Towards a Swarm of Robots for Detecting Breaches in Social Distancing

Thesis Report

Serge Saaybi

TU Delft

Towards a Swarm of Robots for Detecting Breaches in Social Distancing

Thesis Report

by

Serge Saaybi

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday January 19, 2022 at 08:00.

Student number:	4999517	
Project duration:	April 1, 2021 – January 19, 2022	
Thesis committee:	Prof. dr. R. V. Prasad,	TU Delft, supervisor
	Prof. dr. ir. C. Verhoeven,	TU Delft
	Dr. A. Y. Majid,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements

Working on this thesis project has been extremely enriching to me on both a professional and a personal level. My time at Delft has been unforgettable and has helped shape me into the person I am today. I am extremely grateful to have been able to pursue a master's degree at TU Delft, which was for me the opportunity of a lifetime.

My time at TU Delft would not have been the same had it not been for the people that supported me throughout it all. I would like to start by expressing my gratitude towards my thesis daily supervisor, Dr. Amjad Majid. His guidance and mentorship were instrumental in aiding me towards the completion of this project. He was always available for any questions, offering his advise and expertise every step of the way. He encouraged me and kept pushing me up until the last minute to get the best possible outcome, and for that I am eternally grateful. I would also like to thank Dr. Ranga Rao Venkatesha Prasad for his valuable input and advice.

I would not have been able to get through this without my family and friends who were always there for me, offering encouragements and support whenever I needed them. I would like to thank my family and especially my parents and sister, who have accompanied me throughout all the ups and downs of my journey, always available to offer me their counsel. I was extremely lucky to have met some wonderful people at Delft, Belkassem Becetti, Mohammed El-Hayek, Avinash Kalloe and Habib El-Kassis. You have made this experience fun and unforgettable. I would also like to thank my friends who helped me execute the experiments for this thesis. Thank you for your patience and help, which was essential for the completion of this project. I would not have been able to do it without you. Finally, I am incredibly thankful for Mariane Bacha, who helped me at every step during this Master, and always encouraged me to overcome the challenges I would face.

Abstract

Robotic agents can continuously provide feedback to people based on their behaviors. For instance, a robot swarm can remind a group of people to respect social distancing guidelines during a pandemic or discourage unwanted behavior such as littering. However, developing a swarm robot to operate in realistic situations is challenging: a robot requires significant resources to operate in the real world, yet costs need to be kept low to produce the robots en masse.

To develop a swarm robot for encouraging social distancing, we, therefore, compare the performance of different deep reinforcement learning algorithms for robot navigation and various vision sensors and algorithms for detecting social distances breaches.

The resulting robot features a novel compound vision system that enables it to detect social distancing breaches up to +12 m away, and is able to navigate using a hybrid navigation stack that combines Deep Reinforcement Learning (DRL) and a probabilistic localization method. We built the complete system and evaluated our robot's performance through extensive sets of experiments both in simulated and realistic environments

Contents

1 Introduction

1.1. Background

Robotic technology is improving rapidly and new opportunities for the application of robots within our society keep emerging. The current COVID-19 pandemic has significantly impacted daily life and caused millions of fatalities around the world. Various guidelines have been proposed to control the spread of the virus including measures such as social distancing and wearing face masks. The degree to which people follow these regulations has a significant influence on how well the virus can be contained and to what extent its negative effects can be minimized. If human individuals are tasked with ensuring that others are adhering to these regulations, they themselves risk contracting and spreading the virus. Thus, a technological solution that does not involve human contact is more desirable.

Different solutions have been proposed to monitor, analyze, and encourage people to follow the social-distancing guidelines. Closed-circuit television (CCTV) [1–3], cellular networks [4, 5], wearable devices and smartphones [4, 5], are examples of such systems. However, CCTVs can only be utilized for monitoring as they cannot give feedback to people. The smartphones and wearables-based solution can alert individuals, but this would require some form of tracking which poses privacy concerns for users.

Mobile robots with the ability to interact with humans can be suitable for desired social behavior encouragement (e.g., a robot can collect empty bottles if people throw them on ground and encourage them not to do so next time). However, using a single robot (e.g., spot from Boston Dynamics [6]) would be insufficient in most cases as it can only give feedback to a limited number of people within a certain time frame. An obvious solution to eliminate this limitation is to use multiple robot units or a *swarm* of robots. However, the main challenges of employing a swarm of robots for desired social behavior encouragement are the associated costs of manufacturing and maintenance. Therefore, this thesis analyzes the performance of a variety of hardware modules and algorithms to provide guidelines on designing such systems in a cost-effective manner with minimal resources. Towards this end, we developed a low-cost robot that has a unique vision system and a hybrid navigation stack.

1.2. Vision and Applications

The trend of Miniaturizing hardware and advances in artificial intelligence (think of your smartphone) give hope to designing low-cost robotic swarms. Swarm systems are meant to be scalable, robust, and easily deployable for various applications. These characteristics make them suitable for encouraging a desired social behavior in groups of people, as multiple robots can be deployed in a certain area allowing for a large scale of monitoring and feedback. To mass-produce robots, however, their design needs to be done in a cost-effective manner. In simple terms, a swarm designer needs to design the cheapest robot that meets the requirements. In this paper, we analyze and profile the performance of different processors and vision sensors to design a swarm robot for monitoring social distancing and giving people feedback (as an example of encouraging desired social behaviors).

1.3. Challenges

Developing such a system entails various challenges. Designing a swarm system that encourages social distancing requires balancing two conflicting requirements: (i) Social robots that interact with people and navigate in dynamic environments are often sophisticated and need costly hardware (ii) Forming a fleet of robots and mass producing them requires the individual robots to be cost-effective and simple.
Other challenges relate to the performance of our system. The robot should detect social distancing breaches in real-time and up to an acceptable range. Additionally, it should autonomously navigate towards breaches in a dynamic environment. However, traditional robot navigation usually requires appropriately configuring and tuning several packages (Figure 3.6) to effectively reach the target, rendering it challenging, especially when planning to deploy the navigation stack on various robots.

1.4. Research Focus & Objectives

This thesis project looks into the design, development and testing of an autonomous robot that encourages social distancing. The developed robot should efficiently detect breaches in social distancing, and navigate towards those breaches, encouraging people to respect the measures in place. The aim of this work is to complete a significant first step in the development of a swarm of such robots that would cover larger distances, hence providing a safer environment. To achieve our principal objective, the following research questions need to be answered:

- What does it take to develop a swarm robot to encourage desirable social behaviors in a cost-effective way?
- How can we design a long-range, low-cost vision system that can estimate the 3D coordinates of people in the scene?
- How can we develop a navigation stack that is generalizable and easily deployable?

It is important to note that the scope of this thesis project is bound to the design, development and testing of a single robot able to detect breaches in social distancing. The swarm system design is hence not included.

1.5. System Design

The first step in developing a robotic system consists of defining its mission profile and design. The mission profile will help in identifying the hardware and software requirements necessary to develop and deploy a robot able to encourage a desired social behavior. The following list outlines the different phases of the robot's journey:

- Phase 1: The robot is on standby in an indoor area, at position A (the origin), and is continuously checking the area for social distancing breaches.
- Phase 2: The robot detects a breach once it locates a group of people standing too close to each other (less than 1.5m apart).
- Phase 3: Once the breach is detected, its coordinates are processed by the robot. In case multiple breaches are detected, the robot locks onto the largest breach.
- Phase 4: The robot autonomously navigates towards the detected breach in the environment, while avoiding static and dynamic obstacles
- Phase 5: The robot reaches its target destination, letting the group know that they should disperse.
- Phase 6: The robot goes back to its original location, and resumes detecting social distancing breaches.

1.5.1. Requirements & System Architecture

Having defined the mission profile of our robot, we looked into the necessary requirement to accomplish the robot's intended objectives. A main aspect of the proposed approach is to have a scalable, modular and easily deployable robotic solution, with the ability to generalize across different platforms and environments. The main functionalities of this novel robot are the detection and navigation systems.

Breach Detection Requirements:- We suggest the use of Computer Vision (CV) algorithms for detecting social distancing breaches in an environment. Since we will be considering a relatively cheap robot, with limited capabilities, not all CV systems can be deployed on its processor. Consequently, it is important to analyze the requirements and performance of several detection algorithms as to find the most suitable one for our robot. The detection system should also be able to detect breaches in real-time and provide efficient detection for both short and long ranges in front of the robot, all the while being cost-efficient.

Navigation Requirements:- Developing a navigation system for an autonomous robot requires several considerations, especially when running it on a robot with limited resources. We should first avoid computationally demanding algorithms. Additionally, the system should be generalizable as to cope with unseen environments, and allow the robot to navigate while avoiding static and dynamic obstacles. Finally, the navigation should be easily deployable, without the need to manually fine-tune a large number of parameters.

1.6. Contribution

This thesis makes the following key contributions:

- We built a robotic platform for detecting social distancing violations.
- We experimented with different vision sensors and algorithms and developed a *compound algorithm* that doubles the effective range of the Intel RealSense depth camera for detecting social distancing breaches.
- We developed a hybrid navigation stack that combines the power of Deep Reinforcement Learning and a probabilistic localization system.
- We conducted extensive simulated and real-world experiments to evaluate the performance of our robot.

1.7. Thesis Outline and structure

The thesis report is outlined as follows:

- Chapter 2 presents the related work, for social distancing breach detection and robot navigation
- Chapter 3 shows the hardware and software stack of our social distancing tracker robot
- Chapter 4 evaluates and compares the performance of the robot's vision and navigation systems
- Chapter 5 concludes this thesis and proposes several recommendations for future research into this topic
- Chapter 6 is the appendix, and shows our overall ROS implementation of the system

2

Related Work

2.1. Social Distancing Breach detection

Following the worldwide increase in COVID-19 cases, measures such as social distancing have been found necessary to limit the spread of the virus [7, 8]. This brought forth recent research [4, 5] discussing how emerging technologies such as wireless and artificial intelligence (AI) can encourage social distancing to mitigate the impact of COVID-19.

Advancements in Deep Learning (DL) have made identifying social distancing violations using computer vision systems easier. Examples include using cameras to monitor social behaviors such as social distancing or wearing face masks [1–3]. Generally, such approaches often use CCTV cameras placed at specific locations, with predefined settings, along with the YOLO object detection algorithm [9] to identify breaches. They also leverage tracking algorithms such as SORT [10] and DeepSORT [11] to keep track of the identified pedestrians. Although such methods take advantage of an already deployed system, they do not encourage pedestrians to follow social distancing rules, but rather act as monitoring tools.

Other methods leveraging smartphones and wearables [12, 13] have also been proposed and are designed to warn people when they get too close to others. Yet, despite their potential positive contribution, these devices and applications often prompt privacy-related questions, discouraging people from using them.

Another line of research proposes using robots to limit viruses' spread [6, 14], by encouraging, for example, social distancing practice [15]. Social robots are flexible and easy to deploy systems meant to interact with humans and other robots to perform a specific task. Autonomous robots for detecting social distancing breaches have therefore been proposed. Fan et al. [16] introduced an autonomous surveillance quadruped robot that can promote social distancing in complex environments using LiDAR readings. However, the high cost of this system prevents its mass production. A cheaper option is presented by Sathyamoorthy et al. [17]. It combines the output from an RGB-D camera placed on an autonomous mobile robot with a CCTV camera placed in a room. Also, a thermal camera is used to detect COVID symptoms such as a fever. However, the range of the mounted depth camera in this robot is limited to 4m, and that of the CCTV camera to 3m, which severely limits the range, and thus the effectiveness, of the proposed mobile system.

Finally, in this work, we propose a compound low-cost vision system that can detect social distancing violations up to +12 m away from the robot.

2.2. Robotic navigation

2.2.1. Classical navigation methods:

Classical robotics navigation has benefitted from several advancements over the past few years. It is categorized into two types: deliberative and reactive navigation [18]. Deliberative navigation depends on a map of the environment and creates a global path for the robot to follow. As for reactive navigation, it consists of identifying a collision-free path based on the robot's instantaneous perception of the environment [18] and can therefore account for dynamic obstacles. Hence, robot navigation in static and dynamic environments often consists of several software packages working together, specifically, global planning (deliberative navigation) based on a static map, followed by local planning (reactive navigation) for dynamic collision avoidance [19]. However, the classical navigation approaches often require substantial tuning [20] to achieve a reliable per-

formance and tend to deteriorate when deployed on different robot models with other characteristics [18, 19].

2.2.2. Reinforcement learning:

Motivated by the success in the Reinforcement Learning domain [21], many researchers have started examining such techniques for mobile robots navigation by mapping input sensory data (images or LiDAR scans) to the output steering commands [22–24]. This is driven by the potential of having a navigation stack that can easily be ported to different robots, along with the possibility for online learning.

Tai et al. [22] were one of the first researchers to develop a mapless motion planner based on the Deep Deterministic Policy Gradient (DDPG) method, and to deploy it in the real world. As a result, the robot could navigate virtual and real unseen environments while avoiding obstacles using LiDAR scans as input. Long et al. [25] utilized the Proximal Policy Optimization (PPO) algorithm [26] and developed a safe and efficient collision avoidance policy for multiple robots. They also validated the policy in various simulated environments. However, being an on-policy algorithm has a significant impact on the robot learning with respect to sample efficiency. Off-policy learning algorithms are therefore typically seen as more suitable for complex robotics tasks due to their improved sample efficiency [27, 28]. Jesus et al. [29] replaced the DDPG with the Soft Actor-Critic [28] (SAC) DRL algorithm and showed its efficacy in simulation. Additionally, instead of using LiDAR for navigation, Kulhánek et al. [30] developed a camera-based navigation system. They extended a version of the batched A2C algorithm [21] and validated the system performance on a real robot [31]. However, considering the novelty of this approach (first attempt to move DRL visual navigation onto a real robot), it is still very limited: the robot was trained to move in a grid, using a discrete set of actions, and collision avoidance methods were also not considered.

In this work, we focused on off-policy DRL methods since they allow for better sample efficiency [28, 32]. We hence decided to train the off-policy DDPG algorithm, one of the most used algorithms for robot navigation. Then, we used the Soft Actor-Critic (SAC) algorithm to navigate our robot in simulation and reality. Compared to other methods, SAC seems to provide many advantages for moving a navigation system to the real world. Its stochastic policy encourages exploration and increases sample efficiency [28, 33]. In SAC, the actor aims to maximize the expected reward while maximizing entropy [28], hence learning various actions that can provide near-optimal behavior.

3

System Overview

Our robot must detect social distancing breaches in real-time and execute safe collision-free navigation towards the offending pedestrians. Figure 3.1 shows the hardware and software stack of our social distancing tracker robot.

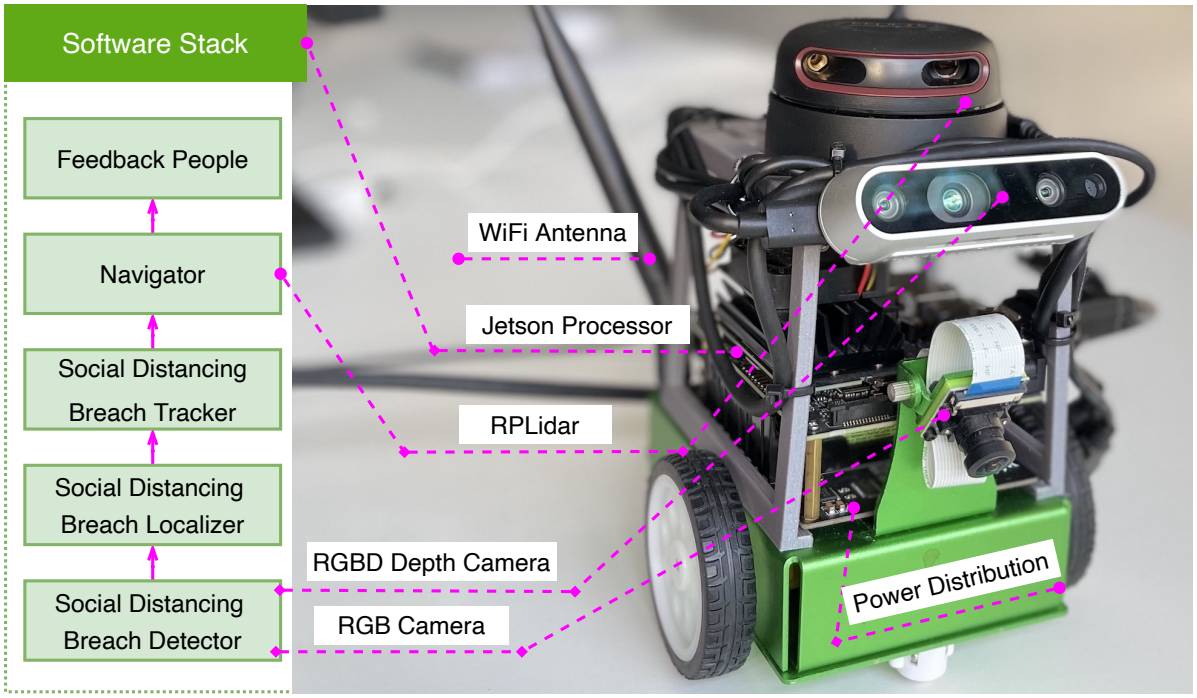


Figure 3.1: Hardware and software overview of a swarm robot.

3.1. Simulator

A simulator provides controlled environments for comparing different hardware modules and algorithms. As a result, it has become an essential part of developing robotics and deep learning applications. Using a robotics simulator thus provides many advantages [34] such as:

Providing large amounts of annotated data for AI tasks: The recent increase in the use of deep learning (DL), provided a major push towards using simulators for collecting large amounts of data in different environments.

Training Deep Reinforcement Learning algorithms: Considering that DRL training requires a large number of "trial and error" episodes where the robot interacts with its environment to learn a policy, training DRL agents is much faster in simulation, and requires less effort compared to the real world.

Accelerating the design cycle and reducing costs: Developing a robot requires going through a series of prototypes, whether using different hardware parts, or different algorithms, before reaching an acceptable one. This iterative process is time-consuming and expensive. However, when testing in simulation, we can try out different robot models and sensors in various environments without the costs and time needed for actual physical changes.
Providing a safe and controlled virtual environment for testing: Simulation plays a major role in providing insights into the system by testing its limitations, corner cases, and how it acts for different scenarios. Ensuring that testing is done in a safe environment, with complete control over the experiments, and without damaging the hardware. In our case, it allows us to check on the performance of our breach detection system, its limits, the safety of our navigation system, how the robot navigates in static and dynamic obstacles, etc.

There are many simulators to choose from, each with different levels of performance and realism. Table 3.1 summarizes the metrics that we used to select a simulator (in our case, Gazebo). In general, Gazebo seems to be the most adequate simulator for our use case. It is closely integrated with ROS and has a large and active community. It also supports a wide variety of robot models, whether by default or from third parties, and is often used by researchers when planning on training Reinforcement Learning tasks to be deployed in reality [35]. For an in-depth comparison we refer the reader to [35–39].

Simulator	Physics engine	ROS Integration	Robotic models	Documentation
Gazebo	Bullet, DART, ODE, Simbody can be added when building from source. Only the ODE physics engine is available by default	Default simulator used in ROS, with a large base of community-developed plugins and code	A diverse library of default robots that mainly includes wheeled and flying robots. Third-party robot models are available	A comprehensive documentation, step by-step tutorials and a large user community are available.
Argos	A 2D and a 3D custom built physics engines available by default	Complex and not properly documented and requires the use of 3rd party plugins such as "argos-bridge"	A relatively small library of robot(e-puck , eyebot , Kilobot , marXbot and Spiri)	Good documentation, but a small user community. Irregular development
Unity	PhysX by NVIDIA	Needs external integration packages, such as "ROS-TCP-Connector" and "ROS-TCP-Endpoint"	Relatively small library of robots	Well documented. Large community. The documentation for robotics development is still low, compared to its game development one.
Vrep	Bullet 2.78, Bullet 2.83, ODE, Vortex and Newton and Newton	Highly complex integration. Tools and libraries are needed such as "ROS Interface" , "V-Rep ROS Bridge".	Large variety of robots, including bipedal, hexapod, wheeled, flying and snake-like robots. Provides a large number of robot actuators and sensors	Good documentation, a large library of tutorials and a large user community
Webots	Customized ODE version	Using the standard ROS controller available by default or using a custom ROS controller for more flexibility	Variety of humanoid, mobile and multi-legged robots	Good documentation and tutorials available on the Webots website

Table 3.1: Robotics Simulator Selection.

Specifications	**Raspberry Pi 4 and Intel Movidius**	**Raspberry Pi 4 and Coral Accelerator**	**Jetson Nano**	**Jetson Xavier NX**
GPU performance	Up to 150 GFLOPs	Up to 4 TOPs (INT8)	472 GFLOPs (FP32)	21 TOPS (INT8)
HW accelerator	Broadcom Video Core VI (32-bit) + Myriad X VPU	Broadcom Video Core VI (32-bit) + Google Edge TPU coprocessor	128-core NVIDIA Maxwell GPU	384-core NVIDIA Volta GPU with 48 Tensor Cores
CPU	Quad-core ARM CortexA72 64-bit @ 1.5 GHz	Quad-core ARM CortexA72 64-bit @ 1.5 GHz	Quad-core ARM Cortex-A57 MP-Core processor	6-core NVIDIA Carmel ARM v8.2 64-bit CPU 6MB L2 + 4MB L3
Memory	8 GB LPDDR4 + 4 GB LPDDR3	8 GB LPDDR4	4 GB 64-bit LPDDR4 25.6 GB/s	8 GB 128-bit LPDDR4x @ 1866MHz 59.7GB/s
Storage	Micro-SD	Micro-SD	Micro-SD or 16 GB eMMC 5.1 flash	16 GB eMMC 5.1
Price	100 €	100 €	100 €	326 €

Table 3.2: The main specifications of the embedded hardware for edge AI taken into account.

3.2. Hardware

3.2.1. Processors

To respect swarm design principles (e.g., simplicity) and meet the application requirements, we considered a variety of hardware modules starting from the lowest cost options before expanding to more expensive and powerful platforms. Although Arduino can run simple machine learning (ML) models, it is not fit for processing images and detecting social distancing breaches. The Raspberry Pi is much more powerful, but lacks a hardware accelerator (e.g., GPU or TPU) that can significantly speed up DL calculations. This limitation can be eliminated by adding a hardware accelerator such as Coral [40] from Google or Movidius [41] from Intel. While the Jetson Nano [42] from NVIDIA has similar performance to the Raspberry Pi with hardware accelerator, it is also a part of the Jetson family that offers other more advanced processors such as Jetson Xavier NX and AGX [42] allowing for manageable up-scaling. From a developer's perspective having the ability to easily port the code to more capable processors is a very valuable feature and in our case is the determinant factor. Table 3.3 puts these processor side by side highlighting the differences and important considerations.

3.2.2. Vision Sensors

An RGB-depth (RGB-D) camera captures RGB images, and their depth information on a per-pixel basis [43]. Therefore, it is a natural choice to be considered when estimating the 3-dimensional (3D) coordinates of objects in images. Examples of such cameras include Stereolabs ZED 2 [44], Asus Xtion Pro [45], Microsoft Kinect [46], and Intel RealSense [47]. However, despite their accurate depth estimation, depth cameras have several shortcomings compared to standard RGB cameras (e.g., a Raspberry Pi camera). For example, they have limited depth range and field of view. They are also more expensive. Consequently, researchers have proposed algorithms that estimate depth information from regular RGB images. For example, MonoPSR [48] and Monoloco [49] algorithms show promising 3D coordinates estimation of people in images captured by relatively low-cost RGB cameras. To figure out the most suitable camera for our robot, we show in Table 3.3 a list of various RGB and RGB-D cameras.

From Table 3.3, the Intel RealSense D435i and Stereolabs Zed2, appear to be the most suitable depth cameras for our system, as they provide a high measuring range. The Zed2 is based on passive stereo vision, meaning that it doesn't emit any laser or IR light like active sensors, unlike the D435i which is based on active stereo vision. Since we are planning on deploying our robot in an indoor environment, an active stereo vision has proven to be less sensitive to indoor textures and is often the preferred choice for the 3D perception of indoor scenes [50, 51]. Finally, we decided to use the Intel RealSense D435i for our robot. The advertized mea-

Camera	**Measuring range (m)**	**FPS**	**Depth FoV** ($H° \times V°$)	**Cost (€)**
Xtion PRO Live	0.8–3.5	60	58×45	199
Kinect v2	0.5–4.5	30	70.6×60	200
RealSense D435	0.2–10.0	90	91.2×65.5	179
RealSense D435i	0.11–10.0	90	85×58	199
Stereolabs Zed 2	0.2-20	60	110×70	400
Raspberry Pi camera	-	30	-	35

Table 3.3: Vision sensors comparison.

LiDAR	**Detection range (m)**	**Scanning method**	**Angular range**	**Scanning Freq. (Hz)**	**Cost (€)**
Slmatec-RPLiDAR A2M8	0.15-8	2D	360°	5-15	333
Slamtec-RP LiDAR S1	Black Object: 0.2-10 White Object: 0.2-40	2D	360°	5-15	569
Hokuyo-UST-20LX	0.2-20	2D	270°	40	2280
Robosense-RS-LIDAR-16	0.2-150	3D	360°	5-20	3440

Table 3.4: Laser sensors comparison.

suring range of 10m is high compared to other alternatives, and is based on active stereo vision. Additionally, it gives a high FPS count, along with a large Depth Field of View (FoV). Overall, it seems more cost-efficient than the two times more expensive Zed2.

3.2.3. Laser Sensors

Based on the literature on robotic navigation, LiDARs are becoming an increasingly popular component for autonomous navigation using DRL, for detecting the environment and autonomously positioning the robot within it. Several considerations should be kept in mind when choosing a LiDAR for our system. For example, detection range, depth requirements (2D vs 3D LiDARs), angular range and scanning frequency. Table 3.4 shows some of the LiDARs which could potentially be used for our navigation system.

In addition to detecting objects, 3D LiDARs produce a detailed point cloud that could determine an object's shape and depth, as opposed to 2D LiDARs, which restrict the detection of obstacles up to a single plane, and do not detect objects below or above that plane [52]. However, 3D LiDAR systems are expensive and hence not scalable for a swarm of robots, while 2D LiDARs may not provide robust navigation due to their detection limitations. Considering that breaches are detected using a camera and that camera-based navigation systems using DRL are a relatively growing field, relying on the camera for navigation would be the most suitable solution for our robot in the future to decrease costs further.

In this thesis, the robot will be navigating in a flat environment (indoor setting), and we are only considering navigation in a 2D plane. Hence, using a 2D LiDAR for navigation seemed sufficient to prototype our robot as an intermediate step towards visual navigation. There are various types of 2D LiDARs, and they often increase in price when the detection distance and scanning frequency increase. Therefore, considering the significant price difference between the Slmatec-RPLiDAR A2M8, the RPLiDAR S1, and the Hokuyo UST-20LX, and taking into account the detection range, the Slmatec-RPLiDAR A2M8 seemed like a good choice for our system.

3.2.4. Robot Body

We selected the JetBot AI kit for developing our AI robotic system due to its low price (≈ 300 €) and its growing usage for AI tasks [53–55] . This differential drive vehicle consists of two wheels in the front and two caster wheels. It has two motors that can be independently driven in both directions. The JetBot also includes an IMX219 8MP camera mounted to the front. It is powered by an NVIDIA processor, allowing it to run AI tasks such as facial recognition, object tracking, auto line following, and collision avoidance. Additionally, it

supports the Robot Operating System (ROS) and comes with existing ROS 1 and ROS 2 Software Development Kit (SDK)s to collect images from the camera and teleoperate the robot.

3.3. Software

The software architecture of our robot can be divided into two main modules: the breach detection and tracking module and the navigation module. Each module is constituted of sub-modules that are implemented as ROS (Robotic Operating System) nodes.

3.3.1. Breach Detection Algorithms

We implemented three methods for detecting social distancing breaches. The first one uses an RGB-D image, the second leverages an RGB one, and finally, a combination of both approaches for a higher detection range.

Breach Detection from an RGB-D image:-

Our method starts by detecting the 2D coordinates of pedestrians identified in an image. Considering that different DL models are available for object detection, we decided to rely on YOLOv3[9], a real-time object detection algorithm, often used with mobile robots. YOLOv3 consists of a feed-forward convolutional neural network, which takes an RGB image as input. It then outputs a set of bounding boxes around the detected objects, along with their respective categories (people in our case) and accuracy scores. YOLOv3 has 53 convolutional layers, in successive layers of size 3 x 3 and 1 x 1, along with residual blocks [9].

To integrate the YOLO detection into our ROS system, we used the YOLOv3-ROS package [9, 56]. This package implements a wrapper for YOLOv3 and takes RGB images from the robot's camera as input. YOLOv3 ROS subsequently publishes three different topics: an array of bounding boxes that gives information on the position and size of the bounding box in pixel coordinates, the category of the detected object, and its confidence score.

Once we identify the pedestrians, we need to determine their actual positions in the 3D space. This is done by fusing the RGB image with the 3D point cloud information extracted from the depth image. A point cloud is a large set of 3D measurements representing an object. Each point is identified by its x,y, and z coordinates. The Intel Realsense provides an organized point cloud dataset, which resembles an image, and has a structure similar to a matrix, making it easily accessible. Taking the point cloud data and the 2D boxes as input, we start by iterating over the bounding boxes, calculating the center of each. We then iterate over the point cloud data and choose the points closest to the center of the current bounding box, meaning 3D localization depends on the center of an identified human. The final identified point cloud provides us with the needed information on each pedestrian: detection accuracy and 3D coordinates.

Once we generate the 2D bounding boxes around the identified pedestrians and compute their 3D positions, we use the Simple Online And Real-time Tracking (SORT) algorithm [10] to track them. First, SORT takes as input the 2D bounding boxes. Starting with the first image is receives, it detects the identified pedestrians and gives a unique ID for each. SORT then propagates these detections onto each new frame and uses a Kalman filter [57] with a linear constant velocity model to predict the new positions of the tracked people. Once SORT receives the new 2D bounding boxes predictions for the current frame, it compares them to the people it is tracking and creates a cost matrix consisting of the Intersection over Union (IoU) between each new detection and the previously tracked people. The IoU is an evaluation metric that specifies the amount of overlap between the predicted and ground truth bounding box. New detections are then associated with the previous tracks using the Hungarian algorithm [58]. SORT creates a new track whenever a new object is detected.

We end up with unique IDs for each identified pedestrian. After tracking the identified pedestrians for 20 frames, we average out their 3D coordinates and calculate the distance between them using the Euclidean distance. Assuming two pedestrians P_a and P_b, the Euclidean distance function is given by:

$$dist(P_a, P_b) = \sqrt{(P_x^{P_a} - P_x^{P_b})^2 + (P_y^{P_a} - P_y^{P_b})^2}$$

In case the distance dist(P_a , P_b), between the centers of the bounding boxes is < 1.5m , the robot reports a breach for that pair of individuals. We repeat this process pairwise for all the detected individuals. As a result, we obtain a list of breaches containing the different groups of non-compliant pedestrians. We choose the largest group, compute its middle coordinates, and send it to the robot navigation module (Figure 3.2).

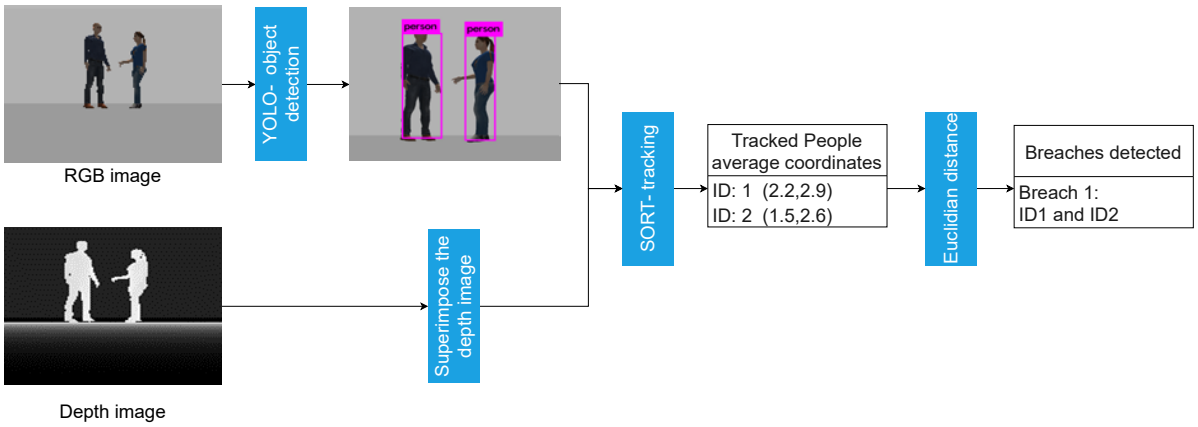


Figure 3.2: RGB-D breach detection

Breach Detection from an RGB image:-
The second method approximates pedestrians' 3D coordinates from monocular RGB images [49] . It also outputs their body orientation which can be an important metric for analyzing social distancing breaches. Bertoni et al. [59] proposed this approach in "Perceiving Humans: from Monocular 3D Localization to Social Distancing". It uses a low-cost RGB camera to extract humans' 3D locations, along with their body orientations, and identify social distancing breaches accordingly.

The overall detection system takes as input a monocular RGB image. It processes it into a set of 2D joints using two pose detectors: Mask R-CNN [60], which works top-down, and OpenPifPaf [61], which works bottom-up. The 2D pose detectors can be thought of as independent modules from the rest of the detection system, since 2D joints from any pose detector can work. Then, the MonoLoco [49] algorithm takes these 2D joints as input and outputs the 3D locations, orientations, and dimensions of the detected people together with the localization uncertainty. For that, the algorithm uses a deep, fully-connected neural network (DNN) with six linear layers of 256 output features. The DNN uses dropout after every fully connected layer and includes batch normalization and residual connections. The output values are analyzed to discover F-formations (spatial patterns constructed during interactions between two people or more) and evaluate social distancing breaches. Finally, the system publishes an approximation of each pedestrian's x,y,z coordinates identified in the image, along with its status (breaching social distancing or not). The overall system architecture is presented in Figure 3.3.

Considering that this system is to be deployed on a robot and that there may be more than one social distancing violation in an environment, we extended this method to fit our needs. Once all the social distancing violations are detected, the breaching pedestrians are grouped based on their coordinates. We then compute the middle coordinates of the largest group identified (the one with the most social distancing violations), and send it to the robot navigation module.

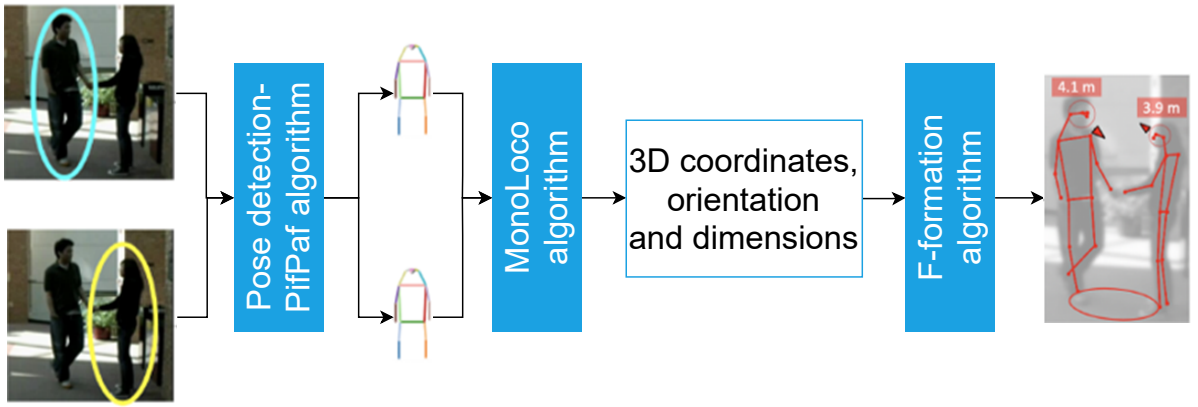


Figure 3.3: RGB breach detection [59]

Compound vision system:-
The RGB-D detection system depends on an RGB-D camera, with an effective range of $\approx 6m$ based on our experiments. From Figure 3.4, only the two pedestrians standing at 4m away from the camera are detected by the depth sensor. As for the RGB detection system, based on the literature, it can detect breaches between 5 and 30m [59]. Hence, we proposed a method that combines both approaches to effectively detect social distancing breaches without depending on more expensive vision hardware. Our vision system scans the scene for people twice: (i) Using RGB-D images, the robot localizes nearby people, and (ii) using pure RGB images, it estimates the localization of people that cannot be detected in the first stage (Figure 3.5).

Figure 3.4: RGB-D limited detection range

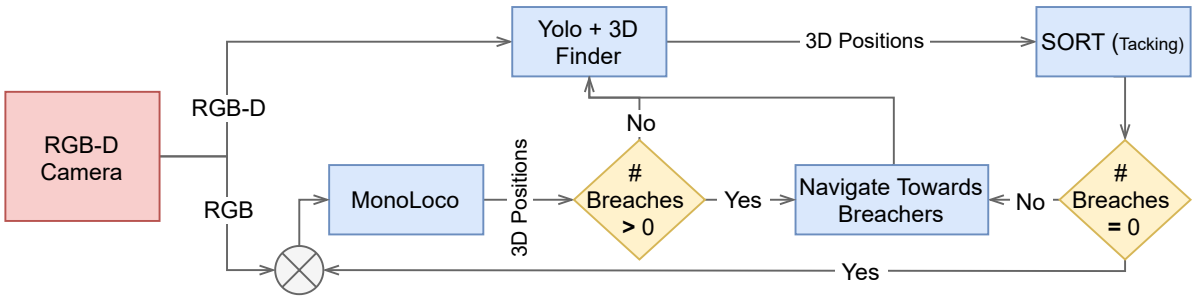


Figure 3.5: Proposed vision system structure

3.3.2. Robot Navigation Algorithms

Autonomous navigation is an essential aspect of mobile robotics. Building an autonomous mobile robot requires a reliable self-localization system, robust navigation, and static and dynamic obstacles avoidance. Once our robot detects social distancing breaches, it must navigate autonomously towards the identified coordinates. As such, we decided to assess several robotics navigation methods and identify the most suitable one for our system.

Odometry:-
To properly navigate, a robot needs to self-allocate and keep track of its position and orientation over time. Self-localization is traditionally done using the Global Positioning System (GPS). However, while it can often provide accurate positioning, GPS is not always reliable, especially for indoor navigation since walls and other objects attenuate radio signals [62]. Therefore, the research community has been investigating self-contained odometry approaches. Different methods exist for self-contained odometry based on the data used to estimate the position. Considering that we are already using a LiDAR for navigation and a camera for detecting breaches and aiming to reduce our costs to a minimum, we decided to either use laser or visual odometry.

LiDAR odometry approximates the position and orientation of a robot by tracking the laser patterns re-

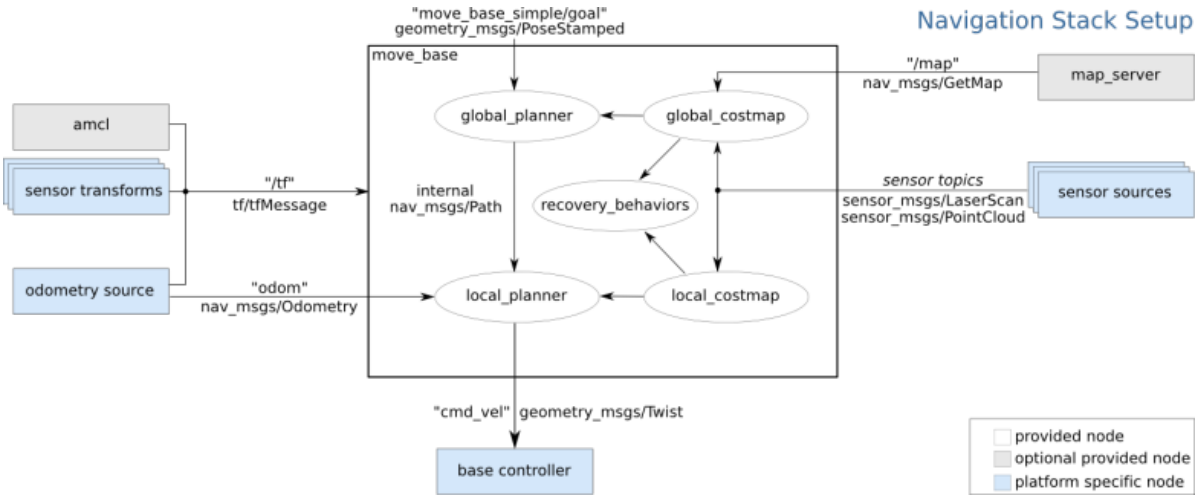


Figure 3.6: ROS Navigation stack setup [67].

flected from surrounding objects [63]. It computes the LiDAR's motion between two consecutive sweeps and uses the Iterative Closest Point (ICP) algorithm to align the newest LiDAR scan with the previous ones.The main drawback of LiDAR odometry is that it is difficult to implement on a resource-constrained platform [64]. Moreover, getting an accurate scan from transparent objects such as glass, is very challenging.

Visual odometry estimates the position and orientation of a robot by examining the variations caused by the camera's movement over a series of images [63]. Visual odometry presents many challenges. It is computationally expensive; it also depends on the image condition and quality: lighting, blurriness, etc. [65]. Finally, it suffers from drifting since it works by incrementally computing the camera's path, which leads to incremental errors between frames.

For our robot,we first tested the "rtabmap_ros" package [66], which outputs the robot's visual odometry from RGB images. However, computing the visual odometry was computationally heavy. The computed odometry was off and did not align with the robot's actual position. As for the LiDAR odometry, packages such as "Hector-SLAM" and "rtabmap_ros", seemed often used for robots. Therefore, after testing, we were able to estimate the robot's position and orientation using either of these laser odometry packages, even when using the Jetson Nano.

Classical ROS Navigation Stack:-
The ROS Navigation stack [67] consists of several software packages to safely navigate a robot from point A to B (Figure 3.6). It includes mapping, localization, and path planning. First, the environment map is built using ROS packages such as "gmapping" [68] or "hector_slam" [69]. Then, the map_server package publishes the map to other ros nodes. The Navigation Stack leverages the "amcl" [70] package to track the 2D pose of a robot against the map. AMCL [71] is a probabilistic localization system for robots, based on the Adaptive Monte Carlo Localization (AMCL) algorithm. Finally, planning is performed via packages such as "move_base", which consists of a global and a local planner. The global planner builds a path towards the given target over the static map. As for the local planner, it recalculates the path to avoid dynamic obstacles [72].

Mapless Autonomous Navigation:-
Our robot has to navigate crowded environments with static and dynamic obstacles, so finding a collision-free path is necessary to reach its target. As previously mentioned, we decided first to train the off-policy DDPG algorithm [22], one of the most used for robot navigation. Then, we used the Soft Actor-Critic (SAC) [29, 73] to navigate our JetBot in simulation and reality.

State Space:- The environment is observed through 10 laser range findings emitted from -90°to 90°in front of the robot. These measurements are combined with the angular and linear velocity and the relative position and angle of the robot to the target, and form the input state to the DRL agents. Figure 3.7 shows the input and output of the DDPG agent.

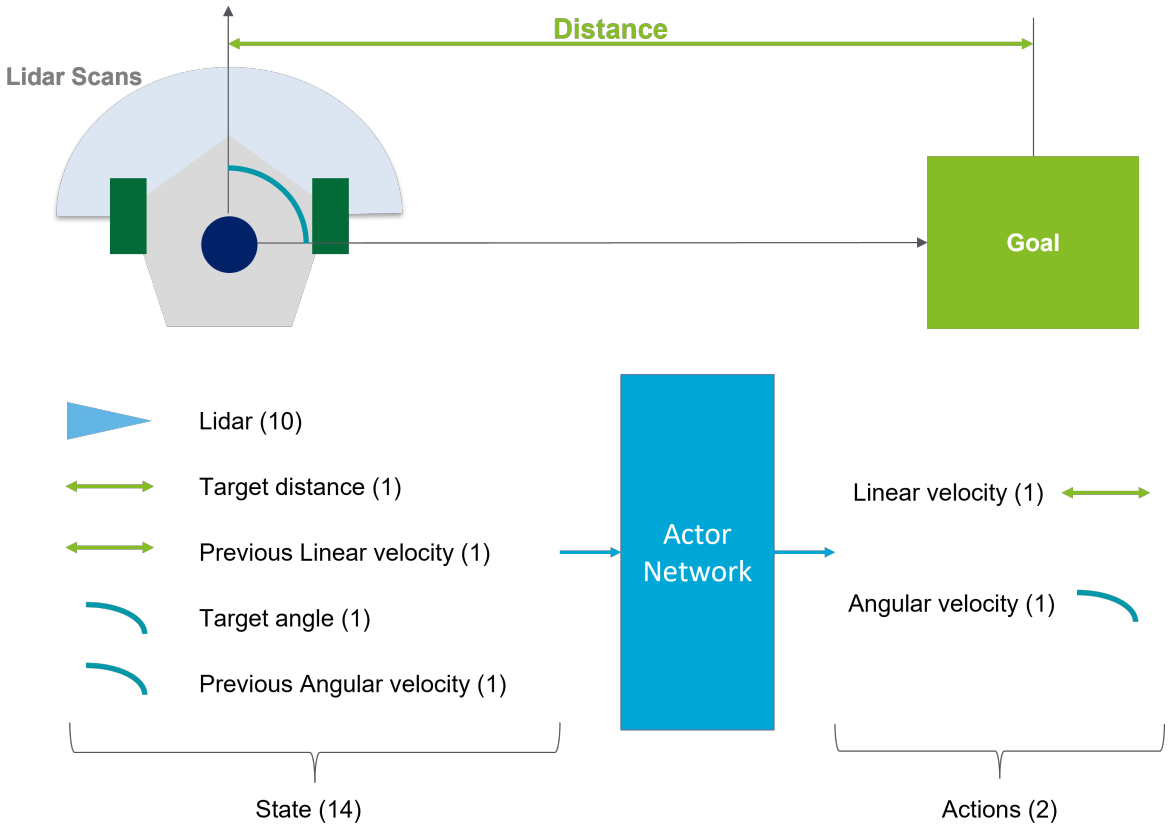


Figure 3.7: DRL state space when running inference on the Jetson processor.

Action Space:- Both the DDPG and SAC agents have actor-critic network architectures that act in a continuous action space [74]. The action space has two dimensions: the angular and linear velocities. The angular velocity is limited to [-2,2] rad/s, while the linear velocity is limited in the range [0,0.2] m/s.

Deep Deterministic Policy Gradient Network (DDPG):- The DDPG [74] network consists of three fully-connected neural networks layers with 512 nodes each. The rectified linear unit (ReLU) activation follows each layer. The output layer produces two action parameters representing the robot's linear velocity a_1 and angular velocity a_2. A hyperbolic tangent $tanh$ activation function is applied to the angular velocity to limit it in the range [-2,2] rad/s, and a $sigmoid$ activation function is applied to the linear velocity output layer to limit it in the range [0,0.2] m/s. The critic-network evaluates the Q value of the state-action pair Q(s, a). The critic first uses a pair of the state s and action a vectors as input to two fully connected networks, and concatenates their outputs. This is followed by two fully-connected neural network layers, which output the corresponding Q value. The full network architecture is visualized in Figure 3.8. The network configuration used for DDPG is initially proposed in [22, 29], with minor modifications to the critic, as shown in Figure 3.8.

Soft Actor-Critic Network (SAC):- SAC consists of three networks: an actor and two critic networks, trained independantly for computing the Q-value. The minimum Q-value is then used to update the policy [75]. The network structure of SAC is shown in Figure 3.9. The actor is composed of 2 fully connected neural network layers with 510 nodes each, and generates the mean and the log standard deviation, used to output the angular and linear velocities sent to the robot. A hyperbolic tangent $tanh$ activation function and the clip operation are applied to limit the linear velocity output layer in the range [0,0.2] m/s, and the angular velocity in the range [-2,2] rad/s. The critic network outputs the Q-value for the current state and action. It uses three fully-connected neural networks layers to process the inputs. The network configuration used for SAC is initially proposed in [73], however, the value network was omitted, as mentioned in [75].

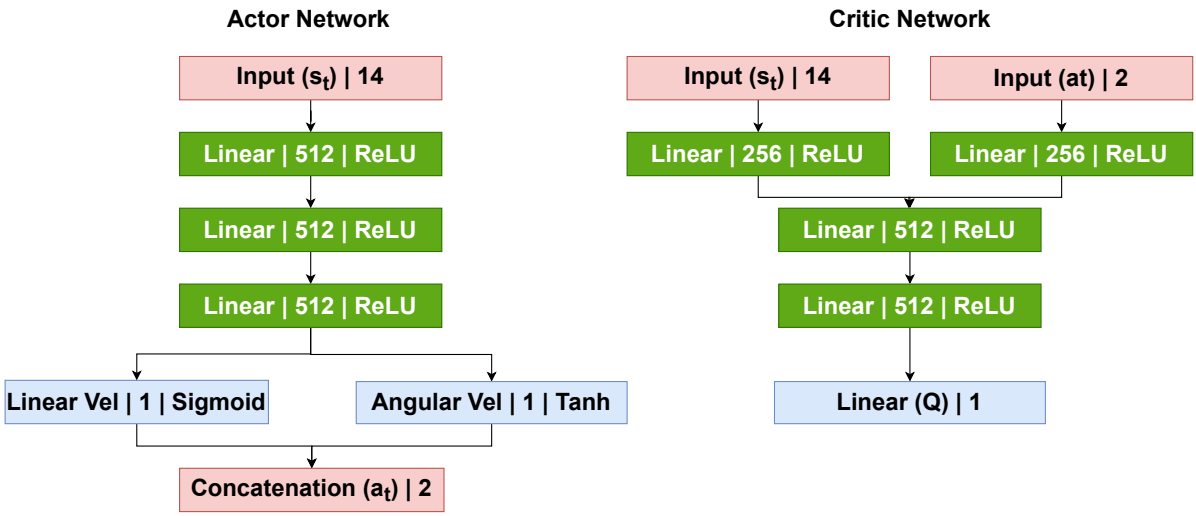


Figure 3.8: DDPG network structure [22].

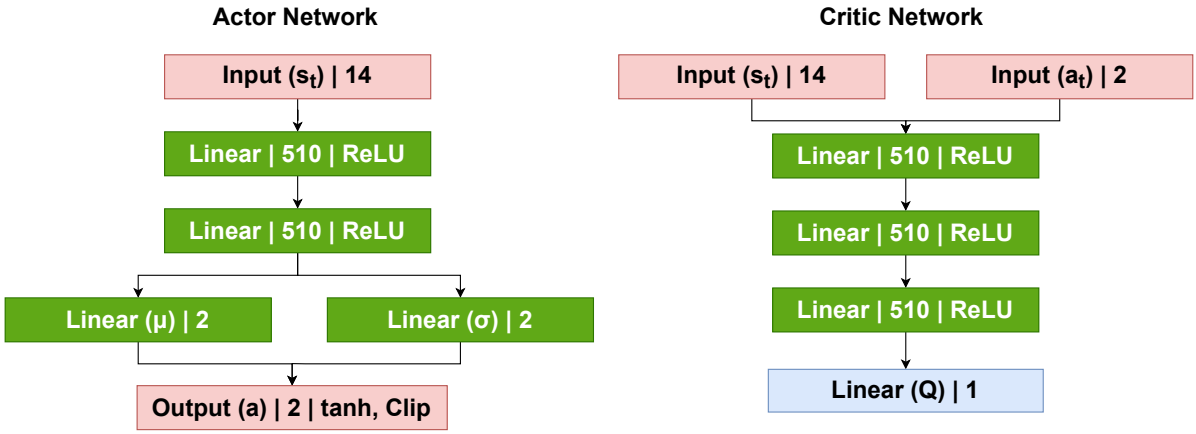


Figure 3.9: SAC network structure [73, 75].

Reward function:- A common reward function is used for both DRL methods, as to enable the agent to learn desired behaviors in given situations.

$$r(s_t, a_t) = \begin{cases} r_a \text{ if } D_t < T \\ r_c \text{ if } min_c < L_t \\ r_{d1}(D_{t-1} - D_t) \text{ if } (D_{t-1} - D_t) > 0 \\ r_{d2} \text{ if } (D_{t-1} - D_t) \leq 0 \end{cases}$$

There are four different rewards possible for our robot. First, if the robot reaches its goal within a specific distance threshold T, it receives a large positive reward. If the LiDAR reading L_t is smaller than a set minimum min_c, the robot has collided or is about to collide; therefore, it gets a big negative reward. The remaining rewards depend on the robot's distance to the target coordinates. If the robot gets closer to its target in a single step, it receives a positive reward, proportional to its progress. Otherwise, it is penalized with a negative reward.

Hybrid Autonomous Navigation:-
When deploying the mapless DRL navigation methods into the real world, we noticed that the LiDAR odometry often gets lost. Whenever this happens, the navigation fails and the robot is unable to reach its target. Hence, we developed a hybrid system, which combines navigation using a map and mapless navigation using DRL. We proposed this alternative as a middle ground between both approaches to solve the limitations we

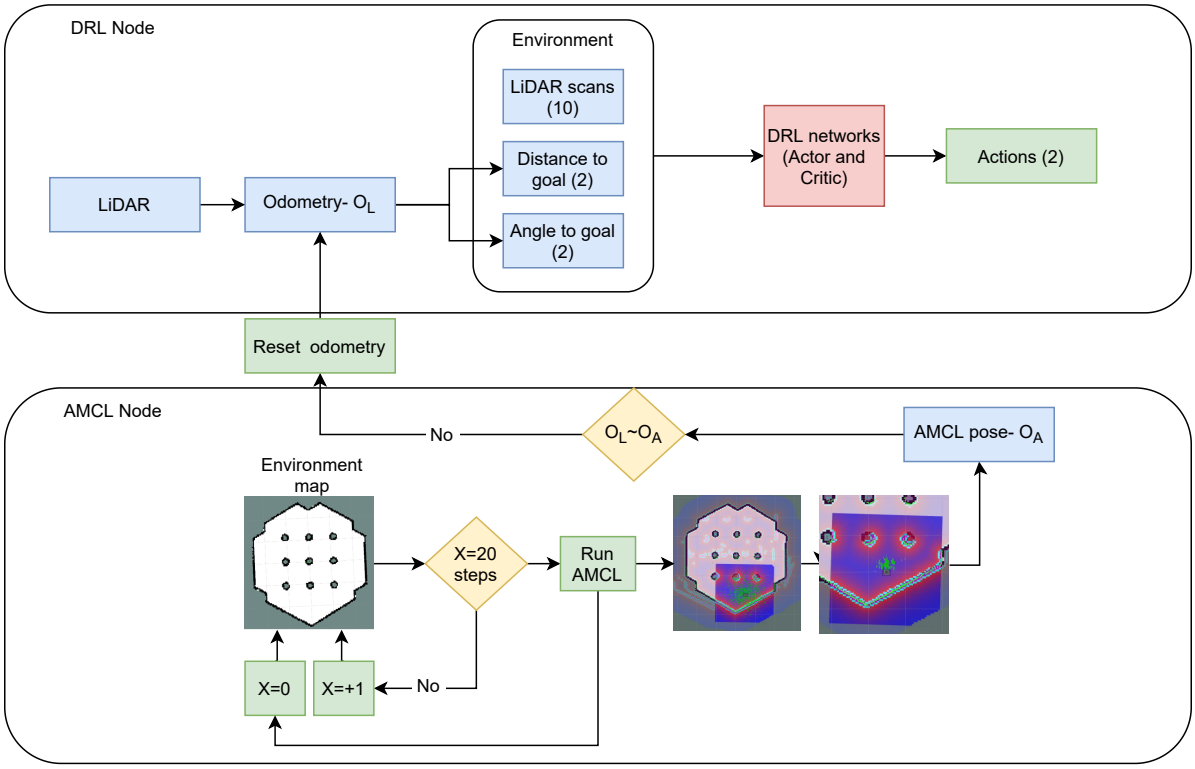


Figure 3.10: Hybrid navigation structure

encountered in each. Our approach first consists of the robot navigating towards a target using mapless DRL navigation and localizing itself using LiDAR odometry. Then, for every X step (in our case, X = 20), the robot compares its LiDAR odometry to its pose estimated using the Adaptive Monte Carlo Localization (AMCL) [71] method. If the difference between the two positions is higher than a preset threshold, the robot reinitializes its pose and orientation using AMCL and the given map of the environment. The overall structure of the hybrid navigation stack is shown in Figure 3.10.

4

Results

In this section, we evaluate and compare the performance of the different configurations of our robot's vision and navigation systems.

4.1. Vision System

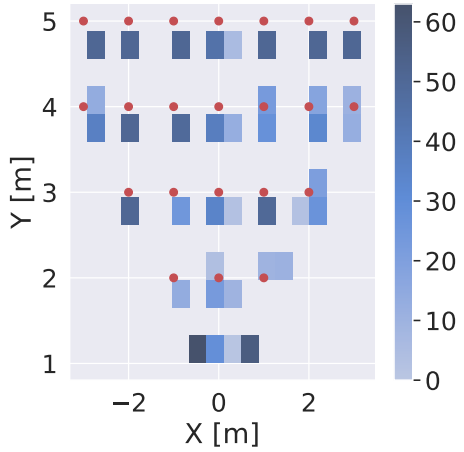
This set of experiments compares the performance of a depth camera (RGB-D) and a standard RGB camera in estimating the location of people in the scene and detecting social distancing breaches.

4.1.1. Experiment setup

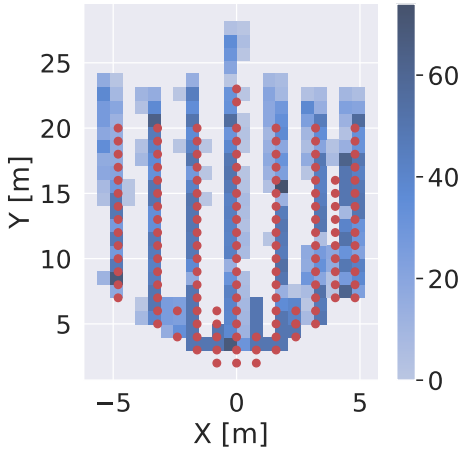
We conduct these experiments in both a simulated and real environment. For the person's location estimation, we moved the person away from the camera by 1m for every new experiment and took 50 pictures at each location. Additionally, we sampled a few other coordinates by moving the pedestrian vertically by 0.8m. We repeated this experiment until the camera could not detect any pedestrian, i.e., 23m for the RGB camera in simulation with 15m in reality and 5m for the RGB-D camera in simulation with 7m in reality. The red circles in Figure 4.1 show the exact coordinates at which our pedestrian was standing.
For the breach detection accuracy, we placed two pedestrians at various coordinates in the scene while ensuring they were in the camera's FOV and took 50 pictures at each location. When placing the pedestrians, we confirmed they breached social distancing rules in half of these locations to get an equal number of breaches and non-breaches readings. For detecting people using the RGB-D camera, we used the YOLOv3 model [9] trained on the COCO dataset [76]. As for the standard RGB camera, we used the Monoloco model [59], trained on the KITTI dataset [77].
Finally, we profiled the execution of these algorithms on different processors, namely, the Jetson Nano and Jetson Xavier NX.

4.1.2. Person Location Estimation

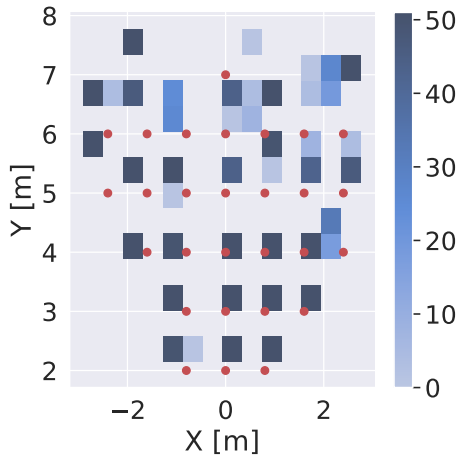
This experiment examines the accuracy of estimating the 3D location of a person using the RGB-D and RGB cameras. We compare the accuracy of each used vision system by calculating the average localization error (ALE) between the actual and estimated locations. Figure 4.1 showcases the ground truth locations of the identified pedestrians plotted as red circles, along with the corresponding heatmap of their estimated locations. As for Figure 4.2, it shows the ALE, along with its confidence interval for each of the RGB-D and RGB breach detection algorithms.



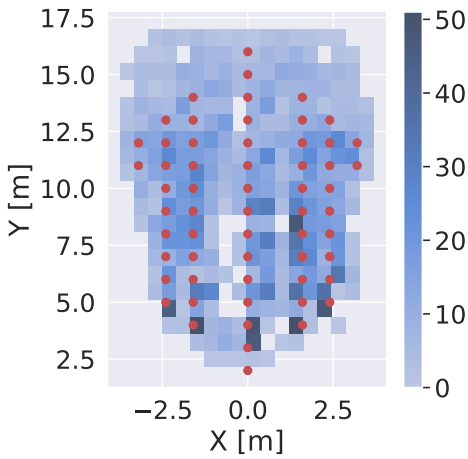
(a) RGB-D simulation



(b) RGB simulation

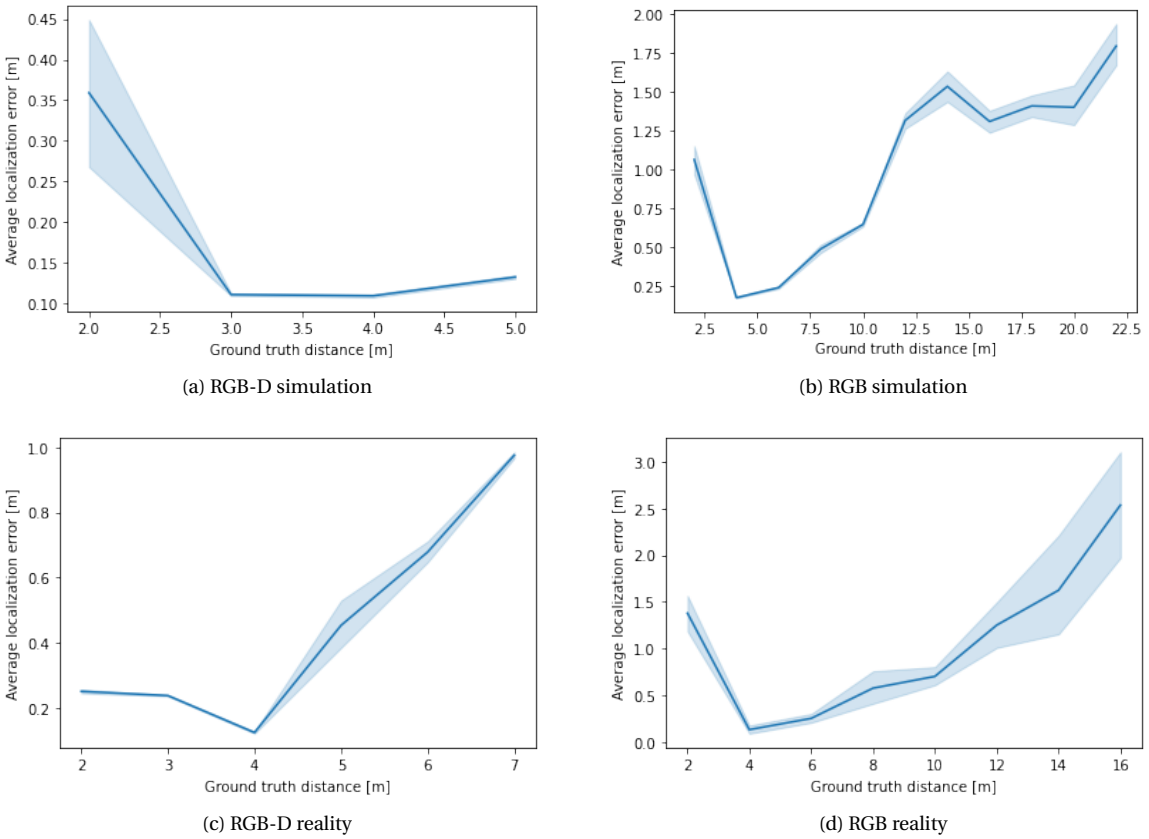


(c) RGB-D reality



(d) RGB reality

Figure 4.1: Heatmap showcasing the ground truth measurements (red circles) versus the approximated ones (blue squares).



(a) RGB-D simulation

(b) RGB simulation

(c) RGB-D reality

(d) RGB reality

Figure 4.2: Average localization error (ALE) with confidence interval as a function of distance: RGBD and RGB based system.

Simulation:-
From Figures 4.1a and 4.1b, we observe that the RGB camera allows our robot to detect people up to 23 m in simulation, as opposed to the RGB-D camera, which only allows detecting pedestrians up to 5m. When analyzing Figures 4.2a and 4.2b, the RGB-D system provides accurate coordinates approximation starting from a minimum threshold of $\approx 2.5m$, while the RGB one starting from $\approx$ 4 m. It is important to note that before the 5 m mark, it seems that the RGB system has nearly double the ALE of the RGB-D one.

We generally observe that the RGB-D detection system is optimal when people are between 2.5 and 5 m, as the ALE is at its lowest. As for the RGB system, its ALE increases up to 2.0 m when pedestrians are at 23m. Hence, if we assume that an ALE of 1m is still acceptable for selecting the target towards which a robot should navigate, the RGB algorithm would best suit people standing between $\approx$ 5 and 12m.

Reality:-
From Figure 4.1c, we notice that the RGB-D system can detect and approximate pedestrian coordinates up to 6m. In comparison, the RGB system can estimate coordinates up to 15m. When assessing the ALE from Figures 4.2c and 4.2d, it seems that the RGB-D camera can detect pedestrians standing at less than 5m, with an ALE < 0.4m. As for the RGB system, it starts with an ALE of $\approx 0.4m$ when pedestrians are standing at 5m and goes up to 2.5m when the pedestrians are at $\approx 15m$. Additionally, the variation in the coordinates estimation is high in the RGB system when compared to the RGB-D one as seen in Figures 4.1d and 4.2d, especially for distances above $\approx 12m$. A wide confidence interval means that the dispersion is high and that the algorithm approximates the 3D coordinates of pedestrians with more uncertainty and larger error margins.

As conjectured in the simulation, at short distances from the robot $\leq 5m$, the RGB-D detection algorithm is the better choice for approximating the 3D coordinates of pedestrians. And from $\approx 5m$ onwards, the RGB detection is a better fit. Considering our assumption that an acceptable ALE is $\leq 1m$, both algorithms could be leveraged in one system, providing us with a detection ranging from 2 to 12m.

4.1.3. Breach Detection Accuracy

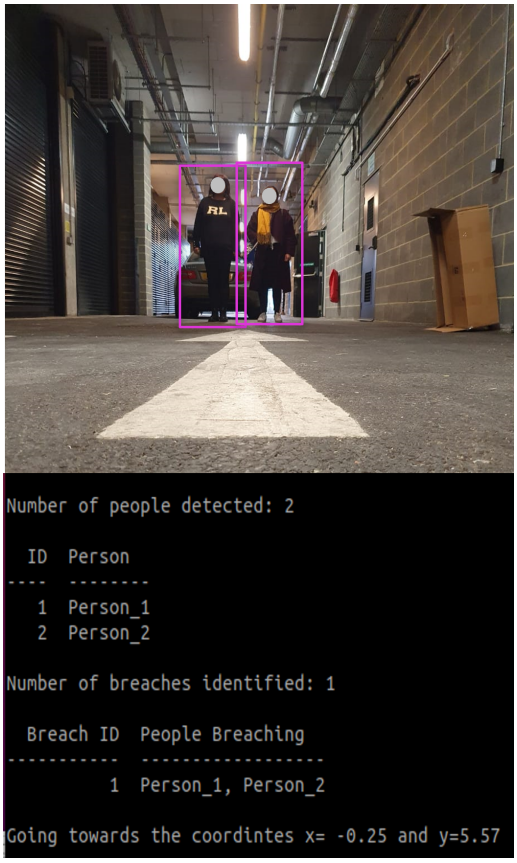
This experiment compares the performance of the RGB-D and RGB-based vision systems in detecting social distancing violations. We treat the problem as a binary classification task and evaluate the detection accuracy, recall, and precision. Figure 5.1 represents the confusion matrices for the RGB-D and RGB breach detection. As for Table 4.1, it compares the accuracy, precision, and recall between each of the two methods.

Simulation:-
From Table 4.1, we notice that the accuracy is equal to 96% and 92% for the RGB-D and RGB systems, respectively, all the while having high precision and recall. Both classifiers can, therefore, accurately classify social distancing breaches in simulation. However, the RGB detection system seems to classify a relatively large number of breaches as safe compared to the RGB-D one, meaning its recall is somewhat lower. A reason for this is the significant increase in the ALE for the RGB system, along with the widening of the confidence interval for large distances (> 13m - Figure 4.2b). Hence, when running the experiments, we noticed that a large number of false negatives were detected for distances above 15m.

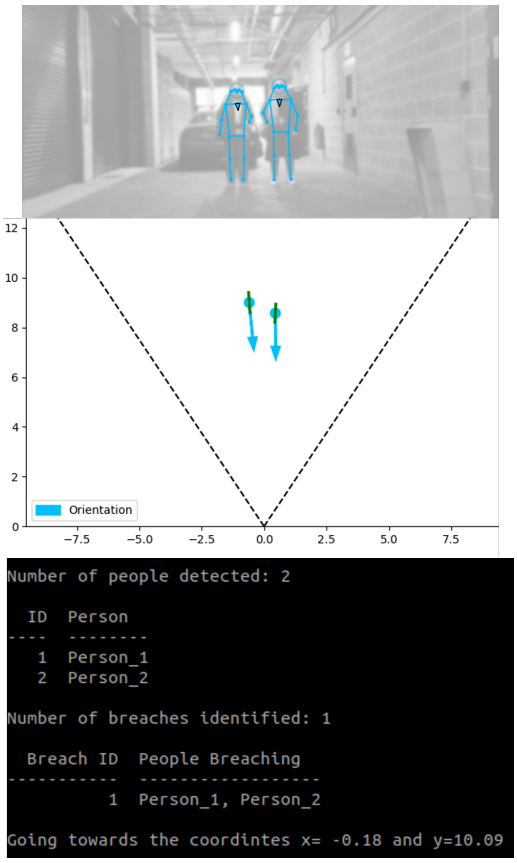
Reality:-
Similarly as in simulation, the classification precision is high in the real environment (≥ 90%) (Table 4.1). However, the accuracy and recall fall off. This decrease is first due to the negative effect of the real environment and its surrounding on the performance of our systems. Additionally, when experimenting, we noticed that a large amount of misclassification happens at the limits of the cameras' FOV, meaning when the ALE from Figure 4.2 is relatively high. Comparing the accuracy and recall we got to the ones from the original paper [59] (84.0% accuracy and 75% recall), it seems that both results are very close.
Overall, both algorithms maintained an accuracy equal to 82%. Qualitative results of both methods are shown in Figure 4.3a and 4.3b. Combined, these results show the promising ability of these approaches in recognizing breaches in social distancing.

Algorithm	Accuracy (%)	Precision (%)	Recall (%)
RGB-D detection Sim.	96	98	95
RGB detection Sim.	92	96	88
RGB-D detection Reality	82	90	72
RGB detection Reality	82	93	70

Table 4.1: Classifying social distancing breaches: Precision, Recall and Accuracy



(a) RGB-D detection simulation

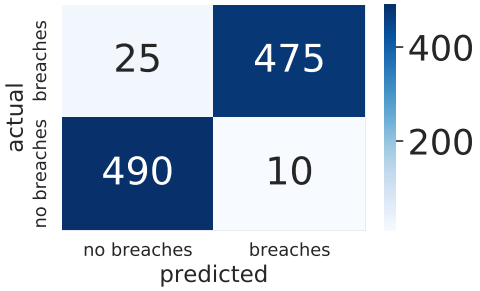


(b) RGB detection simulation

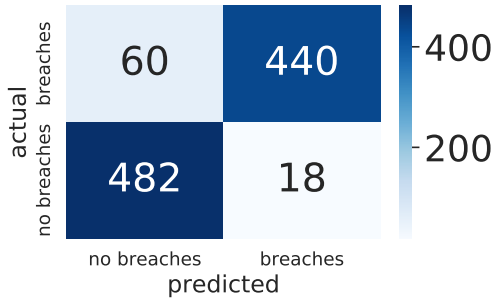
Figure 4.3: Example of our robot's vision modules (i.e., RGB and RGB-D) breaches detection accuracy in reality.

Embedded HW	Status	CPU1(%)	CPU2(%)	CPU3(%)	CPU4(%)	Memory (GB)	GPU(%)
Xavier NX	Idle	15	15	13	13	2.5/8	1
	RGB detection	44	42	36	34	7.2/8	59
	RGB-D detection	60	50	56	65	3.3/8	15
	RGB + RGB-D	76	63	70	75	7.4/8	63
Nano	Idle	18	15	15	14	1.6/4.1	3
	RGB detection	NA	NA	NA	NA	NA	NA
	RGB-D detection	99	98	99	99	2.7/4.1	83

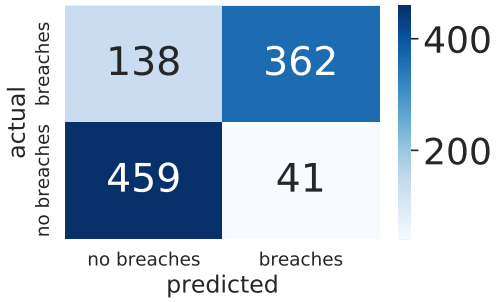
Table 4.2: Breach detection algorithms performance comparison



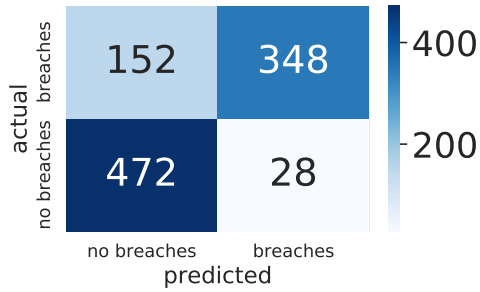
(a) RGB-D detection simulation



(b) RGB detection simulation



(c) RGB-D detection reality



(d) RGB detection reality

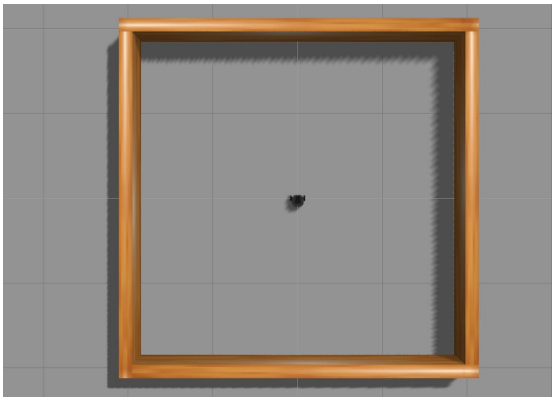
Figure 4.4: Breaches Detection Accuracy in reality: RGB and RGB-D based breach detection system.

4.1.4. Hardware Performance

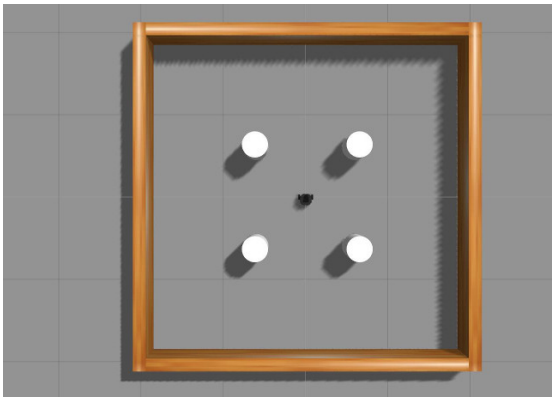
After testing the accuracy of the two breach detection systems, we deployed them on the identified edge devices to profile their execution on different processors. We tested the performance in terms of CPU, GPU, and memory usage.

Firstly, we measured the resources utilization levels at an idle condition, and then we executed the model for a few minutes to reach steady-state behavior and took 100 performance measurements that we averaged out. The results are presented in Table 4.2. Starting with the Jetson Nano, it was able to run the RGB-D system based on YOLOv3 [9]. As seen in Table 4.2, when running the RGB-D detection system, the CPU resource consumption is maximal ($\geq$ 98%), and the GPU utilization is very high. Additionally, the RGB breach detection did not run on the Jetson Nano due to its high resource requirements (i.e., it consists of three different deep learning models to detect breaches: Mask R-CNN [60] and OpenPifPaf [61] for pose detection and MonoLoco for 3D coordinates approximation).

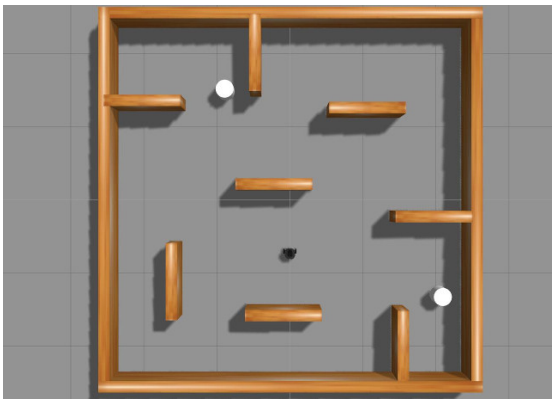
On the other hand, both systems ran individually on the more expensive Jetson Xavier NX, with appro-



(a) Scenario 1: No obstacles



(b) Scenario 2: Static obstacles



(c) Scenario 3: Static and dynamic obstacles

Figure 4.5: Training environments.

priate resource utilization. The RGB breach detection system noted the highest GPU utilization of 59%, while the RGB-D system had the highest CPU utilization equal to 65% . However, the memory consumption of the RGB system seemed to be very high and averaged 7.2 GB RAM out of the eight available. Finally, considering that the individual performances were favorable, we tested our compound vision system on the Xavier NX. Again, the overall performance seemed promising, with the highest average CPU utilization being equal to 76%, along with a 63% GPU utilization, and 7.4 GB of RAM usage.

Overall, the Jetson Nano has very limited performance and does not leave much room to run the overall system composed of three Deep Learning detection algorithms. However, the Xavier NX seems to provide more resources and could be leveraged to ensure a functional design.

4.2. Navigation System

An essential aspect of our autonomous robot revolves around the navigation system. Therefore, we measured the robustness of our navigation solutions by assessing three main metrics: success rate, failure rate (whenever the robot collides or cannot reach its goal within a specific time limit), and average speed.

4.2.1. Experiment setup

We trained the DRL models on a computer equipped with an NVIDIA GeForce GTX 1060, 16 GB of RAM, and an Intel Core i7-8750H processor. We used the three virtual environments shown in Figure 4.5, to train and test our DRL algorithms using the Jetbot robot model. The environments consist of a $4 \times 4\,m^2$ room-like environment with no obstacles, static obstacles, and dynamic obstacles. The DRL models are trained for 4000 episodes by navigating a Jetbot robot model in these environments. We tested the models capabilities in simulation and reality by sampling 35 random configurations for each environment type.

Algorithms	Success rate [/100]	Failure Rate (collisions rate / deadlocks rate)	Average Speed [m/s]
ROS navigation stack	97	3 (3 / 0)	0.077
DDPG	87	13 (8 / 5)	0.125
SAC	92	8 (6 / 2)	0.17
Hybrid navigation	93	7 (7 / 0)	0.046

Table 4.3: Comparison of the four different navigation approaches in simulation. Results are averaged over 100 episodes.

Algorithms	Success rate [/100]	Failure Rate (collisions rate / deadlocks rate)	Average Speed [m/s]
DDPG	74	26 (11 / 15)	0.099
SAC	80	20 (8 / 12)	0.13
Hybrid navigation	90	10 (10 / 0)	0.057

Table 4.4: Comparison of the three different navigation approaches in reality. Results are averaged over 100 episodes.

4.2.2. Training performance

Figure 4.6 shows the cumulative rewards obtained by DDPG and SAC during training in an environment with static and dynamic obstacles where each data point represents the average reward over 25 episodes. After ≈ 350 episodes, SAC surpasses DDPG in performance constantly. SAC reached a maximum average reward of approximately 3000, double that of DDPG. We hypothesize that as SAC is a stochastic policy, it is able to explore better the environment and therefore collect higher rewards.

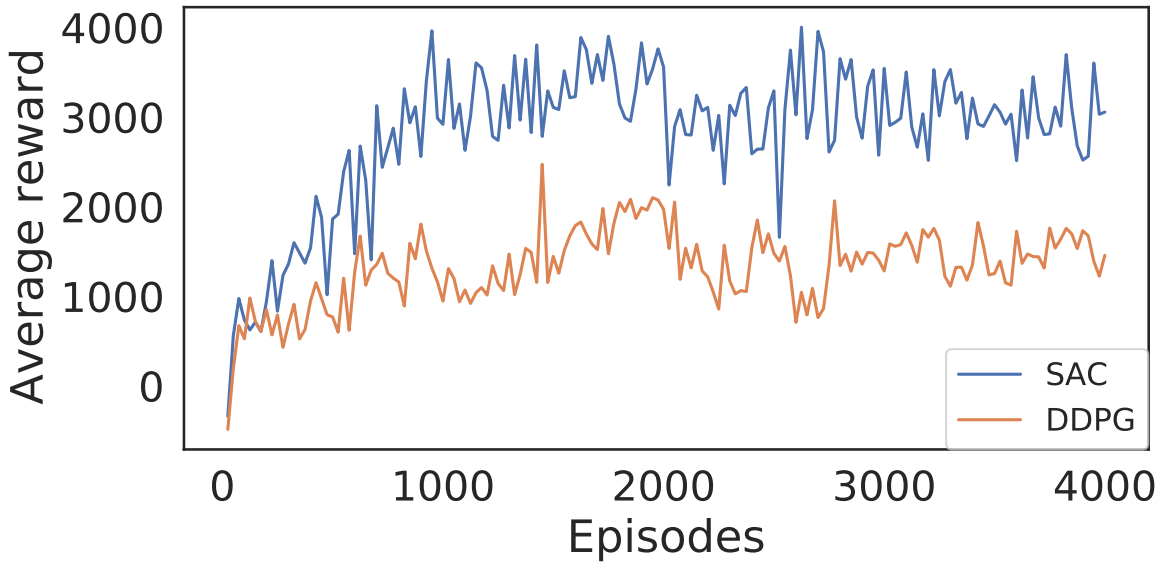


Figure 4.6: Training in environment with varying obstacles.

4.2.3. Simulation

From table 4.3, we observe that the four different methods assessed perform well in the different environments and rarely collide with static and dynamic obstacles with success rates higher than ≥ 87%. However, both mapless navigation approaches were stuck due to a position loss and could not recover, leading to a deadlock rate equal to 5% and 2% for DDPG and SAC, respectively. On the contrary, our hybrid model seems to fix this issue, with no noted deadlocks in simulation. It, therefore, appears to provide a solution for our

self-localization problem.

Another main difference between the analyzed methods is the average speed. Methods leveraging a map, i.e. the SLAM and hybrid approaches, are nearly two to three times as slow as the DDPG and SAC methods, with the hybrid approach being the slowest, averaging a speed of 0.046 m/s. This behavior is expected as the robot would need to keep track of its position on the given map.

4.2.4. Reality

Similarly as in simulation, we notice from table 4.4 that the mapless methods (DDPG and SAC) fail on multiple occasions due to the loss of odometry (15% and 12% deadlock rates for the DDPG and SAC algorithms, respectively). Considering that we took 100 measurements, these rates are relatively high. On the contrary, using the hybrid approach mitigates this failure, and our DRL agent reaches its target successfully 90% of the time.

The hybrid method is, however, twice as slow as both mapless solutions, with an average speed of $\approx 0.057 m/s$, compared to the average speed for DDPG ($\approx 0.099 m/s$), and that of SAC ($\approx 0.13 m/s$). Moreover, SAC is faster than DDPG in reaching the target: we suspect that the stochastic nature of SAC provides better exploration and allows the model to find an optimal path, whereas DDPG is deterministic.

In general, we observe that the hybrid approach can be considered a viable alternative to mapless DRL navigation whenever the odometry fails. However, it favors robustness over speed. Finally, the DRL algorithms, and our hybrid method seemed to effectively generalize to unseen real-world environments using the model trained on the environment with static and dynamic obstacles. They hence performed well, without the need to change any of the DRL parameters previously defined in the simulation. This further shows the potential of using DRL autonomous navigation methods for robot navigation in the real world.

5

Conclusion & Future Work

Designing and deploying a robot swarm that encourages social distancing is a challenging task that requires many considerations to ensure that the robots can effectively detect violations while being scalable. Single robot systems have already been proposed for detecting social distancing violations between pedestrians. However, the aforementioned systems are either not effective enough to detect breaches or very expensive and thus not scalable enough to form a swarm.

Hence, in this work, we developed an end-to-end cost-efficient robotic system using the JetBot AI robot from NVIDIA, which aims at encouraging social distancing. Our solution leverages computer vision to detect breaches and DRL for robot navigation.

For breach detection, we assessed the individual performance of the RGB-D and RGB detection systems. Based on the reality measurements, the RGB-D vision system accurately approximated the 3D coordinates of pedestrians for short distances, ranging from 2 to 5m, while the RGB system best estimated 3D people coordinates from ≈ 5 to 12m. In general, both methods seemed to detect breaches accurately (Table 4.1). This was further demonstrated in the qualitative results we got in Figure 4.3. Finally, using our compound solution deployed on the Jetson Xavier NX, and running both the RGB-D and RGB detection systems, we were able to accurately detect breaches up to 12m.

As for navigating our robot, we first utilized the mapless SAC actor-critic DRL algorithm, which successfully reached the targets more than 80% of the time. However, considering that the LiDAR odometry, responsible for self-localizing the robot in its environment, failed on multiple occasions, we developed a hybrid navigation stack. The latter is designed to overcome this limitation and performs best with respect to the failure rate at the cost of slower navigation. This proposed solution proved to be successful 90% of the time and is a good fit for our robot.

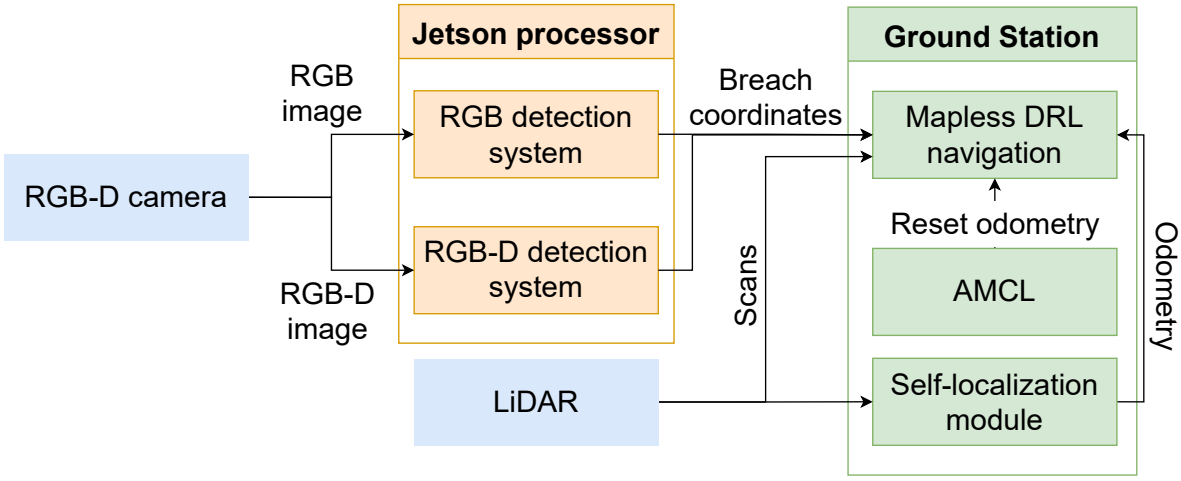


Figure 5.1: Overall architecture of our system.

Our final system runs the compound detection on the Jetson processor. A ground station runs the LiDAR odometry computations and the hybrid navigation system. The ground station and the Jetson processor are connected to each other via ROS communication. Figure 5.1 shows the overall architecture of the robot. Each component runs individually as a ROS node, and all the nodes communicate with each other by publishing and subscribing to topics. A more in depth view of the nodes is available in the appendix.

5.1. Recommendations & Suggestions for Future Work

Developing a robot swarm is a complex process, and many considerations should be taken into account as to optimize the cost and performance. In this thesis, we took a bottom-up approach and started by building a single robot that could constitute the first step towards a swarm. As such, in this section, we present some of the suggestions for future work to improve our system:

- *Deploying a swarm of robots:-* Developing a complete swarm of robots that leverages our compound vision system and hybrid navigation, thus covering more range for detecting breaches and encouraging social distancing. Additionally, we could target other applications such as alerting littering individuals, finding objects of interest, e.g., security of airports.

- *Vision-based navigation:-* Developing a pure vision based navigation system for our robot would allow us to forgo the LiDAR which is an expensive sensor. As previously mentioned, visual robot navigation using DRL is still a relatively new approach [31], with several limitations: it uses a discrete set of actions, and does not account for obstacles while navigating.

- *Account for privacy concerns:-* Preserving the privacy of the detected people is necessary when identifying breaches. While the RGB detection system [59] approximates 3D coordinates from a 2D pose (a low-dimensional representation that respects privacy concerns), the RGB-D detection using YOLO requires an RGB-D image as input which may contain sensitive data. Solutions have been proposed in the literature for detecting people while preserving their privacy, such as blurring their faces [78], or altering the pixels of the images (producing alterations in saturation, contrast, brightness, etc.) [79].

- *Improve the detection accuracy of the RGB system:-* Based on our experiments, the RGB detection system is best suited for detecting breaches between 5 and 12m. However, considering the cost of an RGB camera (10x cheaper than the RGB-D one), it would be beneficial to leverage it for both short range and long range detection if we are able to improve on its accuracy for short range distances, thus forgoing the RGB-D camera.

Bibliography

[1] M. Rezaei and M. Azarmi, "Deepsocial: Social distancing monitoring and infection risk assessment in covid-19 pandemic," *Applied Sciences*, vol. 10, no. 21, p. 7514, Oct. 2020, ISSN: 2076-3417. DOI: 10.3390/app10217514. [Online]. Available: http://dx.doi.org/10.3390/app10217514.

[2] I. Ahmed, M. Ahmad, J. Rodrigues, G. Jeon, and S. Din, "A deep learning-based social distance monitoring framework for covid-19," *Sustainable Cities and Society*, vol. 65, p. 102 571, Nov. 2020. DOI: 10.1016/j.scs.2020.102571.

[3] N. S. Punn, S. K. Sonbhadra, S. Agarwal, and G. Rai, *Monitoring covid-19 social distancing with person detection and tracking via fine-tuned yolo v3 and deepsort techniques*, 2021. arXiv: 2005.01385 [cs.CV].

[4] C. T. Nguyen *et al.*, "A comprehensive survey of enabling and emerging technologies for social distancing—part i: Fundamentals and enabling technologies," *IEEE Access*, vol. 8, pp. 153 479–153 507, 2020, ISSN: 2169-3536. DOI: 10.1109/access.2020.3018140. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2020.3018140.

[5] S. Agarwal *et al.*, *Unleashing the power of disruptive and emerging technologies amid covid-19: A detailed review*, 2021. arXiv: 2005.11507 [cs.CY].

[6] H.-W. Huang *et al.*, *Agile mobile robotic platform for contactless vital signs monitoring*, 2020.

[7] C. Sun and Z. Zhai, "The efficacy of social distance and ventilation effectiveness in preventing covid-19 transmission," *Sustainable Cities and Society*, vol. 62, p. 102 390, 2020, ISSN: 2210-6707. DOI: https://doi.org/10.1016/j.scs.2020.102390.

[8] Z. Voko and J. Pitter, "The effect of social distance measures on covid-19 epidemics in europe: An interrupted time series analysis," *GeroScience*, vol. 42, Jun. 2020. DOI: 10.1007/s11357-020-00205-0.

[9] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, 2018. arXiv: 1804.02767 [cs.CV].

[10] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," *2016 IEEE International Conference on Image Processing (ICIP)*, Sep. 2016. DOI: 10.1109/icip.2016.7533003. [Online]. Available: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[11] N. Wojke, A. Bewley, and D. Paulus, *Simple online and realtime tracking with a deep association metric*, 2017. arXiv: 1703.07402 [cs.CV].

[12] A. Channa, N. Popescu, J. Skibinska, and R. Burget, "The rise of wearable devices during the covid-19 pandemic: A systematic review," *Sensors*, vol. 21, no. 17, p. 5787, 2021.

[13] S. Sun *et al.*, "Using smartphones and wearable devices to monitor behavioral changes during covid-19," *Journal of Medical Internet Research*, vol. 22, no. 9, e19992, Sep. 2020, ISSN: 1438-8871. DOI: 10.2196/19992. [Online]. Available: http://dx.doi.org/10.2196/19992.

[14] Y. Shen *et al.*, "Robots under covid-19 pandemic: A comprehensive survey," *Ieee Access*, 2020.

[15] C. McCaffrey, A. Taylor, S. Roy, S. B. Banisetty, R. Mead, and T. Williams, "Can robots be used to encourage social distancing?," ser. HRI '21 Companion, Boulder, CO, USA: Association for Computing Machinery, 2021, pp. 475–478, ISBN: 9781450382908. DOI: 10.1145/3434074.3447217. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3434074.3447217.

[16] T. Fan *et al.*, *Autonomous social distancing in urban environments using a quadruped robot*, 2020. arXiv: 2008.08889 [cs.RO].

[17] A. J. Sathyamoorthy, U. Patel, M. Paul, Y. Savle, and D. Manocha, "Covid surveillance robot: Monitoring social distancing constraints in indoor scenarios," *PLOS ONE*, vol. 16, pp. 1–20, Dec. 2021. DOI: 10.1371/journal.pone.0259713. [Online]. Available: https://doi.org/10.1371/journal.pone.0259713.

[18] K. Rana, B. Talbot, V. Dasagi, M. Milford, and N. Sünderhauf, "Residual reactive navigation: Combining classical and learned navigation strategies for deployment in unknown environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 11 493–11 499.

[19] B. Cèsar-Tondreau, G. Warnell, E. Stump, K. Kochersberger, and N. R. Waytowich, "Improving autonomous robotic navigation using imitation learning," *Frontiers in Robotics and AI*, vol. 8, 2021.

[20] K. Zheng, "Ros navigation tuning guide," in *Robot Operating System (ROS)*, Springer, 2021, pp. 197–226.

[21] A. Y. Majid, S. Saaybi, T. van Rietbergen, V. Francois-Lavet, R. V. Prasad, and C. Verhoeven, "Deep reinforcement learning versus evolution strategies: A comparative survey," *arXiv preprint arXiv:2110.01411*, 2021.

[22] L. Tai, G. Paolo, and M. Liu, *Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation*, 2017. arXiv: 1703.00420 [cs.RO].

[23] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, *Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments*, 2020. arXiv: 2005.13857 [cs.RO].

[24] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 5129–5136.

[25] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, *Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning*, 2018. arXiv: 1709.10082 [cs.RO].

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[27] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine, "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6284–6291. DOI: 10.1109/ICRA.2018.8461039.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: 1801.01290 [cs.LG].

[29] J. Costa de Jesus, V. Kich, A. Kolling, R. Grando, M. Cuadros, and D. F. Gamarra, "Soft actor-critic for navigation of mobile robots," *Journal of Intelligent Robotic Systems*, vol. 102, Jun. 2021. DOI: 10.1007/s10846-021-01367-5.

[30] J. Kulhánek, E. Derner, T. De Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," pp. 1–8, 2019.

[31] J. Kulhanek, E. Derner, and R. Babuska, *Visual navigation in real-world indoor environments using end-to-end deep reinforcement learning*, 2020. arXiv: 2010.10903 [cs.RO].

[32] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[33] T. Haarnoja, S. Levine, M. Dalal, A. Zhou, K. Hartikainen, and V. Pong, *Soft actor critic-deep reinforcement learning with real-world robots*, Dec. 2018. [Online]. Available: https://bair.berkeley.edu/blog/2018/12/14/sac/.

[34] H. Choi *et al.*, "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward," *Proceedings of the National Academy of Sciences*, vol. 118, no. 1, 2021.

[35] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, "Comparing popular simulation environments in the scope of robotics and reinforcement learning," *arXiv preprint arXiv:2103.04616*, 2021.

[36] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, "Feature and performance comparison of the v-rep, gazebo and argos robot simulators," in *Annual Conference Towards Autonomous Robotic Systems*, Springer, 2018, pp. 357–368.

[37] S. Ivaldi, V. Padois, and F. Nori, "Tools for dynamics simulation of robots: A survey based on user feedback," *arXiv preprint arXiv:1402.7050*, 2014.

[38] L. Nogueira, "Comparative analysis between gazebo and v-rep robotic simulators," *Seminario Interno de Cognicao Artificial-SICA*, vol. 2014, no. 5, 2014.

[39] A. Ayala, F. Cruz, D. Campos, R. Rubio, B. Fernandes, and R. Dazeley, *A comparison of humanoid robot simulators: A quantitative approach*, 2020. arXiv: 2008.04627 [cs.RO].

[40] *Frequently asked questions.* [Online]. Available: https://coral.ai/docs/edgetpu/faq/.

[41] *Intel movidius vision processing units (vpus).*

[42] *Nvidia embedded systems for next-gen autonomous machines.* [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[43] Z. Lv, J. L. Mauri, and H. Song, "Editorial rgb-d sensors and 3d reconstruction," *IEEE Sensors Journal*, vol. 20, no. 20, pp. 11 751–11 752, 2020. DOI: 10.1109/JSEN.2020.3015417.

[44] *Zed 2 ai stereo camera.* [Online]. Available: https://www.stereolabs.com/zed-2/.

[45] *Asus xtion pro.* [Online]. Available: http://xtionprolive.com/asus-xtion-pro-live.

[46] *Microsoft kinect.* [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect/.

[47] *Intel realsense camera.* [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html.

[48] A. Simonelli, S. R. R. Bulò, L. Porzi, M. López-Antequera, and P. Kontschieder, *Disentangling monocular 3d object detection*, 2019. arXiv: 1905.12365 [cs.CV].

[49] L. Bertoni, S. Kreiss, and A. Alahi, *Monoloco: Monocular 3d pedestrian localization and uncertainty estimation*, 2019. arXiv: 1906.06059 [cs.CV].

[50] J. Zhong, M. Li, X. Liao, and J. Qin, "A real-time infrared stereo matching algorithm for rgb-d cameras' indoor 3d perception," *ISPRS International Journal of Geo-Information*, vol. 9, no. 8, p. 472, 2020.

[51] S. Liu, D. Gao, W. Peng, X. Guo, J. Xu, and D.-X. Liu, "A depth-based weighted point cloud registration for indoor scene," *Sensors*, vol. 18, p. 3608, Oct. 2018. DOI: 10.3390/s18113608.

[52] A. N. Catapang and M. Ramos, "Obstacle detection using a 2d lidar system for an autonomous vehicle," in *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, IEEE, 2016, pp. 441–445.

[53] S. Kawakura and R. Shibasaki, "Deep learning-based self-driving car: Jetbot with nvidia ai board to deliver items at agricultural workplace with object-finding and avoidance functions," *European Journal of Agriculture and Food Sciences*, vol. 2, no. 3, 2020.

[54] A. Raman, V. Krovi, and M. Schmid, "Empowering graduate engineering students with proficiency in autonomy," Aug. 2018, V05AT07A080.

[55] R. M. Garcıa, D. H. de la Iglesia, J. F. de Paz, V. R. Leithardt, and G. Villarrubia, "Urban search and rescue with anti-pheromone robot swarm architecture," in *2021 Telecoms Conference (ConfTELE)*, IEEE, 2021, pp. 1–6.

[56] M. Bjelonic, *Yolo ros: Real-time object detection for ros*, 2018.

[57] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[58] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[59] L. Bertoni, S. Kreiss, and A. Alahi, *Perceiving humans: From monocular 3d localization to social distancing*, 2021. arXiv: 2009.00984 [cs.CV].

[60] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask r-cnn*, 2018. arXiv: 1703.06870 [cs.CV].

[61] S. Kreiss, L. Bertoni, and A. Alahi, *Pifpaf: Composite fields for human pose estimation*, 2019. arXiv: 1903.06593 [cs.CV].

[62] M. Mozaffari, A. Broumandan, K. O'Keefe, and G. Lachapelle, "Weak gps signal acquisition using antenna diversity," in *2014 Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS)*, 2014, pp. 11–18. DOI: 10.1109/UPINLBS.2014.7033705.

[63] S. A. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, "A survey on odometry for autonomous navigation systems," *IEEE Access*, vol. 7, pp. 97 466–97 486, 2019.

[64] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. Ismail, "Review of visual odometry: Types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, 2016.

[65] R. González, F. Rodríguez, J. L. Guzmán, C. Pradalier, and R. Y. Siegwart, "Control of off-road mobile robots using visual odometry and slip compensation," *Advanced Robotics*, vol. 27, pp. 893–906, 2013.

[66] *Rtabmap_ros ros package*. [Online]. Available: http://wiki.ros.org/rtabmap_ros.

[67] *Ros navigation stack*. [Online]. Available: http://wiki.ros.org/navigation.

[68] *Gmapping ros package*. [Online]. Available: http://wiki.ros.org/gmapping.

[69] *Hector_slam ros package*. [Online]. Available: http://wiki.ros.org/hector_slam.

[70] *Amcl ros package*. [Online]. Available: http://wiki.ros.org/amcl.

[71] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," Jan. 1999, pp. 343–349.

[72] P. Marin-Plaza, A. Hussein, D. Martin, and A. d. l. Escalera, "Global and local path planning study in a ros-based research platform for autonomous vehicles," *Journal of Advanced Transportation*, vol. 2018, 2018.

[73] J. Xiang, Q. Li, X. Dong, and Z. Ren, "Continuous control with deep reinforcement learning for mobile robot navigation," in *2019 Chinese Automation Congress (CAC)*, 2019, pp. 1501–1506. DOI: 10.1109/CAC48633.2019.8996652.

[74] T. P. Lillicrap *et al.*, *Continuous control with deep reinforcement learning*, 2019. arXiv: 1509.02971 [cs.LG].

[75] T. Haarnoja *et al.*, *Soft actor-critic algorithms and applications*, 2019. arXiv: 1812.05905 [cs.LG].

[76] T.-Y. Lin *et al.*, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV].

[77] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[78] P. He *et al.*, *Privacy-preserving object detection*, 2021. arXiv: 2103.06587 [cs.CV].

[79] I. Rodriguez-Conde, C. Campos, and F. Fdez-Riverola, "On-device object detection for more efficient and privacy-compliant visual perception in context-aware systems," *Applied Sciences*, vol. 11, no. 19, p. 9173, 2021.

6

Appendix

6.1. Appendix A: RGB-D ROS nodes

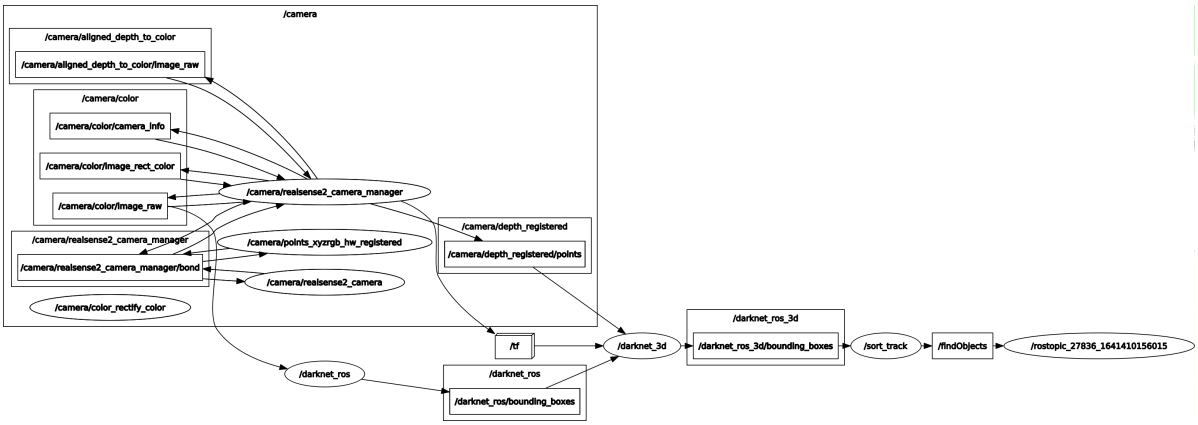


Figure 6.1: RGB-D ROS nodes.

6.2. Appendix B: RGB ROS nodes

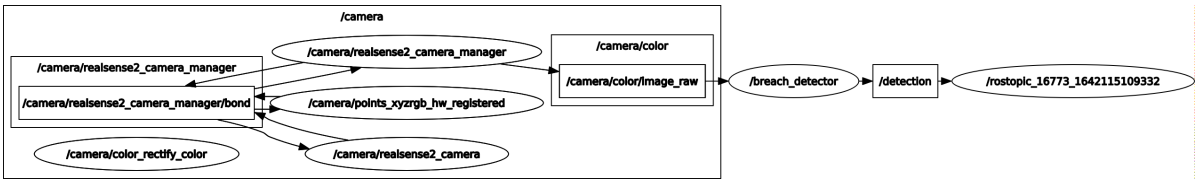


Figure 6.2: RGB ROS nodes.

6.3. Appendix C: Hybrid navigation ROS Nodes

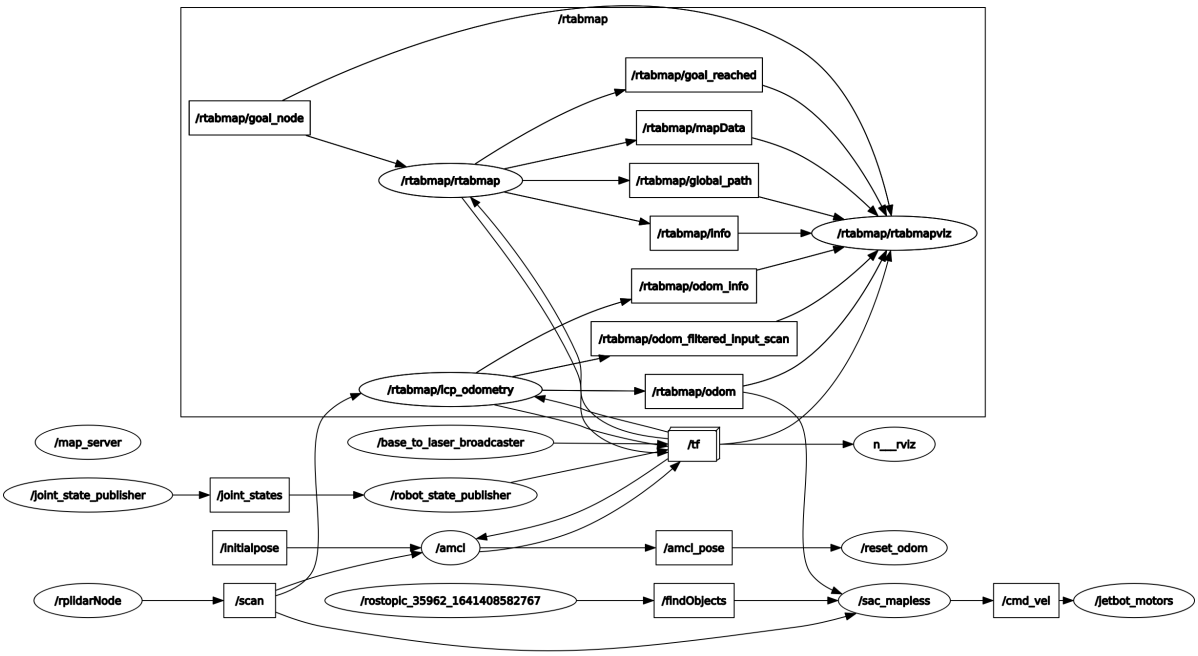


Figure 6.3: Hybrid navigation ROS nodes.