e89e9.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":11,"license":1,"file_name":"6_jpg.rf.26f8075eb2486252c5fad54c202dce79.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":12,"license":1,"file_name":"17_jpg.rf.33462b9f5e791735daaa2a2deac793bd.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":13,"license":1,"file_name":"32_jpg.rf.27b0bab6574ca4e3b3a06bdc8831e30df.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":14,"license":1,"file_name":"10_jpg.rf.15a2ad1e8804741c9dcc4a3561823e79.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":15,"license":1,"file_name":"32_jpg.rf.47f2d73f30350b8cf9081e0c3b6d56f5.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":16,"license":1,"file_name":"9_jpg.rf.37f6d40ef805717aa68f02b4f2dbfb48.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":17,"license":1,"file_name":"24_jpg.rf.47007999cc225ac071975b0748c8c9c.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":18,"license":1,"file_name":"29_jpg.rf.49e880ef42e381f5133431e572bf1db9.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":19,"license":1,"file_name":"3_jpg.rf.4a376c165b31234f4654066b23496e30.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":20,"license":1,"file_name":"19_jpg.rf.4da629308c7c8a7033eef5a35bdc0c95.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":21,"license":1,"file_name":"6_jpg.rf.10047202cfa963c3784301414b3ccc7c.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":22,"license":1,"file_name":"14_jpg.rf.533cbf4f87ae31bab03b281ab9413a4f.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":23,"license":1,"file_name":"22_jpg.rf.5138515785 6a4ae87e0b4eaa0eed218d.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":24,"license":1,"file_name":"33_jpg.rf.66f8accede5719a0a501c1c66c3c0917.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":25,"license":1,"file_name":"4_jpg.rf.443ee1d2c5a97e185d578622a2e538ee.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":26,"license":1,"file_name":"3_jpg.rf.53b2a04d83118865c70bd2ccd5795563.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":27,"license":1,"file_name":"32_jpg.rf.769b85b99078fe737abe0539535db6c2.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":28,"license":1,"file_name":"24_jpg.rf.86ff6ff3fdbcb3a8f50b15339e4e1021.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":29,"license":1,"file_name":"10_jpg.rf.969c6e5e262e26bed9cd34df41d759fd.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":30,"license":1,"file_name":"29_jpg.rf.52cf06e5ad6b7a921e37cf5058176684.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":31,"license":1,"file_name":"8_jpg.rf.af086004d904a28853d58b687568b169.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":32,"license":1,"file_name":"31_jpg.rf.69b99238 7d2d85515c0aab77a72214a3.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":33,"license":1,"file_name":"18_jpg.rf.8a721ea0bd5406f213b500d823aae723.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":34,"license":1,"file_name":"31_jpg.rf.b61e8581ad3046c5183f790e10d8af4e.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":35,"license":1,"file_name":"24_jpg.rf.acb4aa1c71f367f17ef54d555e243b2.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":36,"license":1,"file_name":"28_jpg.rf.919aaf8283b04e91d10af91cd490212e.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":37,"license":1,"file_name":"1_jpg.rf.a466f1ea0637859a4ec6c76f1b6cf2df.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":38,"license":1,"file_name":"29_jpg.rf.987c8db7845e6d3c01c24adacde2e37c.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":39,"license":1,"file_name":"18_jpg.rf.cd7952dabf35a68ec7caba58a0f1ccec.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":40,"license":1,"file_name":"15_jpg.rf.beibef2d2107367a0a9166fd7d6b343f.jpg","height":640
```

```
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":41,"license":1,"file_name":"33_jpg.rf.68fdf22e0d58fe22180e7430e3c38608.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":42,"license":1,"file_name":"28_jpg.rf
.7c9440b8c98cd41425e90b7870f1bdfa.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":43,"license":1,"file_name":"17_jpg.rf.87ce9bc9820ba9e56d8bcfe91408177a.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+0
0:00"},{"id":44,"license":1,"file_name":"25_jpg.rf.916d88ecc7eac6818831d6a33650ca35.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":45,"license":1,"file_name":"28_jpg.rf.b829388805b9b49afcefc747e765f903b.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":46,"license":1,"file_name":"22_jpg.rf.adfd3d9988c589636ae911b1e82e3406.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":47,"license":1,"file_name":"10_jpg.rf
.86670c0c9d1174fbca6102f2ee75ed8f.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":48,"license":1,"file_name":"23_jpg.rf.d3fdedcdac5edfbdf67af7e9a475f1bc.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+0
0:00"},{"id":49,"license":1,"file_name":"8_jpg.rf.9265deeec104716a5077becb69091e16.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":50,"license":1,"file_name":"14_jpg.rf.d740c4729d9f3ec4c87bccc82d573a22c.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":51,"license":1,"file_name":"20_jpg.rf.e045dfcd2063edb675e1d2b5756df5d3.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":52,"license":1,"file_name":"25_jpg.rf
.db61bf2614893cee1f074826c2d219c7.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":53,"license":1,"file_name":"15_jpg.rf.dae4546f272601096ad2053e03ddc675.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+0
0:00"},{"id":54,"license":1,"file_name":"9_jpg.rf.e933fdbe6ab24bd259a79f8a4c11a1ac.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":55,"license":1,"file_name":"0_jpg.rf.df1230efada6e47f5daba00ad86c2f5a.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":56,"license":1,"file_name":"18_jpg.rf.d9995cc6ce9442e24e4b5cc250b94026.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":57,"license":1,"file_name":"15_jpg.rf
.9ebc0f2bca3e196baecc3f47eea3b8e7.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":58,"license":1,"file_name":"14_jpg.rf.f39d81890537d502929adca6619f90eb.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+0
0:00"},{"id":59,"license":1,"file_name":"0_jpg.rf.f33b191e45910912ba5ad4a025883906.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":60,"license":1,"file_name":"33_jpg.rf.e38cbe4671ecb66f0fbafc273924ea9cc.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":61,"license":1,"file_name":"9_jpg.rf.f59eae655236886f7da8f92f9c9d4770.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":62,"license":1,"file_name":"0_jpg.rf
.fe552a64279d8324b197d14bab4a9f8c.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":63,"license":1,"file_name":"1_jpg.rf.f40c5f1d8257719d0d78e2a5134ac705.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+0
0:00"},{"id":64,"license":1,"file_name":"17_jpg.rf.f7688b5f4337035e5d73c1e11d0e0535.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":65,"license":1,"file_name":"25_jpg.rf.cd2050df95e88aa5ae8ca034a15fc369.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":66,"license":1,"file_name":"4_jpg.rf.e54473e2aeaa5550b8e93c2c242dfd5995.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":67,"license":1,"file_name":"1_jpg.rf
.dc254bcb4055e38f7fc313f514de5f8a.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":68,"license":1,"file_name":"5_jpg.rf.f2349d3da8c2c1e48fd81e6bf91a165c.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+0
0:00"},{"id":69,"license":1,"file_name":"19_jpg.rf.76ab9bd21b73ea3a44d34b43e88e5d2f.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":70,"license":1,"file_name":"6_jpg.rf.fdabe6bd8b92807eb6d1c503c45f1c44.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"}],"annotations":[{"id":0,"image_id":0,"category_id":1,"bbox":[227,318,230,322],"area":74060,"segmentation":[],"iscrowd":0},{"id":1,"image_id":1,"category_id":1,"bbox":[98,93,200,540],"area":108000
,"segmentation":[],"iscrowd":0},{"id":2,"image_id":2,"category_id":1,"bbox":[104,226,242,414],"area":100188,"segmentation":[],"iscrowd":0},{"id":3,"image_id":3,"category_id":1,"bbox":[0,0,334,639],"area":213426,"segmentation":[],"iscrowd":0},{"id":4
,"image_id":4,"category_id":1,"bbox":[56,187,275,445],"area":122375,"segmentation":[],"iscrowd":0},{"id":5,"image_id":5,"category_id":1,"bbox":[289,226,272,414],"area":112608,"segmentation":[],"iscrowd":0},{"id":6,"image_id":6,"category_id":1,"bbox":[235
,173,151.5,462],"area":69993,"segmentation":[],"iscrowd":0},{"id":7,"image_id":7,"category_id":1,"bbox":[213,11,204,626],"area":127704,"segmentation":[],"iscrowd":0},{"id":8,"image_id":8,"category_id":1,"bbox":[222,334,166,305],"area":50630,"segmentation"
:[],"iscrowd":0},{"id":9,"image_id":9,"category_id":1,"bbox":[238,175,169.5,464],"area":78648,"segmentation":[],"iscrowd":0},{"id":10,"image_id":10,"category_id":1,"bbox":[425,269,104,370.5],"area":38532,"segmentation":[],"iscrowd":0},{"id":11,"image_id
":11,"category_id":1,"bbox":[119,244,132.5,395.5],"area":52403.75,"segmentation":[],"iscrowd":0},{"id":12,"image_id":12,"category_id":1,"bbox":[121,17,322.5,622],"area":200595,"segmentation":[],"iscrowd":0},{"id":13,"image_id":13,"category_id":1,"bbox"
:[193,2,220.5,631],"area":139135.5,"segmentation":[],"iscrowd":0},{"id":14,"image_id":14,"category_id":1,"bbox":[0,110,164,530],"area":86920,"segmentation":[],"iscrowd":0},{"id":15,"image_id":15,"category_id":1,"bbox":[187,0,231,634],"area":146454
,"segmentation":[],"iscrowd":0},{"id":16,"image_id":16,"category_id":1,"bbox":[236,72,292,568],"area":165856,"segmentation":[],"iscrowd":0},{"id":17,"image_id":17,"category_id":1,"bbox":[230,407,148,232],"area":34336,"segmentation":[],"iscrowd":0},{"id"
:18,"image_id":18,"category_id":1,"bbox":[239,209,137,431],"area":59047,"segmentation":[],"iscrowd":0},{"id":19,"image_id":19,"category_id":1,"bbox":[91,63,200,540],"area":108000,"segmentation":[],"iscrowd":0},{"id":20,"image_id":20,"category_id":1,"bbox
":[166,157,252,483],"area":121716,"segmentation":[],"iscrowd":0},{"id":21,"image_id":21,"category_id":1,"bbox":[107,260,212,380],"area":80560,"segmentation":[],"iscrowd":0},{"id":22,"image_id":22,"category_id":1,"bbox":[169,1,123.5,328],"area":40508
,"segmentation":[],"iscrowd":0},{"id":23,"image_id":23,"category_id":1,"bbox":[224,245,188,394],"area":74072,"segmentation":[],"iscrowd":0},{"id":24,"image_id":24,"category_id":1,"bbox":[152,0,264,637],"area":168168,"segmentation":[],"iscrowd":0},{"id":25
,"image_id":25,"category_id":1,"bbox":[0,6,411,634],"area":260574,"segmentation":[],"iscrowd":0},{"id":26,"image_id":26,"category_id":1,"bbox":[125,85,136.5,527],"area":71935.5,"segmentation":[],"iscrowd":0},{"id":27,"image_id":27,"category_id":1,"bbox"
:[150,0,306,640],"area":195840,"segmentation":[],"iscrowd":0},{"id":28,"image_id":28,"category_id":1,"bbox":[253,390,200,250],"area":50000,"segmentation":[],"iscrowd":0},{"id":29,"image_id":29,"category_id":1,"bbox":[0,63,98.5,576],"area":56736
,"segmentation":[],"iscrowd":0},{"id":30,"image_id":30,"category_id":1,"bbox":[229,206,174,434],"area":75516,"segmentation":[],"iscrowd":0},{"id":31,"image_id":31,"category_id":1,"bbox":[419,270,110,370],"area":40700,"segmentation":[],"iscrowd":0},{"id"
:32,"image_id":32,"category_id":1,"bbox":[234,15,161.5,616],"area":99484,"segmentation":[],"iscrowd":0},{"id":33,"image_id":33,"category_id":1,"bbox":[156,78,315,562],"area":177030,"segmentation":[],"iscrowd":0},{"id":34,"image_id":34,"category_id":1
,"bbox":[218,12,194,624],"area":121056,"segmentation":[],"iscrowd":0},{"id":35,"image_id":35,"category_id":1,"bbox":[252,392,196,248],"area":48608,"segmentation":[],"iscrowd":0},{"id":36,"image_id":36,"category_id":1,"bbox":[242,294,137,346],"area":47402
,"segmentation":[],"iscrowd":0},{"id":37,"image_id":37,"category_id":1,"bbox":[286,0,434,640],"area":277760,"segmentation":[],"iscrowd":0},{"id":38,"image_id":38,"category_id":1,"bbox":[241,211,130,428.5],"area":55705,"segmentation":[],"iscrowd":0},{"id"
:39,"image_id":39,"category_id":1,"bbox":[190,92,232.5,546.5],"area":127061.25,"segmentation":[],"iscrowd":0},{"id":40,"image_id":40,"category_id":1,"bbox":[88,0,449,640],"area":287360,"segmentation":[],"iscrowd":0},{"id":41,"image_id":41,"category_id":1
,"bbox":[119,0,329,640],"area":210560,"segmentation":[],"iscrowd":0},{"id":42,"image_id":42,"category_id":1,"bbox":[219,289,161,351],"area":56511,"segmentation":[],"iscrowd":0},{"id":43,"image_id":43,"category_id":1,"bbox":[54,0,454,640],"area":290560
,"segmentation":[],"iscrowd":0},{"id":44,"image_id":44,"category_id":1,"bbox":[281,444,93.5,195.5],"area":18279.25,"segmentation":[],"iscrowd":0},{"id":45,"image_id":45,"category_id":1,"bbox":[247,295,131.5,344],"area":45236,"segmentation":[],"iscrowd":0}
,{"id":46,"image_id":46,"category_id":1,"bbox":[180,232,241,408],"area":98328,"segmentation":[],"iscrowd":0},{"id":47,"image_id":47,"category_id":1,"bbox":[0,185,157,535],"area":83995,"segmentation":[],"iscrowd":0},{"id":48,"image_id":48,"category_id":1
,"bbox":[224,324,207,316],"area":65412,"segmentation":[],"iscrowd":0},{"id":49,"image_id":49,"category_id":1,"bbox":[350,294,185,346],"area":64010,"segmentation":[],"iscrowd":0},{"id":50,"image_id":50,"category_id":1,"bbox":[170,0,189,323],"area":61047
,"segmentation":[],"iscrowd":0},{"id":51,"image_id":51,"category_id":1,"bbox":[199,166,225,474],"area":106650,"segmentation":[],"iscrowd":0},{"id":52,"image_id":52,"category_id":1,"bbox":[248,439,113,201],"area":22713,"segmentation":[],"iscrowd":0},{"id"
:53,"image_id":53,"category_id":1,"bbox":[89,0,449,640],"area":287360,"segmentation":[],"iscrowd":0},{"id":54,"image_id":54,"category_id":1,"bbox":[210,58,352,582],"area":204864,"segmentation":[],"iscrowd":0},{"id":55,"image_id":55,"category_id":1,"bbox"
:[107,31,236,605],"area":142780,"segmentation":[],"iscrowd":0},{"id":56,"image_id":56,"category_id":1,"bbox":[163,81,279,559],"area":155961,"segmentation":[],"iscrowd":0},{"id":57,"image_id":57,"category_id":1,"bbox":[146,2,334,633],"area":211422
,"segmentation":[],"iscrowd":0},{"id":58,"image_id":58,"category_id":1,"bbox":[168,0,135,328],"area":44280,"segmentation":[],"iscrowd":0},{"id":59,"image_id":59,"category_id":1,"bbox":[84,60,303,580],"area":175740,"segmentation":[],"iscrowd":0},{"id":60
,"image_id":60,"category_id":1,"bbox":[84,0,401,640],"area":256640,"segmentation":[],"iscrowd":0},{"id":61,"image_id":61,"category_id":1,"bbox":[267,93,218.5,546],"area":119301,"segmentation":[],"iscrowd":0},{"id":62,"image_id":62,"category_id":1,"bbox"
:[145,50,165.5,589.5],"area":97562.25,"segmentation":[],"iscrowd":0},{"id":63,"image_id":63,"category_id":1,"bbox":[250,1,389.5,637.5],"area":248306.25,"segmentation":[],"iscrowd":0},{"id":64,"image_id":64,"category_id":1,"bbox":[50,0,472,640],"area"
:302080,"segmentation":[],"iscrowd":0},{"id":65,"image_id":65,"category_id":1,"bbox":[205,431,138,209],"area":28842,"segmentation":[],"iscrowd":0},{"id":66,"image_id":66,"category_id":1,"bbox":[0,0,350,640],"area":224000,"segmentation":[],"iscrowd":0}
,{"id":67,"image_id":67,"category_id":1,"bbox":[185,0,455,640],"area":291200,"segmentation":[],"iscrowd":0},{"id":68,"image_id":68,"category_id":1,"bbox":[113,226,200,414],"area":82800,"segmentation":[],"iscrowd":0},{"id":69,"image_id":69,"category_id":1
,"bbox":[229,171,160,465],"area":74400,"segmentation":[],"iscrowd":0},{"id":70,"image_id":70,"category_id":1,"bbox":[113,250,166,390],"area":64740,"segmentation":[],"iscrowd":0}]}
```


<hide>

D3 – JSON Annotations for the Validation dataset in COCO format


```
{"info":{"year":"2023","version":"1","description":"Exported from roboflow.ai","contributor":"","url":"https://public.roboflow.ai/object-detection/undefined","date_created":"2023-07-20T14:04:04+00:00"},"licenses":[{"id":1,"url":"https://creativecommons.org
/licenses/by/4.0/","name":"CC BY 4.0"}],"categories":[{"id":0,"name":"human","supercategory":"none"},{"id":1,"name":"person","supercategory":"human"}],"images":[{"id":0,"license":1,"file_name":"13_jpg.rf.260f978a2db65cc07b53307c1673dee2.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":1,"license":1,"file_name":"34_jpg.rf.e6bf91419cbe9051cdc477c34c70917c.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":2,"license":1,"file_name":"2_jpg.rf
.8ce99784edbc806704b71e2638d12af5.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":3,"license":1,"file_name":"11_jpg.rf.475ffbf6ec9dccffe6caa80e8f079d42.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+0
0:00"},{"id":4,"license":1,"file_name":"21_jpg.rf.47b2da5cb0c9391f73b2900cac5f3c30.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":5,"license":1,"file_name":"30_jpg.rf.d34e4860067a0cfd05a7f5a2d4672bc1.jpg","height":640
,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"},{"id":6,"license":1,"file_name":"26_jpg.rf.01d54773a8eef11a77de3d425588b4af.jpg","height":640,"width":640,"date_captured":"2023-07-20T14:04:04+00:00"}],"annotations":[{"id":0,"image_id":0
,"category_id":1,"bbox":[405,0,164.5,330],"area":54285,"segmentation":[],"iscrowd":0},{"id":1,"image_id":1,"category_id":1,"bbox":[127,2,371.5,637.5],"area":236831.25,"segmentation":[],"iscrowd":0},{"id":2,"image_id":2,"category_id":1,"bbox":[75,0,228,638]
,"area":145464,"segmentation":[],"iscrowd":0},{"id":3,"image_id":3,"category_id":1,"bbox":[465,69,174,570.5],"area":99267,"segmentation":[],"iscrowd":0},{"id":4,"image_id":4,"category_id":1,"bbox":[196,149,234,490],"area":114660,"segmentation":[],"iscrowd"
:0},{"id":5,"image_id":5,"category_id":1,"bbox":[257,123,129.5,507.5],"area":65721.25,"segmentation":[],"iscrowd":0},{"id":6,"image_id":6,"category_id":1,"bbox":[247,430,144.5,209],"area":30200.5,"segmentation":[],"iscrowd":0}]}
```

Appendix E – Modifying the YOLOv3 configuration file

E1 – Code for 'yolov3.py'

```
_base_ = ['./yolov3_base.py']

# yapf:disable
model = dict(
    bbox_head=dict(
    num_classes=1,
        anchor_generator=dict(
            base_sizes=[[(220, 125), (128, 222), (264, 266)],
                        [(35, 87), (102, 96), (60, 170)],
                        [(10, 15), (24, 36), (72, 42)]])))
# yapf:enable

input_size = (640, 640)
train_pipeline = [
    dict(type='LoadImageFromFile', backend_args={{_base_.backend_args}}),
    dict(type='LoadAnnotations', with_bbox=True),
    # `mean` and `to_rgb` should be the same with the `preprocess_cfg`
    dict(
        type='Expand',
        mean=[123.675, 116.28, 103.53],
        to_rgb=True,
        ratio_range=(1, 2)),
    dict(
        type='MinIoURandomCrop',
        min_ious=(0.4, 0.5, 0.6, 0.7, 0.8, 0.9),
        min_crop_size=0.3),
    dict(type='Resize', scale=input_size, keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='PackDetInputs')
]
test_pipeline = [
    dict(type='LoadImageFromFile', backend_args={{_base_.backend_args}}),
    dict(type='Resize', scale=input_size, keep_ratio=True),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor'))
]
train_dataloader = dict(dataset=dict(dataset=dict(pipeline=train_pipeline)))
val_dataloader = dict(dataset=dict(pipeline=test_pipeline))
test_dataloader = val_dataloader
```

E2 – Code for 'yolov3_base.py'

```
_base_ = ['../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py']
# model settings
data_preprocessor = dict(
    type='DetDataPreprocessor',
    mean=[123.675, 116.28, 103.53],
    std=[58.395, 57.12, 57.375],
    bgr_to_rgb=True,
    pad_size_divisor=32)
model = dict(
    type='YOLOV3',
    data_preprocessor=data_preprocessor,
    backbone=dict(
        type='MobileNetV2',
        out_indices=(2, 4, 6),
        act_cfg=dict(type='LeakyReLU', negative_slope=0.1),
        init_cfg=dict(
            type='Pretrained', checkpoint='open-mmlab://mmdet/mobilenet_v2')),
    neck=dict(
        type='YOLOV3Neck',
        num_scales=3,
        in_channels=[320, 96, 32],
        out_channels=[96, 96, 96]),
    bbox_head=dict(
        type='YOLOV3Head',
        num_classes=1,
        in_channels=[96, 96, 96],
        out_channels=[96, 96, 96],
        anchor_generator=dict(
            type='YOLOAnchorGenerator',
            base_sizes=[[(116, 90), (156, 198), (373, 326)],
                        [(30, 61), (62, 45), (59, 119)],
                        [(10, 13), (16, 30), (33, 23)]],
            strides=[32, 16, 8]
        ),
        bbox_coder=dict(type='YOLOBBoxCoder'),
        featmap_strides=[32, 16, 8],
        loss_cls=dict(
            type='CrossEntropyLoss',
            use_sigmoid=True,
            loss_weight=1.0,
            reduction='sum'),
        loss_conf=dict(
            type='CrossEntropyLoss',
            use_sigmoid=True,
            loss_weight=1.0,
```

```
            loss_weight=1.0,
            reduction='sum'),
        loss_xy=dict(
            type='CrossEntropyLoss',
            use_sigmoid=True,
            loss_weight=2.0,
            reduction='sum'),
        loss_wh=dict(type='MSELoss', loss_weight=2.0, reduction='sum')),
    # training and testing settings
    train_cfg=dict(
        assigner=dict(
            type='GridAssigner',
            pos_iou_thr=0.5,
            neg_iou_thr=0.5,
            min_pos_iou=0)),
    test_cfg=dict(
        nms_pre=1000,
        min_bbox_size=0,
        score_thr=0.05,
        conf_thr=0.005,
        nms=dict(type='nms', iou_threshold=0.45),
        max_per_img=100))
# dataset settings
dataset_type = 'CocoDataset'

# Example to use different file client
# Method 1: simply set the data root and let the file I/O module
# automatically infer from prefix (not support LMDB and Memcache yet)

# data_root = 's3://openmmlab/datasets/detection/coco/'

# Method 2: Use `backend_args`, `file_client_args` in versions before 3.0.0rc6
# backend_args = dict(
#     backend='petrel',
#     path_mapping=dict({
#         './data/': 's3://openmmlab/datasets/detection/',
#         'data/': 's3://openmmlab/datasets/detection/'
#     }))
backend_args = None

train_pipeline = [
    dict(type='LoadImageFromFile', backend_args=backend_args),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(
        type='Expand',
```

```
        mean=data_preprocessor['mean'],
        to_rgb=data_preprocessor['bgr_to_rgb'],
        ratio_range=(1, 2)),
    dict(
        type='MinIoURandomCrop',
        min_ious=(0.4, 0.5, 0.6, 0.7, 0.8, 0.9),
        min_crop_size=0.3),
    dict(type='RandomResize', scale=[(320, 320), (416, 416)], keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='PackDetInputs')
]
test_pipeline = [
    dict(type='LoadImageFromFile', backend_args=backend_args),
    dict(type='Resize', scale=(416, 416), keep_ratio=True),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor'))
]

train_dataloader = dict(
    batch_size=4,
    num_workers=4,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    batch_sampler=dict(type='AspectRatioBatchSampler'),
    dataset=dict(
        type='RepeatDataset',  # use RepeatDataset to speed up training
        times=10,
        dataset=dict(
            type=dataset_type,
            data_root='coco_image/train',
            ann_file='_annotations.coco.json',
            data_prefix=dict(img=''),
            filter_cfg=dict(filter_empty_gt=True, min_size=32),
            pipeline=train_pipeline,
            backend_args=backend_args)))
val_dataloader = dict(
    batch_size=4,
    num_workers=4,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='DefaultSampler', shuffle=False),
```

```
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
        data_root='coco_image/valid',
        ann_file='_annotations.coco.json',
        data_prefix=dict(img=''),
        test_mode=True,
        pipeline=test_pipeline,
        backend_args=backend_args))
test_dataloader = val_dataloader

val_evaluator = dict(
    type='CocoMetric',
    ann_file='coco_image/valid/_annotations.coco.json',
    metric='bbox',
    backend_args=backend_args)
test_evaluator = val_evaluator

train_cfg = dict(max_epochs=100)

# optimizer
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(type='SGD', lr=0.003, momentum=0.9, weight_decay=0.0005),
    clip_grad=dict(max_norm=35, norm_type=2))

# learning policy
param_scheduler = [
    dict(
        type='LinearLR',
        start_factor=0.0001,
        by_epoch=False,
        begin=0,
        end=4000),
    dict(type='MultiStepLR', by_epoch=True, milestones=[24, 28], gamma=0.1)
]

find_unused_parameters = True

# NOTE: `auto_scale_lr` is for automatically scaling LR,
# USER SHOULD NOT CHANGE ITS VALUES.
# base_batch_size = (8 GPUs) x (24 samples per GPU)
auto_scale_lr = dict(base_batch_size=192)
```

Appendix F – Human Detection Algorithm

F1 – 'hd.py'

```
class HumanDetectionNode(Node):
    def __init__(self):
        super().__init__('human_detection_node')
        self.bridge = CvBridge()
        self.subscription = self.create_subscription(Image, '/camera/image_raw', self.image_callback, 10)
        self.pose_array_publisher = self.create_publisher(PoseArray, 'detected_persons', 10)  # Create PoseArray publisher
        self.model = init_detector('../mmdetection/configs/yolo/yolov3.py', '../mmdetection/work_dirs/new_50/epoch_50.pth', device='cuda:0')

    def image_callback(self, msg):
        print("Received a new image!")
        cv_image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
        result = self.detect_persons(cv_image)
        pose_array = self.create_pose_array(result)
        self.publish_pose_array(pose_array)

        # Display the image with detected bounding boxes
        image_with_boxes = self.draw_boxes(cv_image, result)
        cv2.imshow("Image with Detected Persons", image_with_boxes)
        cv2.waitKey(1)  # Wait for a short time to update the window

        # Print the generated PoseArray
        print("Generated PoseArray:")
        print(pose_array)

    def draw_boxes(self, image, persons):
        image_with_boxes = image.copy()

        for person in persons:
            _, _, x1, y1, x2, y2 = person
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            cv2.rectangle(image_with_boxes, (x1, y1), (x2, y2), (0, 255, 0), 2)

            # Display the label inside the box
            font = cv2.FONT_HERSHEY_SIMPLEX
            label_text = 'Person'
            label_size = cv2.getTextSize(label_text, font, 0.5, 1)[0]
            cv2.putText(image_with_boxes, label_text, (x1, y1 - label_size[1]), font, 0.5, (0, 255, 0), 1)

        return image_with_boxes

```

```
    def detect_persons(self, image):
        result = inference_detector(self.model, image)

        pred_instances = result.pred_instances

        persons = []  # Initialize an empty list to store detected persons

        # Loop through each predicted instance and extract relevant information
        for i in range(len(pred_instances)):
            label = pred_instances.labels[i].item()  # Get the label (class index)
            score = pred_instances.scores[i].item()  # Get the confidence score
            bbox = pred_instances.bboxes[i].cpu().numpy()  # Convert the bounding box to a NumPy array

            # Append the extracted information as a tuple to the persons list
            persons.append((label, score, *bbox))

        return persons

    def create_pose_array(self, persons):
        pose_array = PoseArray()
        pose_array.header.frame_id = 'camera_frame'  # Update the frame ID as needed

        for person in persons:
            label, score, x1, y1, x2, y2 = person
            pose = Pose()
            pose.position.x = (x1 + x2) / 2
            pose.position.y = (y1 + y2) / 2
            pose.position.z = 0.0  # Assuming a 2D plane
            pose.orientation.w = 1.0  # Default orientation
            pose_array.poses.append(pose)

        return pose_array

    def publish_pose_array(self, pose_array):
        self.pose_array_publisher.publish(pose_array)  # Publish the PoseArray message

def main():
    rclpy.init()
    node = HumanDetectionNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

F2 – 'hd_conf.py' – modified 'draw_boxes' function

```
    def draw_boxes(self, image, persons):
        image_with_boxes = image.copy()
        person_detected = False  # Flag to check if at least one person is detected

        for person in persons:
            label, score, x1, y1, x2, y2 = person
            if score > 0.1:
                person_detected = True  # Set the flag to True if any person with score > 0.1 is detected
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                cv2.rectangle(image_with_boxes, (x1, y1), (x2, y2), (0, 255, 0), 2)

                # Display the label inside the box
                font = cv2.FONT_HERSHEY_SIMPLEX
                label_text = 'Person'
                label_size = cv2.getTextSize(label_text, font, 0.5, 1)[0]
                cv2.putText(image_with_boxes, label_text, (x1, y1 - label_size[1]), font, 0.5, (0, 255, 0), 1)

        if not person_detected:
            print("No person detected")  # Print the message if no person with score > 0.1 is detected

        return image_with_boxes
```

Appendix G – Human Tracking Algorithm

G1 – 'ht0.py'

```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import PoseArray, Twist

class SimpleHumanFollowingNode(Node):
    def __init__(self):
        super().__init__('simple_human_following_node')
        self.subscription = self.create_subscription(PoseArray, 'detected_persons', self.pose_array_callback, 10)
        self.velocity_publisher = self.create_publisher(Twist, 'cmd_vel', 10)  # Publish control commands

    def pose_array_callback(self, msg):
        if msg.poses:
            # Get the position of the first detected person
            target_position = msg.poses[0].position

            # Calculate linear velocity
            linear_velocity = 0.5  # Fixed linear velocity

            # Calculate angular velocity to keep the target in front
            angular_velocity = 0.0  # No angular velocity for a simple following algorithm

            # Publish control commands
            control_command = Twist()
            control_command.linear.x = linear_velocity
            control_command.angular.z = angular_velocity
            self.velocity_publisher.publish(control_command)

def main():
    rclpy.init()
    node = SimpleHumanFollowingNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

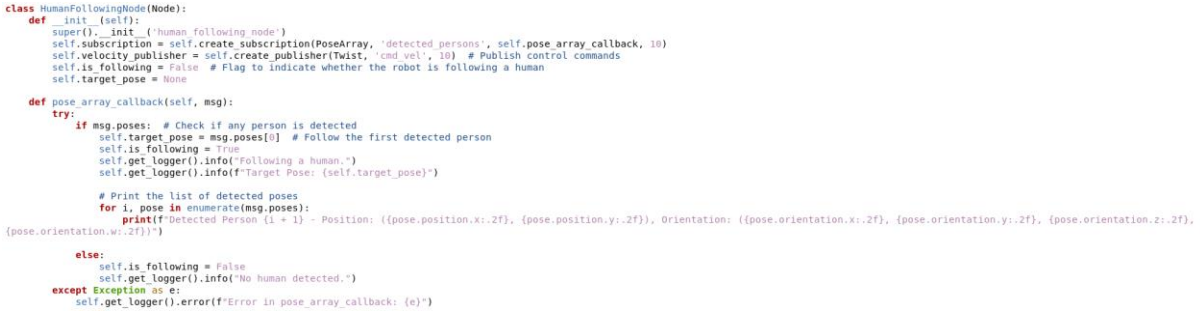
G2 – ‘ht1.py’ – Variable velocity

```
    def pose_array_callback(self, msg):
        if msg.poses:  # Check if any person is detected
            self.target_pose = msg.poses[0]  # Follow the first detected person
            self.is_following = True
            print("Following a human.")
            print("Target Pose:", self.target_pose)
        else:
            self.is_following = False
            print("No human detected.")

    def follow_human(self):
        if self.is_following and self.target_pose:
            # Calculate the distance between the robot and the human
            distance = math.sqrt(
                (self.target_pose.position.x ** 2) + (self.target_pose.position.y ** 2)
            )
            print("Distance to human:", distance)

            # Calculate the desired linear velocity based on the distance
            desired_linear_velocity = max(0.0, min(0.5, distance - 0.5))
            print("Desired linear velocity:", desired_linear_velocity)

            # Calculate the desired angular velocity based on the target's position
            angle_to_human = math.atan2(self.target_pose.position.y, self.target_pose.position.x)
            desired_angular_velocity = angle_to_human * 0.5  # Adjust the scaling factor as needed
            print("Desired angular velocity:", desired_angular_velocity)

            # Adjust the robot's linear velocity based on the angle to the human
            if abs(angle_to_human) > math.pi / 4:  # If the human is at a significant angle
                desired_linear_velocity = 0.1  # Slow down the robot's forward movement
                print("Adjusting linear velocity due to angle:", desired_linear_velocity)

            # Publish control commands
            control_command = Twist()
            control_command.linear.x = desired_linear_velocity
            control_command.angular.z = desired_angular_velocity
            self.velocity_publisher.publish(control_command)
            print("Published control commands.")
```

G3 – ‘ht2.py’ – Safety logic

```
    def pose_array_callback(self, msg):
        if msg.poses:  # Check if any person is detected
            self.target_pose = msg.poses[0]  # Follow the first detected person
            self.is_following = True
        else:
            self.is_following = False

    def follow_human(self, human_pose):
        # Define constants for desired behaviors
        FAST_LINEAR_SPEED = 0.9  # Adjust this value based on desired speed when not too close
        SAFE_DISTANCE = 0.5  # Adjust this value based on desired safe following distance
        ANGULAR_SPEED_GAIN = 0.75  # Adjust this value to control angular velocity adjustment

        robot_pose = self.get_robot_pose()  # Get the robot's current pose
        distance = self.calculate_distance(robot_pose, human_pose)

        if distance > SAFE_DISTANCE:
            # Calculate desired linear velocity
            desired_linear_velocity = FAST_LINEAR_SPEED

            # Calculate desired angular velocity
            angle_to_human = math.atan2(human_pose.position.y - robot_pose.position.y, human_pose.position.x - robot_pose.position.x)
            desired_angular_velocity = angle_to_human * ANGULAR_SPEED_GAIN

        else:
            # Robot is too close, stop
            desired_linear_velocity = 0.0
            desired_angular_velocity = 0.0

        # Publish control commands
        self.publish_control_commands(desired_linear_velocity, desired_angular_velocity)

    def main(self):
        rate = self.create_rate(10)  # Set the publishing rate

        while rclpy.ok():
            if self.is_following and self.target_pose:
                self.follow_human(self.target_pose)  # Provide the target_pose to the function
            rate.sleep()
```

G4 – 'ht3.py'



```
class HumanFollowingNode(Node):
    def __init__(self):
        super().__init__('human_following_node')
        self.subscription = self.create_subscription(PoseArray, 'detected_persons', self.pose_array_callback, 10)
        self.velocity_publisher = self.create_publisher(Twist, 'cmd_vel', 10)  # Publish control commands
        self.is_following = False  # Flag to indicate whether the robot is following a human
        self.target_pose = None

    def pose_array_callback(self, msg):
        try:
            if msg.poses:  # Check if any person is detected
                self.target_pose = msg.poses[0]  # Follow the first detected person
                self.is_following = True
                self.get_logger().info("Following a human.")
                self.get_logger().info(f"Target Pose: {self.target_pose}")

                # Print the list of detected poses
                for i, pose in enumerate(msg.poses):
                    print(f"Detected Person {i + 1} - Position: ({pose.position.x:.2f}, {pose.position.y:.2f}), Orientation: ({pose.orientation.x:.2f}, {pose.orientation.y:.2f}, {pose.orientation.z:.2f},
{pose.orientation.w:.2f})")

            else:
                self.is_following = False
                self.get_logger().info("No human detected.")
        except Exception as e:
            self.get_logger().error(f"Error in pose_array_callback: {e}")
```

```
    def follow_human(self):
        try:
            if self.is_following and self.target_pose:
                # Calculate the distance between the robot and the human
                distance = math.sqrt(
                    (self.target_pose.position.x ** 2) + (self.target_pose.position.y ** 2)
                )

                # Calculate the desired linear velocity based on the distance
                desired_linear_velocity = max(0.0, min(0.5, distance - 0.5))

                # Calculate the desired angular velocity based on the target's position
                angle_to_human = math.atan2(self.target_pose.position.y, self.target_pose.position.x)
                desired_angular_velocity = angle_to_human * 0.75

                # Check and adjust robot's linear velocity if too close to human
                if distance < 0.2:
                    desired_linear_velocity = 0.0

                # Publish control commands
                control_command = Twist()
                control_command.linear.x = desired_linear_velocity
                control_command.angular.z = desired_angular_velocity
                self.velocity_publisher.publish(control_command)
        except Exception as e:
            self.get_logger().error(f"Error in follow_human: {e}")

    def main(self):
        rate = self.create_rate(10)  # Set the publishing rate

        while rclpy.ok():
            try:
                self.follow_human()  # Implement the human-following algorithm
                rate.sleep()
            except Exception as e:
                self.get_logger().error(f"Error in main loop: {e}")

def main():
    rclpy.init()
    node = HumanFollowingNode()
    try:
        node.main()
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```