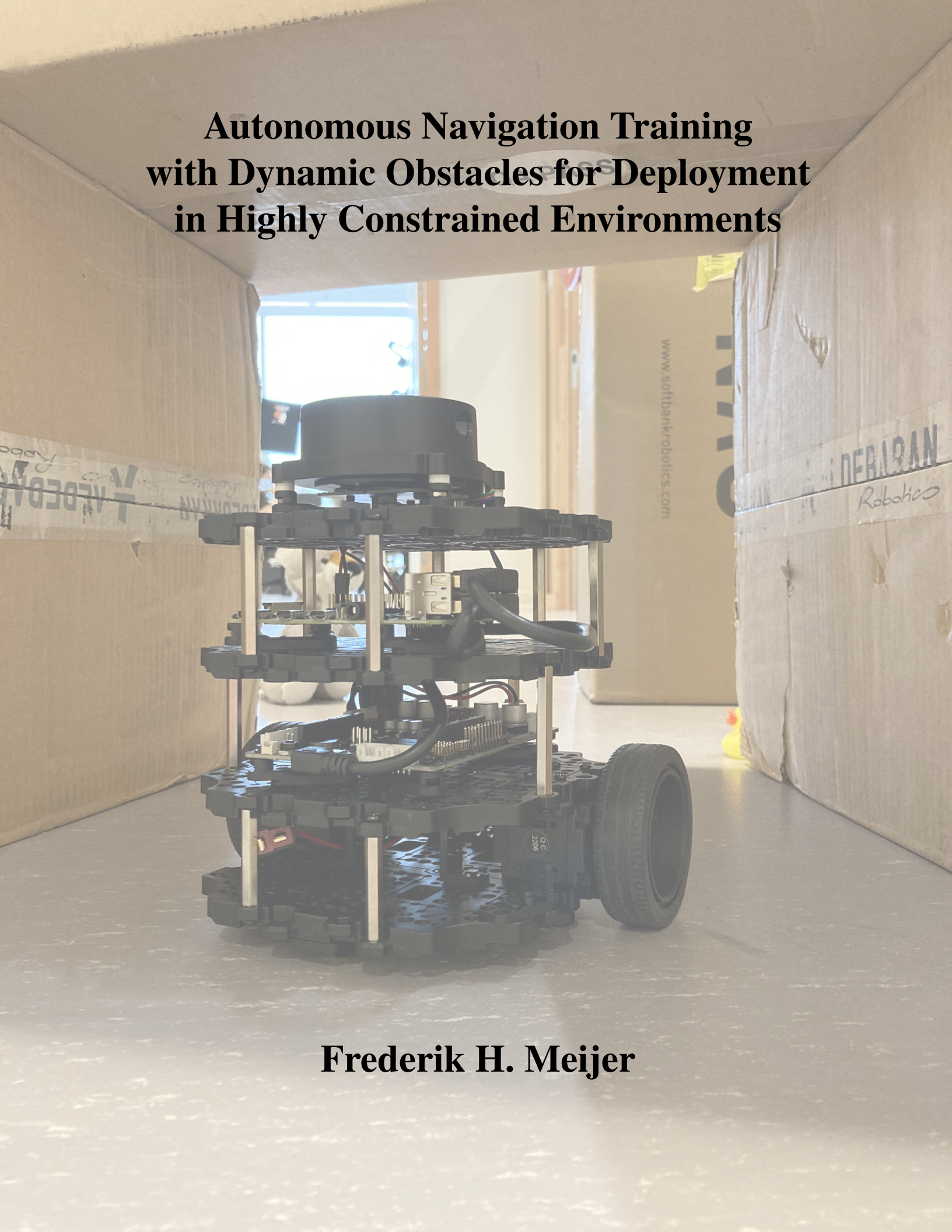
Autonomous Navigation Training with Dynamic Obstacles for Deployment in Highly Constrained Environments



Frederik H. Meijer

Layout: typeset by the author using LaTeX.
Cover illustration: photo taken by the author at UvA Science Park
Thanks to my supervisor Arnoud Visser for guiding me during my research and providing feedback for my thesis, and to Joey van der Kaaij for reserving and configuring university workstations for me to use.

Autonomous Navigation Training with Dynamic Obstacles for Deployment in Highly Constrained Environments

Frederik H. Meijer
13151673

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 900
1098 XH Amsterdam

Supervisor
Dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

June 30, 2023

Abstract

The BARN benchmark is meant as a general testbed to compare the performance of different navigation policies and consists of 300 procedurally generated static environments. In many real-world applications, however, navigation policies have to deal not only static obstacles, but also dynamic ones such as other mobile robots and human pedestrians. In order to evaluate the applicability of the BARN benchmark on such policies, this paper compares two different navigation policies that were trained in a dynamic environment with 34 moving pedestrains. The DRL-VO model is based on reinforcement learning with a reward function containing knowledge distillation term, which allows it to learn expert behaviour from a velocity obstacle based model. The DRL model does not have this knowledge distillation term. Both models showed noticably different behaviour in the BARN benchmark, and DRL managed to attain a higher navigation benchmark score.

Table of Content

Chapter 1

Introduction

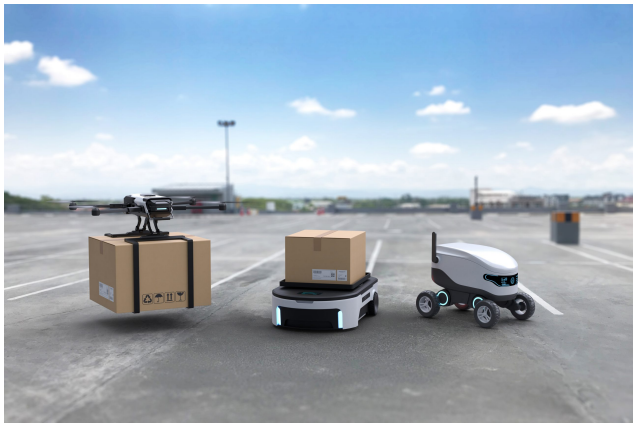


Figure 1.1: Autonomous delivery robots, one possible future application of autonomous robot navigation.[1]

Autonomous robot navigation is an important subdomain within the field of robotics that has been studied for decades. It concerns the problem of moving from one point to another, without an operator's instructions on how to do so. This is a broad problem which has many real-world applications. Autonomous cars, for instance, rely on their navigation policies to avoid collisions with other vehicles and safely transport passengers to their destination. In the home environment, robot vacuum cleaners need to navigate tight gaps between furniture in order to remove dust in hard-to-reach spaces. In near the future, delivery robots could be deployed in cities to deliver groceries to your door without the need for a human driver (Figure 1.1). These applications in diverse environments pose several challenges for autonomous robots, which warrants the development of robust navigation policies.

Over the years, many systems have been designed to implement autonomous robot navigation in a collision-free manner. These include classical navigation planners, such as the dynamic window approach (Appendix 7.1), making use of techniques such as SLAM (Appendix 7.2). Another area of research is in learning-based navigation systems, making use of techniques such as reinforcement learning (RL) to learn a navigation policy from trial-and-error data. Within learning-based approaches, there has also been innovation in end-to-end navigation policies: here, most intermediate data representation and navigation algorithms are replaced with a neural network architecture that learns to produce motion commands from raw sensor data (Pfeiffer, Schaeuble, Nieto, Siegwart, & Cadena, 2017). With all of these various training methods come various measuring methods as well. Until recently, there has not been a commonly agreed upon method to mea-

[1] Image source: Infinium Logistics

sure the performance of different systems. A standardised testing method is required to compare the results between different methods of autonomous robot navigation. The Benchmark for Autonomous Robot Navigation (BARN) attempts to fulfil this role by providing a general testbed for ground navigation in a variety of randomly generated worlds (Perille et al., 2020).

1.1 BARN

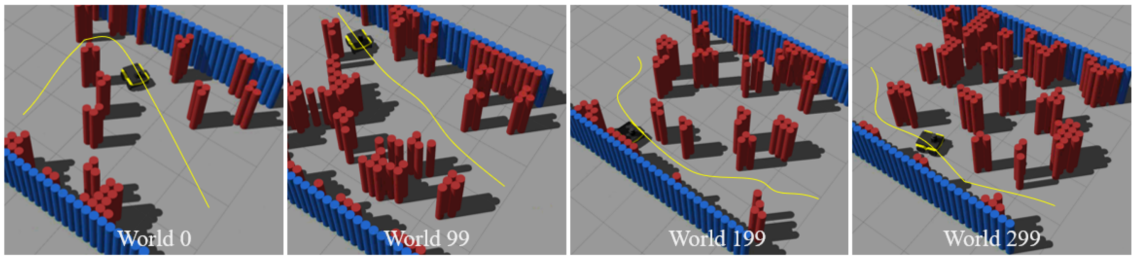


Figure 1.2: Examples of the generated worlds in the BARN dataset. The worlds scale in difficulty as their number increases (Perille et al., 2020).

The BARN dataset consists of 300 procedurally generated environments, or worlds (Figure 1.2), in which a simulated robot can navigate from a fixed start position to a goal position. The worlds are made up of a grid of 30x30 tiles that can either be open space or a wall. They are numbered, where lower number worlds are more open and thus easier to navigate, and higher number worlds are more cramped and difficult. 5 different difficulty metrics have been developed to indicate the difficulty of a path through the environment. These are: distance to closest obstacle, average visibility, dispersion, characteristic dimension, and tortuosity. Each of these measures apply to a single cell, and they are calculated along the entire path from start to finish to establish the world's difficulty. This path is found by an expert path planning algorithm (A*) which has a complete overview of the entire world's layout.

1.2 The BARN challenge

Since 2022, an annual competition called the BARN challenge has been organised, in which contestants try to design a navigation system that can achieve an as high as possible score in the BARN benchmark. It consists of two rounds: first, teams can submit their system to the organisers to be tested in a simulated environment. This simulated environment is meant to re-create the challenges of the real world, and as such the robots do not have access to the full map of the BARN worlds. The Gazebo (Section 2.2) simulator is used to generate realistic sensor input and move a simulated robot through the environment. The three teams that attain the highest score on this trial are then invited to compete in the physical round at the ICRA convention.

Performance on the simulated trials is measured using the world's optimal traversal time OT and the robot's actual traversal time AT. The task is to navigate a Clearpath Jackal robot from a start to a goal location as quickly as possible without any collisions. Both in the simulated environment and in the real world, this robot is equipped with a 2D LiDAR (Section 2.6), a motor controller with a maximum velocity of 2 m/s and an on-board computer. It is up to the participants to design a system that consumes the LiDAR input and outputs commands to the motor controller. This must be submitted as a collection of launchable ROS (Section 2.1) nodes. Equation 1.1 below shows the calculation of the navigation performace s_i is the score on world i:

$$s_i = \mathbb{1}^{\text{success}} \times \frac{OT}{\text{clip}(AT_i, 4OT_i, 8OT_i)} \tag{1.1}$$

where $\text{clip}(x, a, b)$ is defined as

$$\text{clip}(x, a, b) = \begin{cases} a & \text{if } x < a \\ b & \text{if } x > b \\ x & \text{otherwise} \end{cases} \tag{1.2}$$

AT is clipped between 4 and 8 times OT in order to not disproportionally skew the score by navigating either extremely quickly or extremely slowly. Because of this, the maximum attainable score is 0.25.

1.3 Competition leaders

Figure 1.3: The contestants at the physical competition round at the ICRA conference on May 2023: KUL+FM, INVENTEC, and University of Almeria (1st, 2nd, and 3rp place).[2]

Ranking	Team	Score	Comment
1	KUL+FM	0.2490	KU Leuven & Flanders Make
2	INVENTEC	0.2445	Inventec Corporation
3	University of Almeria	0.2439	University of Almeria & Teladoc
4	UT AMRL	0.2424	The University of Texas at Austin
5	Temple TRAIL[22]	0.2415	Temple University
6	LfLH[b]	0.2334	Details
7	UT AMRL[22]	0.2310	The University of Texas at Austin
8	Temple TRAIL	0.2290	Temple University
9	UVA AMR	0.2237	University of Virginia
10	RIL	0.2203	Indian Institute of Science, Bengaluru

Figure 1.4: The top 10 teams in the simulation round of the BARN challenge as of June 2023.[2]

Figure 1.4 shows the current competition leaders in simulation as of June 2023. In May, the score of the Champion of the 2022 competition, Temple Univerity's DRL-VO (Xie & Dames, 2023) model, was still the highest ranking (0.2415), so this algorithm was chosen as basis for this research.

The DRL-VO model uses deep reinforcement learning (DRL) to solve a POMDP which defines the navigation problem. The reward function uses a reward function with a knowledge distillation term based on velocity obstacles (VO) (Section 2.8) that allows the model to learn expert behaviours. A custom neural network is trained using the PPO (Section 2.4) algorithm with gradient descent using the Adam optimiser (Section 2.5). The policy is designed to navigate in open spaces crowded with pedestrians. They use the YOLOv3 detector (Section 2.10) to find pedestrians in an RGB-D image produced by a ZED stereo camera (Section 2.11) and track the pedestrian's motion using a multiple hypothesis tracker (MHT) (Section 2.9). The model implements an end-to-end navigation policy that tracks a sub-goal point calculated by the pure pursuit algorithm (Section 2.7).

Xie & Dames also tested a version of this model without the VO-based reward function, referred to as simply DRL. However, DRL was never applied to the BARN benchmark. Because VO's are used to handle

[2]Image source: cs.gmu.edu

navigation in environments that contain dynamic obstacles, and the BARN worlds are all static, an interesting question is in what way training with a VO-based reward function influences the navigation performance on BARN's challenging environments.

1.4 Research question

How does distilled knowledge on avoiding moving obstacles affect navigation performance in challenging static environments?

Chapter 2

Theory

This chapter details several components and techniques used in the DRL-VO model discussed in the next chapter.

2.1 ROS

The Robot Operating System (ROS) is an open-source framework for robotics software (Quigley et al., 2009). It is not an operating system in the sense of process management, but it provides a layer of communication between different models of the robot software. It sits above the host operating system of a single machine or the os's of a heterogenous compute cluster (Figure 2.1). The motivation behind ROS is that as the scale and complexity of robotics has grown, writing software for robots has become more and more difficult. Different types of robots can have widely different hardware, which requires large amounts of different driver-level code. On top of this, the software ranging from perception to abstract reasoning algorithms amounts to a daunting stack of code which can be difficult to reuse. ROS solves this problem by dividing the robotics software stack into intercommunicating modules, which makes it easy to adapt a system by simply changing out modules.

A system built using ROS consists of a number of different processes called 'nodes' (Figure 2.1). These nodes can run on multiple different machines, and are connected at runtime in a peer-to-peer fashion. The benefit of using a peer-to-peer topology over a monolith-based topology is that there is no need for a central server. Placing such a server either on board of the robot or offboard would cause considerable overhead, because many message routes are fully contained in the subnets either onboard or offboard. In order for processes to find each other at runtime, a lookup mechanism is needed. This is provided by the name service, or master node. Nodes communicate with each other through messages. A publisher node can publish these to a topic, where it is received by all subscribers to that topic. A topic can have multiple publishers and subscribers, and a single node can publish and subscribe to multiple topics. For synchronous transactions, there are services: these are analogous to web services with well-defined request and response types, and only one node can advertise a service of any particular name.

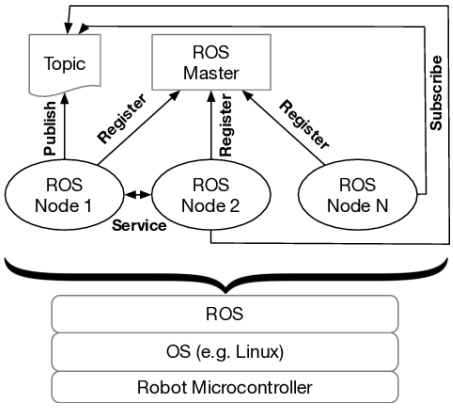


Figure 2.1: ROS hierarchy and inter-node communication.[1]

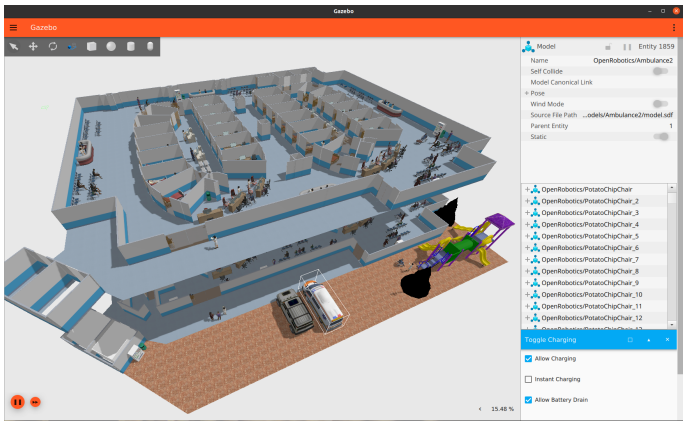


Figure 2.2: Example of a simulated environment in Gazebo.[2]

2.2 Gazebo

Gazebo is ROS's default 3D simulator (Figure 2.2). It is designed especially for robotics development (Koenig & Howard, 2004). Its original goal was to be used in realistic outdoor simulations, but it is suited for indoors environments as well. Its development has been motivated by the advent of all-terrain robotics platforms at the Robotics Research Lab at the University of Southern Carolina. These robots need to account for changes in ground elevation, so many other simulators, that are often restricted to 2D worlds, are not sufficient. Gazebo is designed to accurately reproduce the dynamic environments that such robots navigate and provide realistic sensor feedback. The robot's hardware is also accurately simulated: they are dynamic structures composed of rigid bodies which can have forces applied to them to generate locomotion. A client program sees the same interface for a real and simulated robot, which allows Gazebo to be seamlessly integrated in in the robot development process.

2.3 POMDP

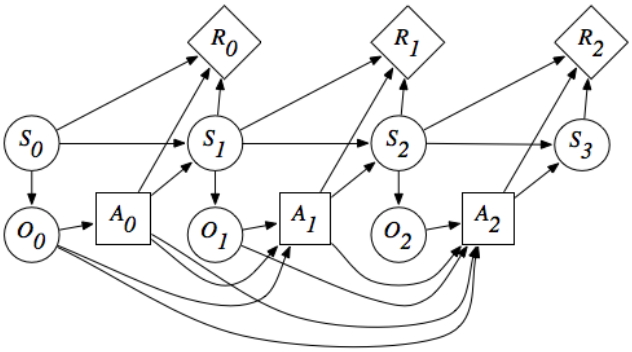


Figure 2.3: Graph illustration of a POMDP.[3]

[1] Image source: (Mazzeo & Staffa, 2020)
[2] Image source: gazebosim.org
[3] Image source: artint.info

The Markov Decision Process (MDP) is a framework for decision making at discrete points in time (Oliehoek & Visser, 2010). It can be used to formalise decision-theoretic discrete-time planning problems for a single agent in a stochastically changing environment, on the condition that the agent can observe the environment. In the case that the environment is not fully observable, a Partially Observable Markov Decision Process (POMDP) is used (Pineau, Gordon, & Thrun, 2006). It handles uncertainty in both the effect of the agent's actions and the state observability. Plans are expressed over information states, rather than world states, since the world state is not fully known. Information states are easily calculated from noisy sensor observations, and represented as a probability distribution over world states. POMDPs typically form plans by optimising a value function, which allows one to numerically trade off between different methods. The POMDP is defined as a tuple of six distinct quanties (Equation 2.1)

$$\{S, A, Z, T, O, R\} \tag{2.1}$$

where:

- S is the state space $S = \{s0, s1, \dots\}$
- A is the action space $A = \{a0, a1, \dots\}$
- Z is the observation space $Z = \{o0, o1, \dots\}$
- T is a state transition model
- O is an observation probability distribution
- R is the reward function

Figure 2.3 shows the states s_i, observations o_i, actions a_i, rewards r_i and the conditional relationships between them. The number of plans increases exponentially with the planning horizon. Because of this, many POMDP domains with only a few states, actions and observations become computationally intractable. The modern way to solve large POMDPs is to use reinforcement learning (RL) techniques, such as the PPO algorithm.

2.4 PPO

The Proximal Policy Optimisation (PPO) algorithm is a policy gradient method for reinforcement learning which alternates between sampling data and optimising an objective (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). Data is first sampled by interacting with the environment, after which the algorithm performs several epochs optimising on this data. The function to optimise (the "surrogate" objective) is an estimator of the policy gradient. The objective function is maximised subject to a constraint on the size of the policy update, because performing multiple steps of optimisation using the same trajectory often leads to destructively large policy updates. In empirical testing, it was shown that PPO performs especially well on robotic locomotion problems, outperforming previous methods on almost all control environments.

2.5 Adam Optimiser

A popular optimiser to use in the PPO algorithm is Adam, which gets its name from adaptive moment estimation. It is an algorithm for first-order gradient descent or ascent of stochastic (noisy) objective functions (Kingma & Ba, 2014). The stochasticity of the objective function could arise from inherent function noise, but in PPO it is a result of randomly sampling mini-batches of datapoints. In high-dimensional stochastic

optimisation problems, higher-order optimisation methods are ill-suited, which is why Adam uses first-order gradients. In order to deal with the stochastic nature of the objective, Adam updates exponential moving averages of the gradient and the squared gradient. These moving averages are initialised as zero vectors, which introduces a bias towards 0, especially in the initial timesteps. To deal with this bias, the moving averages are divided by bias correction terms.

2.6 LiDAR

There are many ways in which an observation can be made for the POMDP. Light detection and ranging (LiDAR) has gained much popularity in recent years for applications ranging from robot navigation to remote sensing (Raj, Hanim Hashim, Baseri Huddin, Ibrahim, & Hussain, 2020). This is mainly due its many attractive attributes, such as being light-weight, compact, accurate and inexpensive. LiDAR operates on the same principle as sonar, but uses laser beams rather than sound. A laser diode emits a short pulse, which, when reflected by an object in the environment, can be picked up again by the LiDAR's photo diode. Using the time of flight (TOF), the distance to the object can be computed. In the DRL-VO (Xie & Dames, 2023) training hardware, a Hokuyo UTM-30LX LiDAR (Figure 2.4) is used. It has a measurement range of [0.1, 30] m, a FOV of 270°, and an angular resolution is 0.25°. This is to say that the laser diode emits laser beams at an interval of 0.25°, which can bounce off of objects up to 30 meters away.

Figure 2.4: Hokuyo UTM-30LX LiDAR.[4]

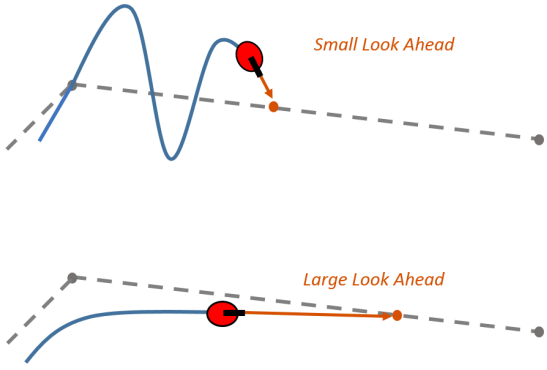


Figure 2.5: Pure pursuit with a small and large look-ahead distance converging to a path.[5]

2.7 Pure Pursuit

Another part of the POMDP's observation space is the target direction. Pure pursuit is a tracking algorithm that calculates a trajectory that will move an agent from its current location to a goal point (Coulter, 1992). It does so by finding a sub-goal point along the full path that is a certain distance ahead of the agent. This is analogous to the way humans drive: we tend to think about moving towards a point on the road ahead of us, such as stopping at the traffic light or turning the corner, rather than our final goal. The distance from the agent to its sub-goal point is called the lookahead distance (Figure 2.5). As the robot moves towards its goal, the lookahead distance stays the same, meaning that the sub-goal point constantly changes and moves towards the final goal. This lookahead distance can be changed to determine the algorithm's behaviour. A larger lookahead distance tends to make the agent converge to the path more gradually, with less oscillation.

[4]Image source: www.hokuyo-aut.co.jp
[5]Image source: www.mathworks.com

2.8 Velocity Obstacles

The aforementioned techniques are sufficient to navigate through static environments. For dynamic environments, a formal theory of moving obstacles is needed.
A Velocity Obstacle (VO, Figure 2.6) maps a dynamic environment to the robot's velocity space by defining a set of robot velocities that would result in a collision with an obstacle moving at a given velocity (Fiorini & Shiller, 1998). The manoeuvre to avoid the obstacle is computed by selecting a velocity that is not in this set, considering the robot's acceleration constraints. The trajectory from the start to end point can be computed by searching a tree of possible avoidance manoeuvres at discrete time intervals. This tree can be searched in a computationally efficient manner by using a heuristic approach, where it is pruned to achieve a prioritised set of objectives: avoiding collisions, reaching the goal, maximising speed, and computing trajectories with desirable topology. The advantages of using this approach over previous methods are the following:

- Efficient geometric representation of potential avoidance manoeuvres
- Any number of obstacles can be avoided using the union of their VO's
- Unifies avoiding moving and stationary obstacles
- Allows for simple consideration of robot dynamics and actuator constraints

In an actual robotics system, the location and velocity of moving obstacles need to be tracked somehow. One way of doing this is using MHT.

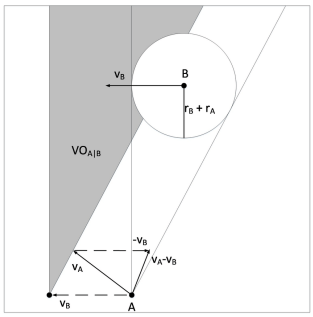


Figure 2.6: Velocity Obstacle VO_B of the moving object B as seen by the robot A. Object B is enlarged by the radius of A.[6]

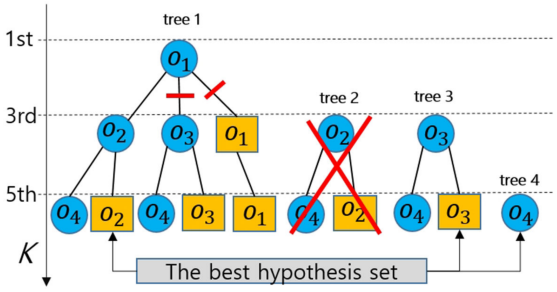


Figure 2.7: Pruning of MHT trees after the best set has been computed (Yoon et al., 2018).

2.9 Multiple Hypothesis Tracking

Multiple hypothesis tracking (MHT) is a method of tracking multiple objects in a scene (Yoon et al., 2018). It forms track-hypothesis trees, of which each branch represents the track of a target. Multi-target tracking is performed by manipulating the status of each track hypothesis (Figure 2.7). This status represents three different tracking stages:

- tracking: targets are being tracked within the view of the camera
- searching: the target has left the view of the camera, and the camera waits for the target to reappear
- end-of-track: the target is lost because it was not found again in time

[6]Image source: (Choi, Rubenecia, Shon, & Choi, 2017)

2.10 YOLOv3

In order to track objects with MHT, objects need to be detected first. You Only Look Once (YOLO) is a family of real-time object recognition algorithms. In the DRL-VO model, YOLOv3 (Redmon & Farhadi, 2018) is used. It predicts bounding boxes for objects and assigns an 'objectness' score to each box using logistic regression. Boxes are predicted at 3 different arbitrarily chosen scales. YOLOv3 struggles to align the boxes perfectly to objects, but the authors argue that this matters little since humans can hardly tell the difference. For each box in the picture, YOLOv3 predicts which class labels the box may contain. It uses multilabel classification, because images can contain overlapping labels (such as 'Woman' and 'Person'). One of the main contributions of version 3 is a new feature extraction network called Darknet-53 (Figure 2.8), which is comprised of a structure that better utilises the GPU. All of this comes together to form a fast and accurate object detector.

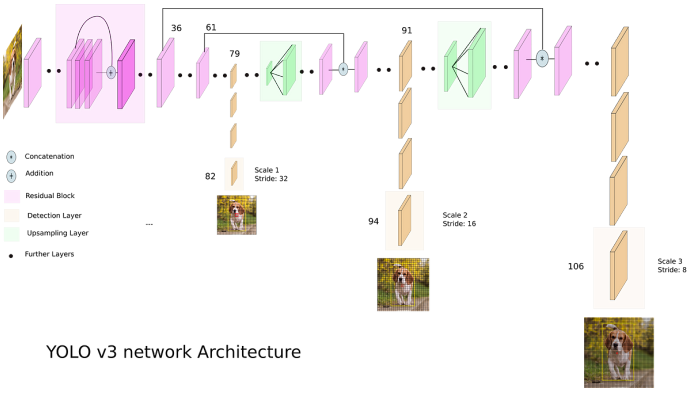


Figure 2.8: YOLOv3 network architecture.[7]

Figure 2.9: ZED stereo camera.[8]

Figure 2.10: RGB (left) and depth image (right) from the ZED stereo camera (Tadic et al., 2022). Darker objects are closer to the camera.

2.11 ZED stereo camera

The ZED stereo camera (Figure 2.9) is a camera developed by Stereolabs that can capture an RGB image with depth information (Figure 2.10), referred to as an RGB-D image together (Tadic et al., 2022). It is a passive ranging tool and therefore does not contain any active sensors such as an IR laser projector. It uses a bipolar camera, along with a stereo matching algorithm to measure the disparity of objects, to calculate a depth map. It is equipped with two 4 mega-pixel RGB sensors providing a video resolution of 2560 x 1440 pixels (2K). The cameras are spaced 12 cm apart, which can generate a depth image with a maximum depth of 40 m with the latest firmware. The depth sensor is optimised for real-time depth calculation using NVidia Compute Unified Device Architecture technology, so a corresponding GPU and appropriate hardware is required to use it. The device also supports multiple software packages, such as ROS, MATLAB, and Python, which allows the user to modify its parameters based on their requirements.
The RGB image from the ZED camera is fed into the YOLOv3 detector to detect pedestrians and the depth image is used to measure their position for input to the MHT.

[7]Image source: miro.medium.com
[8]Image source: dl.acm.org

Chapter 3

Method

This chapter will explain the basis for this thesis, the DRL-VO model, in greater detail, as well as present adjustments to this model and the methods used to train and test the used models. The method used in this thesis is based on previous work by Temple University (Xie & Dames, 2023). I chose this model because it achieved the highest score in the simulation rounds of the 2022 ICRA BARN challenge.

3.1 Previous work: DRL-VO

The DRL-VO (Xie & Dames, 2023) model was designed to solve the problem of navigating through dynamic scenes crowded with pedestrians. It uses Deep Reinforcement Learning (DRL) to learn a control policy to avoid both static obstacles and dynamic pedestrians using the concept of Velocity Obstacles (VO). The paper's main contribution is a novel reward function design that incorporates aforementioned VO's.

The model is based on previous works that use deep neural networks to learn an end-to-end navigation policy, where raw motion commands are generated from raw sensor data. One of these is the network by Pfeiffer et al. (Pfeiffer et al., 2017), which learns to generate linear and angular velocities from LiDAR data. One problem of only using raw 2D LiDAR data is that dynamic obstacles cannot be distinguished from static obstacles. To address this issue, Xie & Dames use a ZED stereo camera in conjunction with a multiple hypothesis tracker (MHT) to track the dynamics of pedestrians. Furthermore, previous works that design pedestrian-avoiding navigation systems usually only focus on exploiting the sensor perception information and its data representations, while ignoring the reward function design. Xu & Karamouzas (2021) make progress towards designing a suitable reward function by adding a 'knowledge distillation' term, which encourages the robot to learn expert behaviours. However, its effect depends largely on the quality of the expert dataset. Uncontrolled and crowded environments make collecting effective training data hard, and manually designing a general model is difficult as well, which is why Xie & Dames opt for a reinforcement-learning based framework as opposed to a traditional model-based approach or supervised learning.

3.1.1 Problem definition

The robot navigates through the environment by making a partial observation $\mathbf{o}^t$ and feeding this into a control policy π_θ to compute the appropriate action $\mathbf{a}^t$, defined as

$$\mathbf{a}^t \sim \pi_\theta(\mathbf{a}^t|\mathbf{o}^t) \tag{3.1}$$

where θ are the policy's parameters. Due to the limited field of view of the sensors, walls obstructing parts of the environment, and inherent sensor noise, there is no complete perception of the environment. This kind

of decision-making process can be defined as a partially observable Markov decision process (POMDP), denoted by the 6-tuple

$$< S, A, T, R, \Omega, O > \tag{3.2}$$

where S is the state space, A the action space, T a state-transistion model, R the reward function, Ω the observation space and O the observation probability distribution. One of the most well-known tools to solve such large POMDPs is using deep reinforcement learning (DRL). The objective $L(\theta)$ is to maximise the expected discounted return by optimising the NN-based policy π_θ:

$$L(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R^t\right] \tag{3.3}$$

where γ^t is the discount rate in the range $[0, 1]$. The DRL network consists of 4 modules: feature generation, the feature extractor, the actor and the critic. The first of these, the feature generation module, generates the partial observation $\mathbf{o}^t$ from the raw sensor data. The feature extractor module uses $\mathbf{o}^t$ to extract high-level features. The critic module generates the state value, and the actor module generates the steering action $\mathbf{a}^t$.

3.1.2 Observation space

The partial observation $\mathbf{o}^t = [\mathbf{l}^t, \mathbf{p}^t, \mathbf{g}^t]$ in the POMDP's observation space consists of three parts:

- $\mathbf{l}^t$ is a short history of LiDAR data, covering 0.5 seconds. A combination of minimum and average pooling is applied to each scan, as this method of preprocessing was found to yield the best results. The pooled data is stacked 4 times to form an 80x80 grid (Figure 3.1). This grid is a hand-crafted intermediate feature which improves generalisability.

- p^t represents the kinematics of pedestrians. These are encoded as a pair of 80x80 grids called the *pedestrian map*: one grid encodes the relative positions of all pedestrians in a 20x20 m area in front of the robot, the other encodes their velocities. To create these grids, the RGB image from a ZED depth camera (Section 2.11) is fed into the YOLOv3 detector (Section 2.10). This detector detects the pedestrians, and their relative positions can be calculated using the ZED's depth image. This data is then fed into a multiple hypothesis tracker (MHT, Section 2.9) to track their velocities. The use of these grids as intermediate features not only improves generalisability, but also allows for easy swapping of YOLO and MHT in favour of other techniques.

- g^t encodes the sub-goal position, selected by the pure pursuit algorithm (Section 2.7). It uses a look-ahead distance of 2m in training and deployment on the TurtleBot platform. This value can be tuned to make the robot look ahead or look nearby more: in the deployment code for the BARN benchmark on the Clearpath Jackal, this value was changed to 1m to account for the narrow passageways in BARN's highly constrained environments.

All of these features are expressed in the robot's local reference frame, which improves robustness against robot localisation errors. It is also more natural to do so in navigation problems, since it is more important to avoid nearby obstacles than to get the absolute position exactly right. All observation data is normalised to $[-1, 1]$ using the Max-Abs scaling procedure (Equation 3.4).

$$o^t = 2\frac{o^t - o^t_{\text{min}}}{o^t_{\text{max}} - o^t_{\text{min}}} - 1 \tag{3.4}$$

3.1.3 Action space

The action $a^t = [v_x^t, \omega_z^t]$ in the POMDP's action space consists of the forward velocity v_x^t and the rotational velocity ω_z^t. These values are capped to the ranges $[0, 0.5]$ m/s and $[-2, 2]$ rad/s respectively, based on the Turtlebot2 platform on which the original DRL-VO policy was trained. Like the observation space, the action space is also normalised using the Max-Abs scaling procedure.
In the deployment code for the BARN benchmark the maximum velocity was adjusted for the Clearpath Jackal robot to be 2 m/s. However, directly mapping the normalised velocity from the control policy to the Jackal's speed range leads to agressive behaviour and frequent collisions. To prevent this, a simple maximum velocity switching mechanism is used:

$$v_{\max} = \begin{cases} 2 \text{ m/s} & \text{if } d_{\text{obs}}^t \geq d_f \\ 0.5 \text{ m/s} & \text{otherwise} \end{cases} \tag{3.5}$$

where d_{obs}^t is the distance to the nearest obstacle at time s, and d_f is a threshold value (set to 2.2m)

3.1.4 Network Architecture

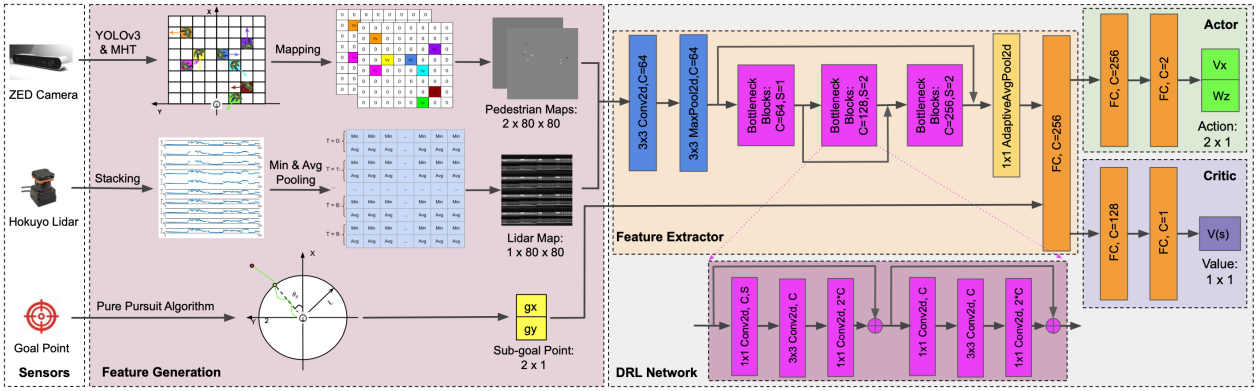


Figure 3.1: The neural network architecture for the DRL-VO control policy (Xie & Dames, 2023). Raw sensor data is preprocessed into 80x80 intermediate features and fed into a feature extractor, and fused with the sub-goal point information. The resulting high-level features are fed into the actor and critic networks seperately.

The network architecture (Figure 3.1) is that of a middle fusion network, where different streams of information are fused halfway in the chain between raw input and output. The lidar history observation l^t and the pedestrian kinematics observation p^t are fused, after which the feature extraction module extracts high-level featured from the fused information stream. After the high-level features are extracted, a single fully-connected (FC) layer fuses these with the sub-goal position. This then feeds into the critic and actor modules, which both consist of two FC layers.

3.1.5 Reward function

The reward function used in DRL-VO is a multi-objective reward function with four terms:

$$r^t = r_g^t + r_c^t + r_w^t + r_d^t \tag{3.6}$$

where r_g^t rewards making progress towards the goal position, r_c^t penalises passively approaching or colliding with an obstacle, r_w^t penalises rapid changes in direction and r_d^t rewards actively steering to avoid obstacles

and move towards the sub-goal. The term r_d^t is a knowledge-distillation term, meant to learn good navigation behaviour from successful model-based navigation policies. It uses velocity obstacles (VO) to set a desired navigation direction, which allows it to use first-order kinematic information (velocity) to avoid collisions and move towards the goal.

$$r_d^t = r_{\text{angle}}(\theta_m - |\theta_d^t|) \tag{3.7}$$

where r is the reward weight of the angle (0.6), θ_m is the maximum allowable deviation of the heading direction ($\pi/6$ rad) and θ_d^t is the desired heading direction. This last term is calculated by looping over N possible headings in the range $[-\pi, \pi]$ and looping over every pedestrian, finding the smallest deviation from the sub-goal heading that does not lead to collisions with pedestrians (Algorithm 1).

Algorithm 1 Search desired steering angle

$\theta_d^t \leftarrow \frac{\pi}{2}$ ▷ Initialise desired steering angle
if $B_{\text{peds}} \neq \emptyset$ **then** ▷ B_{peds}: pedestrians from MHT
  $\theta_{\text{min}} \leftarrow \infty$
  for $i = 1, 2 \ldots, N$ **do** ▷ N: number of samples
    $\theta_u \leftarrow$ sample from $[-\pi, \pi]$
    free $\leftarrow$ True
    for $B \in B_{\text{peds}}$ **do**
      $\theta_{v_{A,B}} \leftarrow \text{atan2}\left(\frac{\mathbf{v}_{Ax}\sin(\theta_u) - \mathbf{v}_{By}}{\mathbf{v}_{Ay}\cos(\theta_u) - \mathbf{v}_{Bx}}\right)$ ▷ $\mathbf{v}_A$: robot velocity, $\mathbf{v}_B$: obstacle velocity
      $\theta, \beta \leftarrow$ from geometric relationships using B
      if $\theta_{v_{A,B}} \in [\theta - \beta, \theta + \beta]$ **then**
        free $\leftarrow$ False
        break
      end if
    end for
    if free **then**
      if $|\theta_u - \theta_g| < \theta_{\text{min}}$ **then** ▷ θ_g: sub-goal angle
        min $\leftarrow |\theta_u - \theta_g|$
        $\theta_d^t \leftarrow \theta_u$
      end if
    end if
  end for
else
  $\theta_d^t \leftarrow \theta_g$ ▷ There are no pedestrians; steer straight to goal
end if

3.2 Experiment

In order to test the effect of training on velocity obstacles on navigation performance in highly constrained static environments, I trained two variants of the previously discussed model: DRL-VO and DRL. The difference between these is that DRL's reward function does not contain the term r_d^t which calculates a desired steering direction based on velocity obstacles. Xie & Dames (2023) also trained both of these variants of the model, but never applied DRL to the BARN benchmark. In my experiment I trained both of these models on a single machine and ran detailed tests for both of these variants on the BARN benchmark.
Both models were trained on a single machine with an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 62GiB memory and an NVIDIA GeForce graphics card with 11264MiB memory. The models were tested on a machine with an Intel(R) Core(TM) i9-9900 CPU @ 3.10GH, 62GiB memory and an NVIDIA GeForce graphics card with 11264MiB memory. Both models were trained using the open-sourced training code found at `https://github.com/TempleRAIL/drl_vo_nav` and tested on 50 BARN worlds using the BARN deployment code found at `https://github.com/TempleRAIL/nav-competition-icra2022-drl-vo`.

3.2.1 Model training

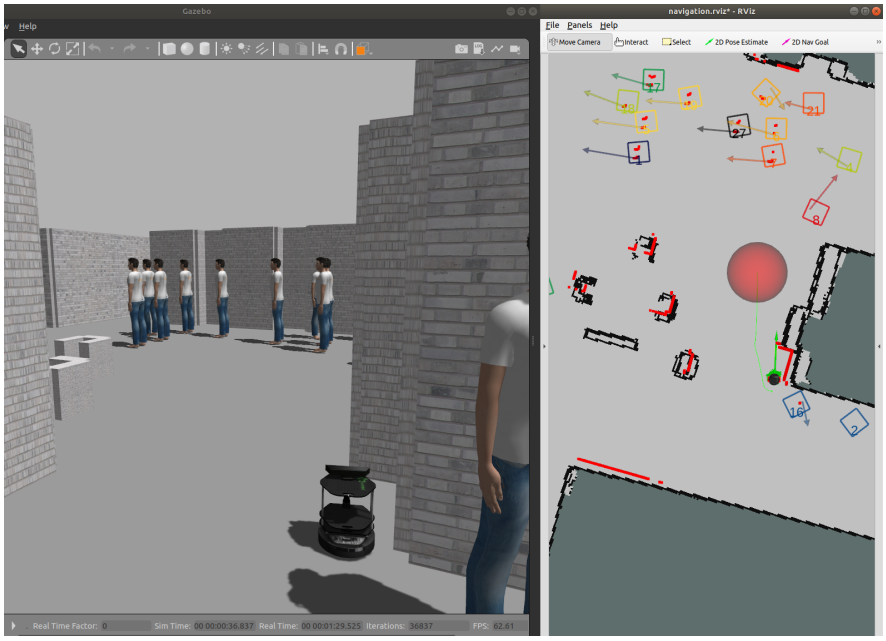


Figure 3.2: Visualisation of training the DRL-VO based navigation policy in the Gazebo simulator.

Models were trained in the Gazebo simulator (Figure 3.2) on a single simulated world of Temple Univerity's lobby containing 34 mobile pedestrians. The robot is repeatedly moved to a random start position and assigned a random goal position to navigate to. The DRL-VO navigation policy was trained using the open-sourced training code with several adjustments to run it on the local machine. A few notable ones include:

- Importing the `tutlebot_env` package, which provides the training environment for `gym`, in a different manner because the standard `setup.py` did not seem to work.
- Changing the mount configuration for the Singularity container containing all training packages.

The DRL navigation policy was trained using the same code as DRL-VO, with the knowledge distillation term r_d^t removed from the reward function. Both models were trained using 3 million simulation timesteps, where 1000 timesteps represent one second. Training DRL-VO and DRL on the local machine using these settings took approximately 10 and 7 days, respectively.

3.2.2 Model testing

Both models' performance was evaluated on the BARN benchmark using the aforementioned adjusted BARN deployment code. Tests were performed on 50 different BARN worlds equally sampled from the 300 worlds, i.e. world 0, 6, 12, etc. Tests were performed 10 times on each world, and these also take place in a simulated environment in Gazebo (Figure 3.3). In these tests the robot is initialised in a fixed start position and tasked to navigate to a fixed goal position as quickly as possible without colliding with any obstacles. Both models took approximately 3 days to test each.

In order to visualise the robot's behaviour within the BARN worlds, I ran additional tests on several worlds where the robot's entire traversed path is logged. This allows the robot's navigation behaviour to be plotted and analysed more thoroughly, as will be discussed in the next chapter.

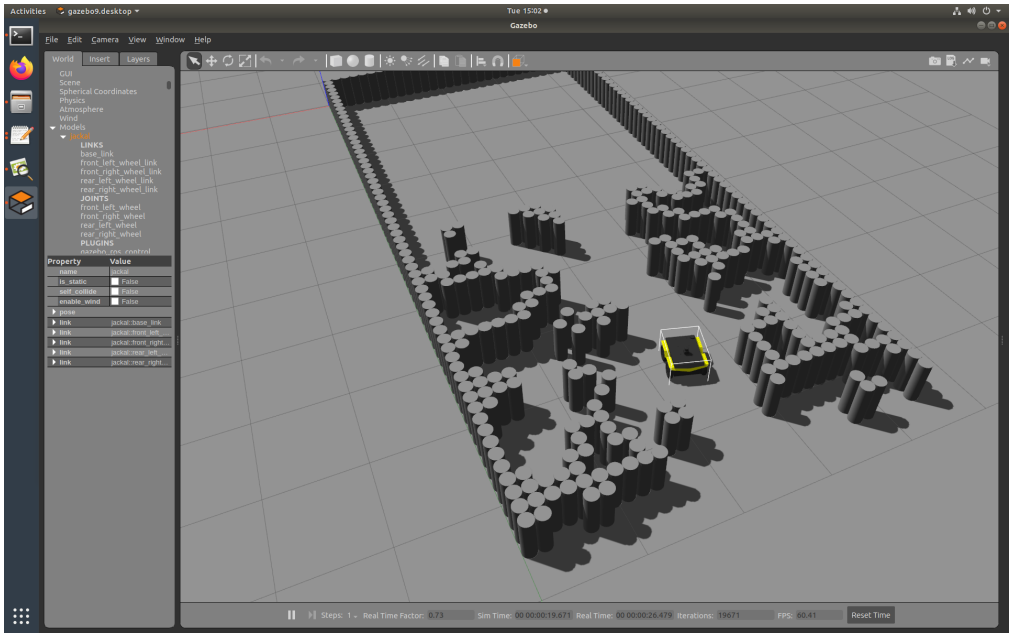


Figure 3.3: Visualisation of testing the DRL based naviation policy in the Gazebo simulator. The depicted environment is BARN world 200.

Chapter 4

Results

This chapter will show the results obtained from testing the navigation policies trained using DRL and using DRL-VO on the BARN benchmark. Apart from the navigation metric mentioned in the introduction of this thesis, which is used as an absolute metric of which model performs better, the following metrics are available:

- Average navigation time: the time that it took the robot to navigate from the start position to the goal position, averaged over all trials where the end was reached
- Average success rate: the number of trials where the goal was reached, divided by the total number of trials
- Average collision rate: the number of trials where the robot collided with an obstacle, divided by the total number of trials
- Average timeout rate: the number of trials where the navigation timed out before the robot could reach the goal, divided by the total number of trials. A timeout occurs when the robot does not reach the goal within 100 seconds.

Table 4.1: Mean performance of DRL-VO and DRL on BARN worlds.

	Avg. time	Navigation metric	Avg. success	Avg. collision	Avg. timeout
DRL-VO	**25.3583**	0.0716	0.3020	0.3700	0.3280
DRL	28.8610	**0.1273**	**0.5880**	**0.3680**	**0.0440**

As can be seen in table 4.1, the DRL trained navigation policy achieved, perhaps surprisingly, a higher navigation metric on the BARN benchmark than DRL-VO. Unfortunately these results do not reproduce the results found by the original authors (Xie & Dames, 2023), but they do show some interesting patterns.
Firstly, DRL tends to succeed on more trials than DRL-VO, while both models have similar amounts of collisions, because DRL-VO times out rather often. Looking at the outcome distribution per world (Figure 4.1), we can see that DRL-VO mostly times out in higher, more constrained BARN worlds. Both navigation policies tend to have more success on easier, more open BARN worlds.

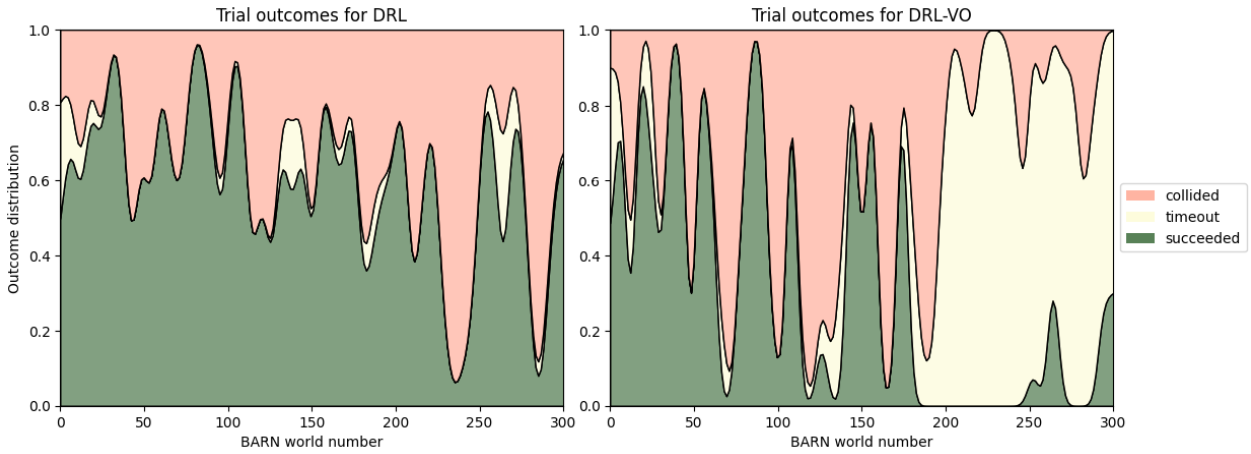


Figure 4.1: Trial outcomes for the control policy trained by DRL (left) and DRL-VO (right) on BARN worlds. The outputs can either be: succeeded (reached the goal position in time), timeout (spent over 100 seconds navigating without reaching the goal) or collided (collided with an obstacle).

In order to visualise the policies' behaviour within a single trial, figures 4.2, 4.4, 4.6 and 4.7 show these trials plotted out within the BARN worlds. 10 paths from the start to goal position are plotted within the BARN environment, with markers to denote where navigation stopped: a red marker denotes a collision, an orange marker denotes a timeout and a green marker denotes success. Both policies' paths are shown in a different colour. The start and goal position are shown with the letters S and G, respectively.

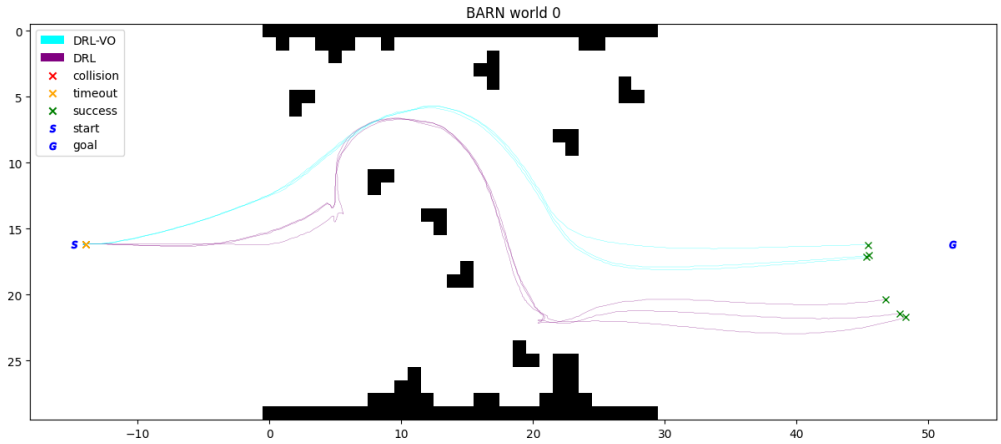


Figure 4.2: 10 trials on BARN world 0 by the DRL and DRL-VO navigation policies.

Figure 4.2 shows the traversed paths on BARN world 0. Two behaviours can be observed here: firstly, the majority of trials timed out. Both models succeeded 3 times and timed out 7 times on this test. Each of these timeouts occured at the start position, meaning the robot did not move at all. Other tests where timeouts occur show simular results.
Secondly, the DRL-VO based navigation seems to follow a smoother path than DRL, which allows it to complete the navigation faster: DRL-VO had an average time of 23.75 seconds on this test, whereas DRL had an average time of 30.86 seconds. DRL can be seen to nearly collide with the wall, slow down, and make sharp turns on two locations. DRL-VO navigates slightly faster in most worlds, as can be seen in figure 4.3.

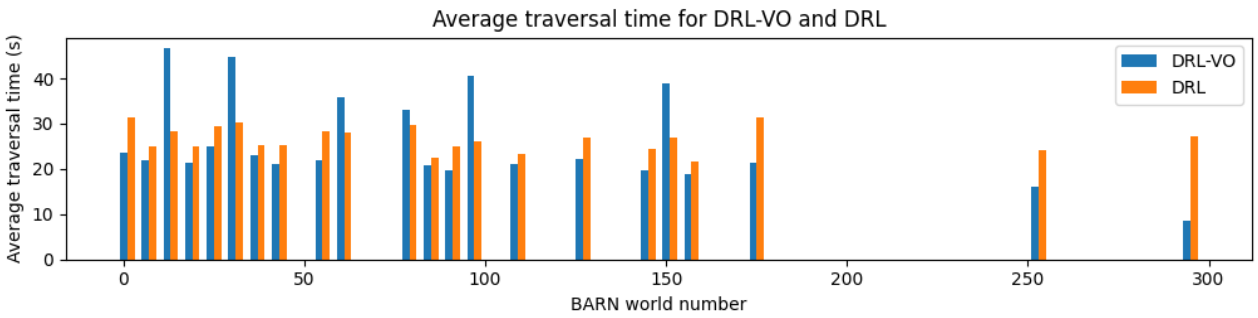


Figure 4.3: Average traversal time for only the 22 BARN worlds that were completed at least once by both models. In 16 of the 22 cases, the DRL-VO navigation policy navigated faster than DRL.

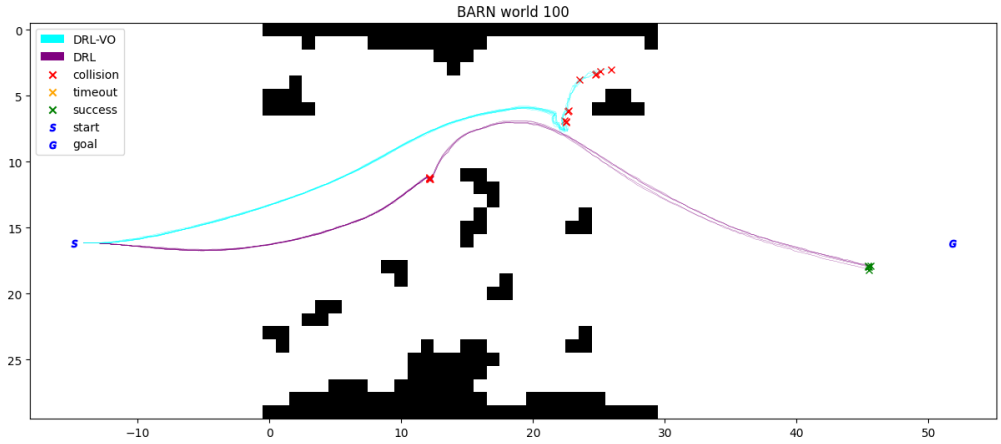


Figure 4.4: 10 trials on BARN world 100 by the DRL and DRL-VO navigation policies.

In BARN world 100 (Figure 4.4), similar behaviour can be observed. Like in world 0, and in all worlds hereafter, DRL-VO tends to drive more to the left and DRL more to the right. In world 100, DRL once again makes a sharp turn in the middle of the level. However, it collided with the obstacle 5 out of 10 times. In figure 4.5, it can be seen that the robot nearly scratches the wall in this manoeuvre, and this seems to result in a collision half of the time. DRL-VO, on the other hand, collided every time on this level. It can be seen that due to its tendency to stay more to the left, it is unable to avoid the last obstacle and tries to go around the left side of it, where there is not enough room.

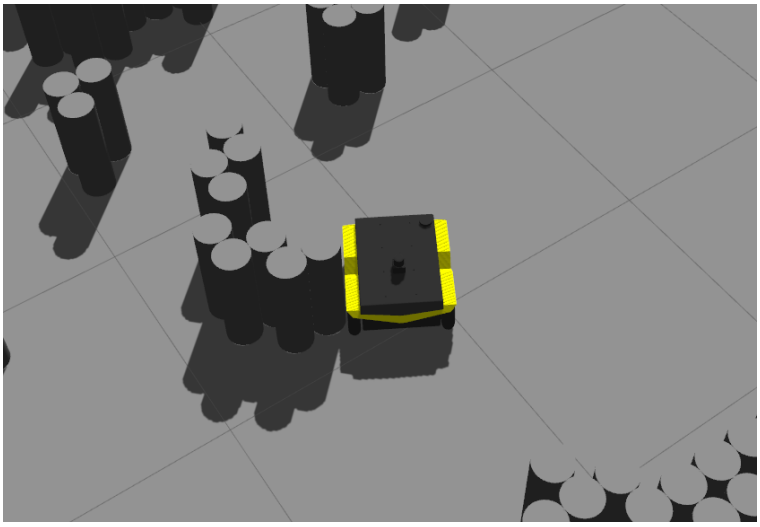


Figure 4.5: A robot controlled by the DRL based navigation policy narrowly avoiding collision with an obstacle on BARN world 100.

This behaviour is also present in higher number BARN worlds. In world 200 (Figure 4.6), DRL succeeds every time, but DRL-VO fails every time because it stays too much to the left and cannot get back on track again. In world 299 (Figure 4.7), both navigation policies are unable to get to the goal position, but both policies are exploring two distinct regions of the environment. In all worlds, DRL tends to stay close to obstacles. One possible explanation for this behaviour could be that DRL, which did not have an 'expert teacher' based on velocity obstacles, collided more often with pedestrians in open spaces than DRL-VO and learned to stick close to the walls of the environment. This kind of behaviour seems to bear fruit in the highly constrained BARN environments.

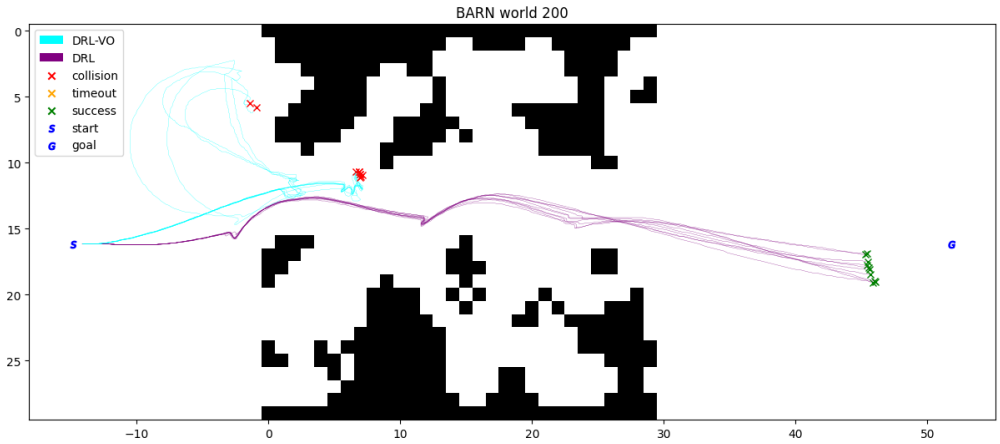


Figure 4.6: 10 trials on BARN world 200 by the DRL and DRL-VO navigation policies.

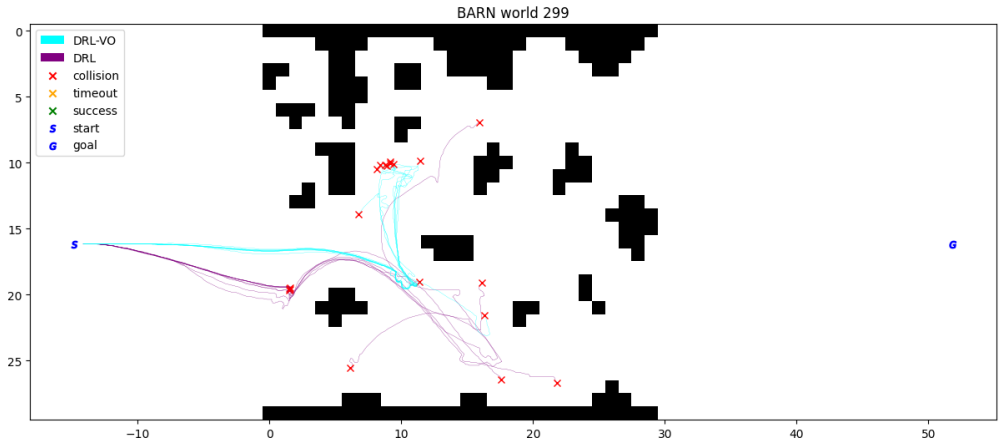


Figure 4.7: 10 trials on BARN world 299 by the DRL and DRL-VO navigation policies.

Most of the failed trials seem to collide with an obstacle pretty quickly. However, it is also possible for the navigation policy to avoid collisions as seen in figure 4.8. Here, all trials end in collision except one trial by the DRL-VO control policy that timed out (highlighted as a bold line). On this trial, the robot moved left and right repeatedly but failed to pass through the opening.

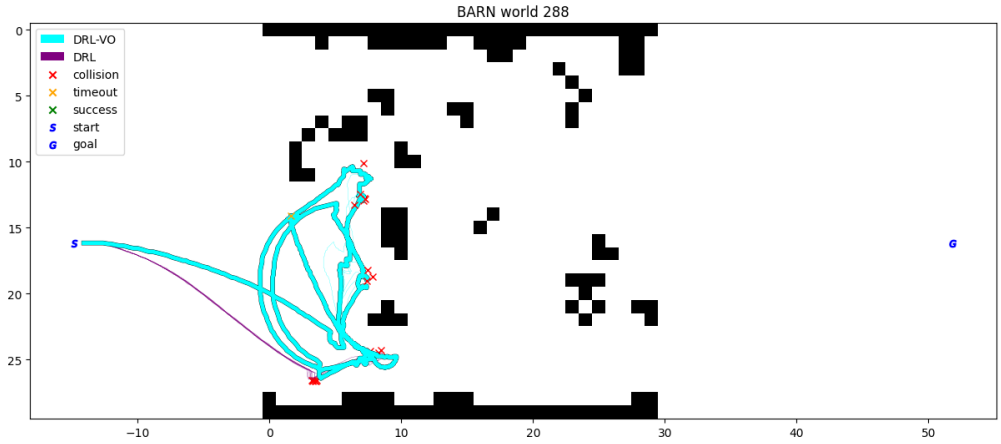


Figure 4.8: 10 trials on BARN world 288 by the DRL and DRL-VO navigation policies. The bold highlighted path shows a 100 second long navigation trial produced by the DRL-VO based policy.

Chapter 5

Conclusion

In this thesis I asked the question of how distilled knowledge on avoiding moving obstacles affects navigation performance in challenging static environments. The results presented in the previous chapter show that there is a clear difference in performance between the DRL and DRL-VO control policies in the BARN benchmark. Perhaps one of the most surprising observations is that the model that learned behaviour from the VO model-based policy had an average lower navigation metric than its simpler counterpart. Even though the DRL-VO based policy has a greater average speed, it suffers great performance loss due to frequent time-out and slightly higher collision rates compered to the DRL based policy.
Even though the BARN environments contain no moving obstacles, both policies generate different paths through the environment. The DRL based policy tends to stick close to obstacles, which reduces its average speed but increases its overall ability to reach the goal.

The results obtained in this thesis show patterns that indicate differences in the behaviour of both navigation policies, but they also contain a great degree of noise. This is mainly emergent from the fact that only 10 trials were conducted on each world. One direction for future research may be to run many more tests on each BARN world, using a more powerful machine or compute cluster. A second improvement that might make the results more reliable is to train the models with the same simulated hardware as is used in the BARN challenge. For instance, the size difference of the Clearpath Jackal used in testing versus the Turtlebot used in training could lead to extra collisions. The paper by Xie & Dames and results from this thesis show that their policy is generalisable across different hardware, but taking away the hardware differences may produce results that better reflect the impact of only removing the knowledge diffusion reward term. A pedestrian detection system using only LiDAR data (Lin et al., 2018) could also be used to remove the necessity of an RGB-D camera during training, which is not present in the BARN hardware either.

Chapter 6

Discussion

In this chapter, I will address some limitations of this thesis and reflect on the lessons learned.

Unfortunately, due to lack of time, I did not have the opportunity to test out more different models. A lot of time was spent in the beginning of this project on getting the models to run on a local machine, and training both models took about 2 and a half weeks. Testing both models on 500 BARN trials each took another week, and performing more tests where the full paths are logged also took more time. This time-consuming testing process contributed to the noisiness of the results: the sample size on each barn worlds is very small (10 trials per world), and as a result the test data might not be entirely representative of the average performance. This could be improved by running more trials.
I spent most of my time researching the workings of the model and trying to get the model to run on a local machine. This comes mainly from the fact that this is a state-of-the-art model, integrating many different techniques into one system. Looking back, it might have been better to choose a simpler model, which might have saved time on setting up the training- and testing pipelines and left more time to actually use them. Nonetheless, I do feel that I learned a lot from researching all the various techniques involved and from the whole process of working with this model.

This project was originally meant to be a two-man project, but unfortunately my partner could not continue about halfway into the project. This led to some reconsiderations in the scope of the project, mainly applying the DRL-VO model to the BARN benchmark instead of designing our own model.

Due to the reasons mentioned above, I was only able to train and test the DRL-VO and DRL models as described originally in the paper by Xie and Dames. It would have been preferable to train more variants of the models, in order to gain a better understanding of factors that influence navigation behaviour. Such variants could be to remove the pedestrians from the training environment, in order to see how the presence of moving obstacles in training effects performance in BARN's constrained environments. Another direction for future work may be to train the models on the Clearpath Jackal robotic platform as well, since that is what is used in the BARN benchmark. The authors of the DRL-VO paper originally participated in the BARN challenge to test the model's generalisability across different environments and different robotic platforms. Even though the policy is generalisable to the Jackal and can perform well in the BARN tests, using the same robot in training may reduce influence by factors other than changing the reward function design.

As mentioned in the introduction, the model used in this thesis is no longer the leader in the BARN challenge. The current top-3 consists of the teams KU Leuven & Flanders Make, Inventec Corporation, and University of Almeria & Teladoc. For all three of these, no policy code has been publically released yet and I have not

found any papers documenting their solutions. It would be interesting to compare these newer models once they become available, to see how they have improved upon last year's winners. This year's winners have also managed to attain nearly the highest possible score in the BARN challenge, which begs the question of whether or not the same challenge can be held next year. Perhaps it is necessary to set the bar higher next time around in order to account for the progress that has been made by the teams. An adjustment to increase the difficulty of the challenge could be to no longer clip the actual traversal time between 4 and 8 times the optimal time, rewarding navigation policies that can reach the goal faster.

References

Choi, M., Rubenecia, A., Shon, T., & Choi, H. H. (2017). Velocity obstacle based 3d collision avoidance scheme for low-cost micro uavs. *Sustainability*, *9*(7), 1174.

Coulter, R. C. (1992). *Implementation of the pure pursuit path tracking algorithm* (Tech. Rep.). Carnegie-Mellon UNIV Pittsburgh PA Robotics INST.

Fiorini, P., & Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research*, *17*(7), 760–772.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23–33.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koenig, N., & Howard, A. (2004). 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).

Lin, T.-C., Tan, D. S., Tang, H.-l., Chien, S.-c., Chang, F.-c., Chen, Y.-Y., ... Hua, K.-L. (2018). Pedestrian detection from lidar data via cooperative deep and hand-crafted features. In *2018 25th ieee international conference on image processing (icip)* (pp. 1922–1926).

Mazzeo, G., & Staffa, M. (2020). Tros: Protecting humanoids ros from privileged attackers. *International Journal of Social Robotics*, *12*, 827–841.

Oliehoek, F. A., & Visser, A. (2010). A decision-theoretic approach to collaboration: Principal description methods and efficient heuristic approximations. *Interactive collaborative information systems*, 87–124.

Perille, D., Truong, A., Xiao, X., & Stone, P. (2020). Benchmarking metric ground navigation. In *2020 ieee international symposium on safety, security, and rescue robotics (ssrr)* (pp. 116–121).

Pfeiffer, M., Schaeuble, M., Nieto, J., Siegwart, R., & Cadena, C. (2017). From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 ieee international conference on robotics and automation (icra)* (pp. 1527–1533).

Pineau, J., Gordon, G., & Thrun, S. (2006). Anytime point-based approximations for large pomdps. *Journal of Artificial Intelligence Research*, *27*, 335–380.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... others (2009). Ros: an open-source robot operating system. In *Icra workshop on open source software* (Vol. 3, p. 5).

Raj, T., Hanim Hashim, F., Baseri Huddin, A., Ibrahim, M. F., & Hussain, A. (2020). A survey on lidar scanning mechanisms. *Electronics*, *9*(5), 741.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Riisgaard, S., & Blas, M. R. (2003). Slam for dummies. *A Tutorial Approach to Simultaneous Localization and Mapping*, *22*(1-127), 126.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Tadic, V., Toth, A., Vizvari, Z., Klincsik, M., Sari, Z., Sarcevic, P., ... Biro, I. (2022). Perspectives of realsense and zed depth sensors for robotic vision applications. *Machines*, *10*(3), 183.

Xie, Z., & Dames, P. (2023). Drl-vo: Learning to navigate through crowded dynamic scenes using velocity obstacles. *arXiv preprint arXiv:2301.06512*.

Xu, P., & Karamouzas, I. (2021). Human-inspired multi-agent navigation using knowledge distillation. In *2021 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 8105–8112).

Yoon, K., Song, Y.-m., & Jeon, M. (2018). Multiple hypothesis tracking algorithm for multi-target multi-camera tracking with disjoint views. *IET Image Processing*, *12*(7), 1175–1184.

Chapter 7

Other algorithms

This appendix contains a description of techniques not used in this thesis, but that are used in many other navigation systems, including several BARN challenge participants.

7.1 DWA

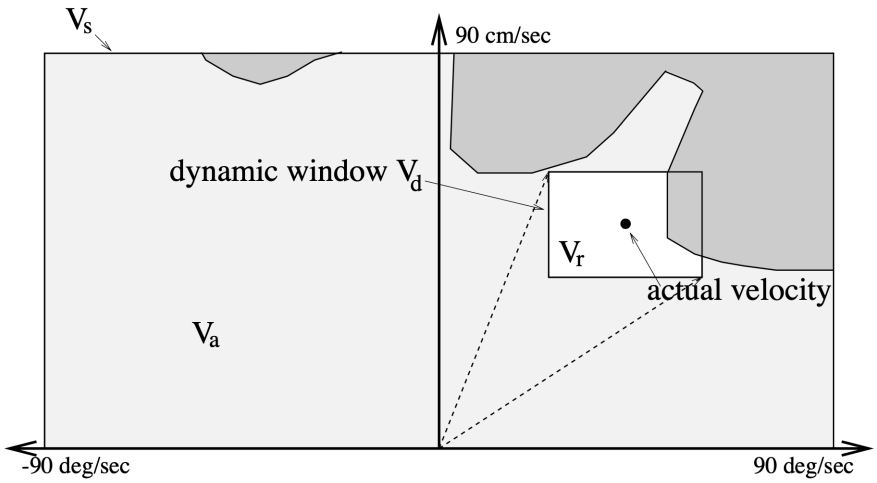


Figure 7.1: Dynamic window within a velocity space (Fox et al., 1997)

(Fox et al., 1997)
The Dynamic Window Approach (DWA) is an approach to reactive collision avoidance for mobile robots. It was originally designed for those equipped with synchro-drives: multiple wheels attached to two motor-driven chains, where one chain controls the direction of the wheels and the other controls the speed. The approach is derived directly from the robot's motion dynamics, making it particularly well-suited for operating at high speed. The 'Dynamic Window' is the two-dimensional space of translational and rotational velocities that can be reached by the robot within a short time interval. This space is centred around the robot's current velocities, and reduced to filter out any velocities that would result in a collision. Among the admissible velocities within the dynamic window, one velocity is chosen by optimising an objective function. This function favours velocities that allow the robot to make progress towards the goal. It also includes measures of the robot's velocity and the distance to the next obstacle on the trajectory, which allows it to find a trade-off between getting to the goal position fast and taking a safe route.

7.2 SLAM

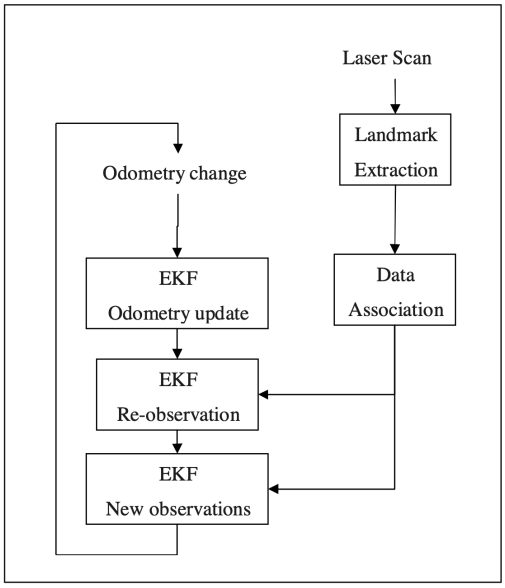


Figure 7.2: Overview of the SLAM process (Riisgaard & Blas, 2003)

(Riisgaard & Blas, 2003) Simultaneous Localisation And Mapping (SLAM) is a technique for a robot to localise itself in and build a map of its environment. It consists of multiple parts: landmark extraction, data association, state estimation, state update and landmark update. Each of these parts can be solved in multiple different ways, depending on available hardware and software. - Landmark extraction is usually done by finding features in a laser scan, sonar scan, or visual image. - Data association is the problem of matching landmarks in different observations. One way of doing this could be to store all previously observed landmarks in a database, and to use the nearest neighbour method to associate the newly observed data with one of the previous observations. State estimation is usually done using the Extended Kalman Filter (EKF). In the EKF, the robot's position is represented as a normal distribution. The robot's odometry is used to calculate a prior for the new position. This is simply the sum of the previous position of the robot and the trajectory that it has moved since the last timestep. Since odometry is not perfect, adding it to the current position increases uncertainty. The estimated state must be updated with the newly observed landmarks. The sensor that provides these observations is not perfect either, so we calculate the Kalman Gain (K) as a measure of 'trust' for the observations. This measure dictates how much the state estimation should be updated. Lastly, the newly observed landmarks are added to the current state. The state vector contains not only the position and rotation of the robot, but also the position of all observed landmarks. This allows for localisation and mapping to occur as one process.