Autonomous Urban Environment Navigation Using Reinforcement Learning

Nathan Durocher 489942

Masters Project in Engineering (Robot Systems)
Supervisor: Leon Bodenhagen
Co-supervisor: Capra Robotics Aps

June 2022

The Maersk Mc-Kinney Moeller Institute
University of Southern Denmark

Word count: 11948

Declaration

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words.

Odense 01. 06. 2022 Nathan Durocher

Abstract

With mobile robots becoming more integrated into society, they are still being met with many challenges when navigating in human urban environments. As a result, a large effort is being made to design robotic systems that minimize their impact on existing human behaviour. As part of that work, this thesis introduces a model-based reinforcement learning method for local motion planning that complies with human social norms in crowds by learning to respect an individual's personal space. Using a sample-based action planner and a deep learning model, the method is capable of predicting unsafe outcomes of future actions using visual input only. Using high-quality simulation, the method is trained to navigate around crowded sidewalks in an urban environment, where it was compared to a state-of-the-art crowd avoidance method. The results indicate that by learning to respect pedestrians' personal space, the method was not only able to reduce the amount of time in their personal space but was also to maintain greater distances from pedestrians. Additionally, the method shows promise of the visual model trained only in simulation transferring to the real world.

Acknowledgements

Firstly, I would like to extend my gratitude to my supervisor, Leon Bodenhagen, for his supervision, support and conversation that made this project possible. I would like to thank Capra Robotics ApS; in particular Niels Jul Jacobsen for providing me with support and resources, in addition to the original project proposal. I would also like to thank Rui Pimentel de Figueiredo for his assistance and feedback throughout the project. Finally, I wish to thank my girlfriend, parents and siblings for their endless love and support throughout all my academic pursuits.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation

Robots are becoming more common and accessible in everyday life with their introduction into areas such as logistics, transportation and home automation. A major factor in the significant uptake of robots is their increasing ability to autonomously navigate and move in complex and unstructured environments. One area that has gained a lot of attention recently is autonomous vehicles (AVs), with companies like Google, Uber, and Tesla all making significant advancements in the past decade [1]. While these companies focus on road vehicles, there has also been a growing interest in deploying AVs in pedestrian-friendly urban areas such as city sidewalks, squares, and parks. Enabling robots to safely roam in these areas would revolutionize industries, with a significant impact on problems such as last-mile delivery, where it is estimated that in the average European city, mobile robots could cut delivery costs by up to 40% by 2030 [2]. Similarly, companies like Capra Robotics - whom the work in this thesis is done in collaboration with - are working on solutions for automating cleaning tasks such as removing chewing gum and cigarette butts from cities [3]. These tasks might sound reminiscent of the more common household robotic vacuum cleaners like iRobot's Roomba [4] which mainly operate in empty households. However, deploying autonomous systems in city environments provides different challenges with an increase in the variety of the possible environment such as roads, vehicles, buildings and trees. The robots that Capra and others are working on have the added challenge of larger and usually unbounded environments, where the system can not reliably use walls or barriers as aids for navigation. This is compounded with the addition of the main tenant of urban areas, pedestrians, who by nature exhibit highly unpredictable behaviours.

Humans can move through crowded areas with little to no effort, even from relatively young ages. Our decision-making in these environments can be described as a policy where we weigh our options about what route we will take and at what speed we can safely traverse that route. This policy also includes consideration for less stringent rules like social norms that, when possible we try to move in ways that account for other people by anticipating their movement. This is done so we do not interfere with their routes and potentially cause any abrupt stops or collisions and to limit the possibility of confrontation that can sometimes follow. The amount of space a person requires before they feel they need to adjust varies across cultures but previous work [5], [6] has shown that each individual has a specific space around them,

which they feel should not be violated. So, if we think back to incorporating robots into crowded areas, how can we ensure they adhere to these norms? This has already been considered by the city of Toronto, Canada where they have banned all mobile robots from operating on sidewalks and barred them from stopping in bike lanes. This decision was made based on a recommendation from the city's Accessibility Advisory Committee that the robots pose a hazard for certain portions of the population [7]. This led to the thought, what if we could teach the robot about social norms, and more specifically is it possible for them to learn to respect an individual's personal space?

A general approach to this problem is to equip the robots with various sensors to allow them to localize in the environment as well as to detect and classify possible hazards and obstacles. Then using the collected data, the robot can make decisions on which actions it should take to complete its task or reach its destination. There have been various methods developed to collect and analyze data for decision making; some actively sense the environment to help build a local map to help make an informed decision, while others record the consequences of a less informed decisions to improve for future evaluations of similar situations. Recently, the latter of these two methods has been favoured in the development of autonomous agents with the proliferation of machine learning, where in as early as 1989 Pomerleau et al. [8] were able to achieve autonomous navigation using a learning-based algorithm. In comparison to other techniques, learning provides the significant benefit of reducing the amount of input needed from a human designer and provides the possibility for patterns and behaviours to be recognized beyond what a human could program. The disadvantage to learning approaches is that they required large amounts of data to learn from which can be costly and difficult to acquire. However, thanks to the increase in computing power and the development of highly realistic simulated environments, the barriers to entry have been drastically lowered. This has been particularly useful for reinforcement learning-based approaches which have been known to require upwards of millions of training episodes to produce adequate results [9].

However, the use of multiple sensors such as LiDAR, and laser range finders adds significant complexity and cost to an autonomous navigation system and are known to underperform in inclement weather [10]. Therefore, many researchers have attempted to substitute these sensors with vision-based systems that utilize cheaper, and more readily available colour cameras. This thesis will look at building upon this area of visual-learning-based navigation by expanding on the work done by Kahn et al. in [11] and [12]. In their work, Kahn et al. use a model-based reinforcement learning technique, referred to as BADGR, to navigate in static urban environments by predicting events that will occur given the robot's current camera view and a sequence of actions. In their previous work, a deep learning model is trained using accelerometer data to learn to identify actions, that would cause the robot to travel over rougher terrain or collide with static obstacles. In their follow-up work, a

model uses disengagement signals from a supervising human to learn to drive along suburban sidewalks. The proposed method, referred to as HERDR to fit with Capra's company goat mascot and theme, utilizes the BADGR framework presented previously with a novel application and training method. This method investigates dynamic obstacle avoidance for pedestrians and learns a control policy that mimics human social norms by respecting personal space, while also learning to navigate collision-free on sidewalks and in other public spaces. The HERDR method leverages a highly realistic simulated urban environment for data collection, training, and testing, with the intent to deploy the visual model into the real world.

1.2 Problem Formulation

The problem is to navigate from one location to another in an outdoor urban environment; while avoiding collisions with obstacles and invasions of pedestrians' personal space. The robot is equipped with a single forward-facing colour camera and a learned model to determine the best actions to safely navigate at each time step. To direct the robot, the global position of both the robot and the goal are known, but the map of the environment is unknown. The task is assumed to take place on a continuous sidewalk or other open areas primarily designated for pedestrians, where the robot will not cross any roads or other car infrastructure. It is also assumed that the robot will not interact with any signalling in the environment either, such as at crosswalks or intersections. The first step will require the testing of the BADGR framework, to verify it can learn to avoid basic dynamic obstacles and then increase the complexity of the environment with more realistic buildings and more reactive pedestrians in a highly realistic simulator. There, the HERDR's avoidance capabilities will be examined based on its ability to maintain a safe distance from pedestrians and how often it enters their personal space compared to another state-of-the-art avoidance method. Additionally, an analysis of HERDR's navigation ability will be completed by looking at success rate and distance travel compared to the minimal traversable path. These metrics fall in line with those recommended in [13] to evaluate navigation methods. Finally, HERDR will be deployed on a Capra Hircus robot to verify how well a model trained in simulation can transfer to the real world.

1.3 Outline

The contents of this report, organized into 6 chapters are as follows;

Chapter 2 describes the frameworks, methods, and techniques used in the report, covering all the backgrounds necessary to follow the implementation.

Chapter 3 reviews related work by introducing and discussing differences between the work in this thesis and other methods.

Chapter 4 defines the proposed method with details of the system pipeline, model architecture and parameters, along with a description of the model training process.

Chapter 5 presents how the method was implemented and tested, where the results are presented with a comparative discussion with alternative methods.

Chapter 6 is the conclusion of the thesis where final statements are made with the inclusion of avenues for future work and improvements.

2 Background

This chapter will review the methods, model architecture and tools used during this research. These will include reinforcement learning (RL), specifically, model-based reinforcement learning, deep learning, and model architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), in particular, long short-term memory (LSTMs). Furthermore, a description of the simulator software used in this research is presented.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a machine learning technique that uses an agent's interactions with its environment and the resulting state or situation the agent is in as a means to learn and reinforce desired behaviours. The method works by defining the environment as a Markov Decision Process (MDP), where at any point in time where a decision is to be made the incident may be described by the tuple [14]: the state ($\mathbf{S}$), a set of potential actions ($\mathbf{A}$), the reward received for taking an action ($\mathbf{R_a}$) and the probabilities of moving to a new state given an action ($\mathbf{P_a}(\mathbf{s}, \mathbf{s}')$). Using this description, a model can be created of a decision process that is partially agent controller and partially random due to the neglect of other agents or forces in the environment. When utilized in a repeated manner the result is a decision loop as seen in figure 2.1 where the agent can collect a cumulative reward for taking a sequence of actions. In this model, the reward is dependent on both the state and the action, where taking the same action while in a different state can and likely should provide a different reward. However, the reward function is only

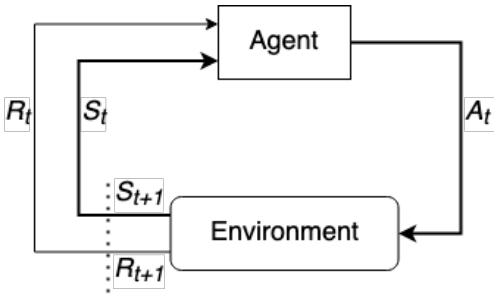

Figure 2.1: The reinforcement learning loop.

an expected reward given a single action and a specific state and the agent is

looking to maximize the reward over a sequence of actions.

$$\mathbf{R}_a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \tag{2.1}$$

For the agent to account for the difference in rewards from possible future actions and states a total reward function $\mathbf{R}_T$ is defined, where future expected rewards are accounted for at a discounted rate.

$$\mathbf{R}_T = \sum_{t=0}^{H} \gamma^t \mathbf{R}_a \tag{2.2}$$

Where γ is the discount rate for each time step and H is the time horizon the agent is considering, which can be infinite. With the total reward function in hand, there are two methods, model-free and model-based, for learning a policy to select the appropriate action.

2.1.1 Model-Based vs. Model-Free Reinforcement Learning

Model-based reinforcement learning (MB RL) is defined as any MDP method that contains a model of the transitions between states and that learns a global value function [15]. The model is typically used to predict the future state which then the learned global value function uses to determine a strategy for selecting actions. This contrasts with model-free reinforcement learning (MF RL) which learns entirely from experience what strategy to employ in action selection. If we look at the two in the case of robotics, MB RL can use the dynamics of the system to help inform future states and rewards, whereas MF RL is completely unaware of its embodiment and only uses past experiences to inform its decisions. The difference between the two is highlighted in figure 2.2

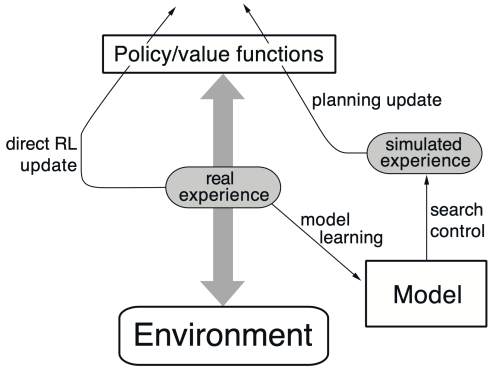

Figure 2.2: MF-RL and MB-RL pipelines [16].

where the left side of the image identifies how MF learns direct from experience, while model-based leverages a learned model to predict experiences before evaluating them in with the value function. This means that model-based

learning has the advantage when planning actions as it can use past experiences to inform new decisions whereas model-free can only draw upon actions it has taken before.

2.2 Deep Learning

Deep learning is another category of machine learning that focuses on how to use computers to model the brain such that they can learn to perform complex tasks. The primary technology used in this area is called an artificial neural network (ANN). An ANN is a collection of artificial neurons that mimics the structure of the brain by linking the signals from multiple neurons to others which then process the input to determine whether to fire or not. This interconnected structure allows for complex decisions with multiple inputs to be made based on the cumulative simple decisions. In practice, these ANNs are called multi-layer perceptions (MLPs) where they are formed by connecting many neurons in rows or layers one after each other with interconnection between each of the neurons in each of the adjoining layers. Each neuron can be represented by a linear combination (eq. 2.3) of the input signals $\mathbf{x}$ from the previous layer and the importance or weighting $\mathbf{w}$ of each of the signals for that neuron along with a bias b.

$$\mathbf{x}^T\mathbf{w} + b = \sum_{n=0}^{N}(x_n w_n) + b \tag{2.3}$$

To process the information the neuron applies an "activation function" $\phi(\mathbf{x}^T\mathbf{w})$ to establish a criteria for how the output will look and act. There are many possibilities for activation function, some linear and others non-linear, where the only requirement is that they must be differentiable. The function choice is tailored to the ANN's application, as some require categorical outputs or continuous. Some popular functions include the Sigmoid function (eq. 2.4), hyperbolic tangent (eq. 2.5) and ReLU (eq. 2.6). The Sigmoid function is typically used as an output activation function since it is bounded between [0,1], which is seen mainly when predicting probabilities. Similarly, the hyperbolic tangent function squeezes values between [-1, 1] and is often used to determine the importance of data. ReLU [17] by comparison is used mostly as a function between layers, as it is simple to calculate and has a grounding in nature as negative signals generally do not exist.

$$Sigmoid(x) = \frac{1}{e^{-x} + 1} \tag{2.4}$$

$$Tanh = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.5}$$

$$ReLU(x) = \max(0, x) \tag{2.6}$$

The collection of interconnected layer neurons is what constitutes feed-forward (FF) networks where the information is passed sequentially through and no information is saved or passed back to previous layers. FF networks also include convolutional neural networks, which are discussed later and are separate from recurrent neural networks, which contain connections to previous states and will also be discussed later.

2.2.1 Convolutional Neural Networks

A convolutional neural network is a type of neural network that retains spatial and temporal context between layers in contrast to the densely connected layers of an MLP. They achieve this by using filters with learned weights and performing a convolution (or cross-correlation) of the weights over the input values as in figure 2.3. A convolution is a mathematical operation between two functions that create a new function by continuously sliding a function over along an axis containing the second function and calculating the overlap for all values along the axis. The value of the output function is then a weighted representation of the values of a neighbourhood of the static function. This spatial context is used in networks to determine patterns and shapes in the input data. CNNs are used heavily in the processing of images given their structure of ordered pixels and colour channels and are extremely useful for reducing a large number of input values.

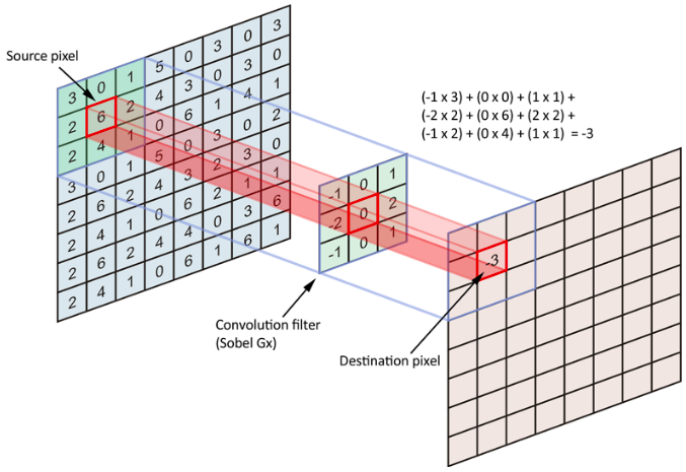

Figure 2.3: One filter in a convolutional neural network [18].

This is referred to as encoding when the input image is mapped from colour space to an abstract latent space that can then be used in future processing.

2.2.2 Recurrent Networks and LSTM

Recurrent Neural Networks (RNNs) are another class of neural networks that are used to process sequential data [19]. RNNs handle data by applying a shared set of weights to inputs, one after another, and storing the history in a hidden state vector **h**. The basic operation of a RNN can be seen in figure 2.4 where the shared set of weights $\mathbf{W}, \mathbf{U}, and \mathbf{V}$ operate on both the input data and the state vector for each time step. Using shared weights provides another benefit over network architecture in that the input can be of variable length, this is particularly useful in language translation [20], [21] where sentences are varying length, but also in robotics for action planning [22], [23] where the planning horizon may change depending on the task or scenario. While the structure of RNNs allows them to process the sequential data efficiently, they have an innate issue when training for long time horizons as the gradient used to update the weights tends to vanish or explode [24].

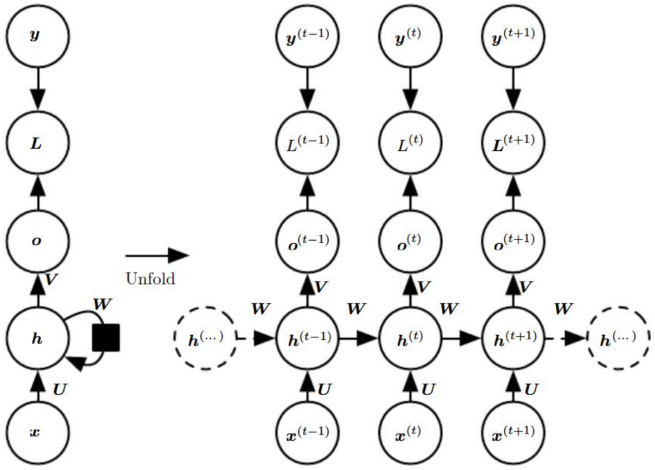

Figure 2.4: A recurrent neural network [19].

To deal with this problem the Long Short-Term Memory (LSTM) algorithm was created. LSTM solves the gradient problem by propagating constant error flow through its gated structure [25]. Introduced in [26], LSTM networks vary from vanilla RNNs in the way they process data. Instead of directly passing along a hidden state an LSTM network includes more operations to determine which information can be forgotten and which should be remembered. This is accomplished by introducing a memory buffer called the cell state, where information is added and subtracted by the various operations. These operations are defined by four gate layers: forget, input, memory and output as seen in figure 2.5.

The forget gate aims to reduce and remove old, unnecessary information and promote and remember the important information through the cell state. This is accomplished by applying a sigmoid activated dense layer to the con-

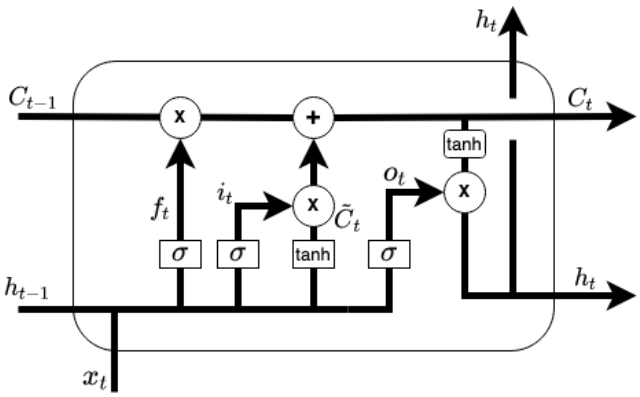

Figure 2.5: An LSTM network cell [27].

catenated previous hidden state and the new input. The result, found by equation 2.7 is the forget state f_t with a value between 0, forget entirely and 1, remember entirely.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.7}$$

Next is the input gate, which is used to determine which values will be updated in the cell state memory. This is an identical operation to the forget gate, with a dense layer with weights and bias W_i and b_i respectively produces i_t again with values between [0,1].

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.8}$$

Then to update the cell state a contender state is created at the memory gate using a hyperbolic tangent activation function. This gate layer processes the input and previous hidden state into a range similar to that of the cell state. The information to be added to the cell state is then determined by multiplying the input state, i_t which is the contender state. A new cell state C_t is then produced by multiplying the forget state with the previous cell state C_{t-1} and adding the new information (eq. 2.10).

$$\tilde{C}_t = Tanh(W_m[h_{t-1}, x_t] + b_m) \tag{2.9}$$

$$C_t = C_{t-1} * f_t + \tilde{C}_t * i_t \tag{2.10}$$

Finally, the output is determined using the output gate. This gate determines what information from the new cell state C_t needs to be returned. This is done using another dense sigmoid layer on the input and previous hidden state, producing o_t. This output state is then multiplied by a hyperbolic tangent squashed cell state to produce the output for the time step and the new hidden state h_t for the next step.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * Tanh(C_t) \tag{2.12}$$

3 Related Works

As mentioned earlier, the work presented in this thesis is building upon the work in [11] and [12]. This chapter will look at other previous works in the field and discuss the differences and similarities with this thesis. This chapter starts by looking into the use of high detail maps in navigation and some methods that use them. Next, purely vision-based navigation methods are discussed. Finally, crowd and multi-agent avoidance methods were examined.

3.1 Map vs Map-less Navigation

In recent years with autonomous vehicles becoming increasingly more viable, there has been a split in the community about whether or not to use high detail maps to assist in localization. The idea that a system could simultaneously localize and map (SLAM) its environment has been around since the 1980s [28] and began to be implemented in the late 1990s and early 2000s with [29], [30]. SLAM typically works by continuously taking sensor measurements from either a camera, LiDAR, or radar to detect "features" of the environment and store their positions relative to the vehicle, creating a sparse map. Then using conditional probability from Bayes' theorem and other tracking techniques, to predict the new location of the vehicle relative to the map following a selected action. If the measurements are of a high enough detail or the system has explored the environment enough, the challenge of path planning for navigation becomes significantly easier as collision-free paths can be calculated in real-time. These methods do however have their drawbacks, as they are typically computationally expensive and their effectiveness tends to degrade with large, complex maps [31]. Using this framework Dayoub et al. [32] have shown robust navigation in a complex indoor environment that can successfully avoid various obstacles. The system, however, was limited in speed due to the heavy computational cost requiring three onboard computers. Another limitation of SLAM methods is their reliance on the detection of features from the measurements. Soares et al. [33] proposed a SLAM method to deal with dynamic obstacles, specifically pedestrians. While their system showed promising results using a neural network to detect and track pedestrians, they cited that they still had limitations when pedestrians took up a large part of the image scene causing issues with static object feature detection. These limitations are commonly referenced in autonomous navigation and are seen by some as more of a problem than a solution. With this in mind, Chen et al. took a different approach to maps in [34] by using multiple small local maps to represent where human pedestrians are relative to the robot. This information was then

combined with a deep learning approach that successfully navigates around moving obstacles in open indoor environments. This method eliminated the need for feature detection and mapping of the environment however, it still requires the robot to know the position and velocity of each pedestrian's at all times, thus increasing in complexity when outside of structured environments. Similarly, Liu et al. [35] used occupancy grids and angular maps of obstacles and pedestrians to inform their RL policy, which relies on 3D LiDARs and stereo cameras. In comparison to the previous methods, Tai et al. [36] use a completely map-less approach using raw laser range finder (LRF) measurements and deep reinforcement learning to map directly to linear and angular velocity controls. This method was able to achieve robust indoor navigation using sparse measurements significantly reducing the computational cost and allowing for a higher more usable maximum velocity.

3.2 Vision-Based Navigation

There has been plenty of research done on vision-based navigation in recent years as many see the complexity and cost of using other sensors, such as LRFs or LiDAR, as unnecessary compared to the common and relatively cheap colour camera. Some of the more straightforward works such as [37], [38], navigation methods have been produced by mapping raw colour images directly to control inputs. These approaches use CNNs to encode colour images that are then processed by FF networks to output linear and angular velocities that can be used to avoid static obstacles. These methods, however, required data to be gathered using costly human demonstrations as they are trained using supervised data. Significant work was done by Mirowski et al. [39], where they achieve near-human-level performance in complex simulated environments using a deep learning approach with LSTMs. This method used the current image, previous reward, previous action, and the current evaluation of the value function to determine angular and linear velocities. In this model architecture, they use the current encoded image and the previous time step's rewards in the recursion of the LSTM to provide context for the next actions. While their method was thoroughly tested, the test environment was limited to simulated indoor spaces with no dynamic obstacles. Taking a different approach, Hirose et al. [40] proposed a vision-based deep learning navigation method that learns to model the output of a model predictive control (MPC); a type of controller that determines the optimal control sequences, given a set of constraints. This design varies from the two previous, as it does not directly map images to actions but predicts an intermediate value for traversability, that is then used in the optimization for action selection. Another technique that has been used with deep learning approaches has the model complete auxiliary tasks during training. Kulháneket et al. [41] presented a method that estimated depth and pixel change between concurrent images, where neither were used to determine

actions but were shown to improve the training of their vision model.

3.3 Crowd Collision Avoidance

Pedestrian modelling and collision avoidance behaviours have been the focus of study for many years. One of the more successful models was the Social Force Model (SFM) presented by Helbing et al. in [42]. Their works provided a pedestrian model that was based on the idea that there are unseen forces that act on people while walking. These 'social forces' are described as attractive forces pulling the pedestrian to their goal and repulsive forces pushing them away from static and dynamic obstacles. More recently there has been a standout among what is considered the state of the art for collision avoidance, that being optimal reciprocal collision avoidance (ORCA) [43]. This method defines a mathematical model for multi-body navigation and collision avoidance that calculates collision-free velocities for all agents, given their current position and velocity. ORCA is a popular choice for pedestrian simulation and is used to train other methods. Chen et al. used ORCA demonstration trajectories in [44], a deep reinforcement learning technique that performs as well or better than ORCA. In later work, Chen et al. [45] introduced a socially aware behaviour to their agent by introducing a reward penalty when an agent is within one of several specific areas, they had defined based on the type of movement between agents, such as crossing or overtaking.

4 Proposed Navigation Method

In this chapter, the details of the control policy and pipeline used to enable autonomous navigation are presented. The policy contains two modules: the deep learning model detailed in section 4.1 and the action planner described in section 4.2. The deep learning model is used to predict whether the outcome of taking an action will result in the robot being in an unsafe state or not. The action planner, serves as the value function of the system, its utilizes learned model's output along with other information, such as the position of the robot relative to a target location, to determine an optimal set of actions for a fixed number of future time steps. Section 4.3 describes the system's costs and rewards and how they are used when the action planner updates its optimal actions. This section also presents the criteria used to define safe and unsafe states. Finally, section 4.4 outlines the training process for the deep learning model.

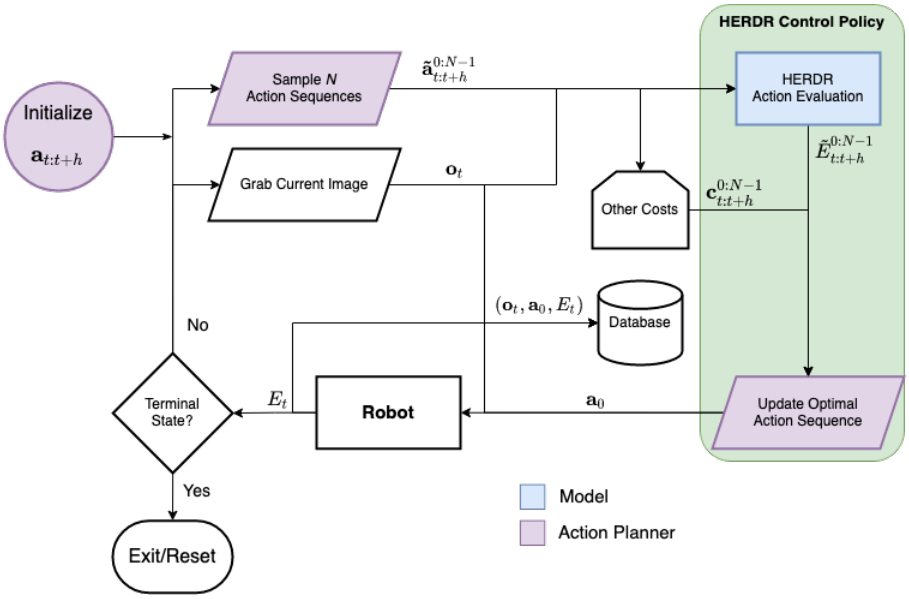

Figure 4.1: An overview of the HERDR framework.

Figure 4.1 provides an overview of the operation of the system. The process is first initialized with predefined optimal sequence of actions with a specified time horizon $\mathbf{a}_{t:t+h}$ and a sampling variance σ for each control variable. An image $\mathbf{o}_t$ from a forward-facing onboard camera is captured and N number of samples are drawn from a normal distribution around the optimal action sequence $\tilde{\mathbf{a}}_{t:t+h}^{0:N-1}$. The image and action sequences are then fed into the deep

learning model which produces estimates for the resulting state $\tilde{\mathbf{E}}_{t:t+h}^{0:N-1}$ of each action. The estimates, along with other costs $\mathbf{c}_{t:t+h}^{0:N-1}$, are then used to update the optimal action sequence. After the update, the first action of the optimal sequence is selected and executed by the robot. The system then records the true state following the action and stores it in a database along with the selected action and the image. If the robot has not collided or reached another terminal state, such as reaching its goal, then the cycle continues otherwise, the system exits or resets.

4.1 Deep Learning Model

The deep learning model is the heart of the control policy. The model predicts the probability of the robot being in an unsafe state based on an action sequence and what it detects in the current image. This process is a combination of two common deep learning problems, image and time series processing, which require different types of neural networks. This model combines both a CNN (Section 2.2.1) and an RNN (Section 2.2.2) specifically, an LSTM.

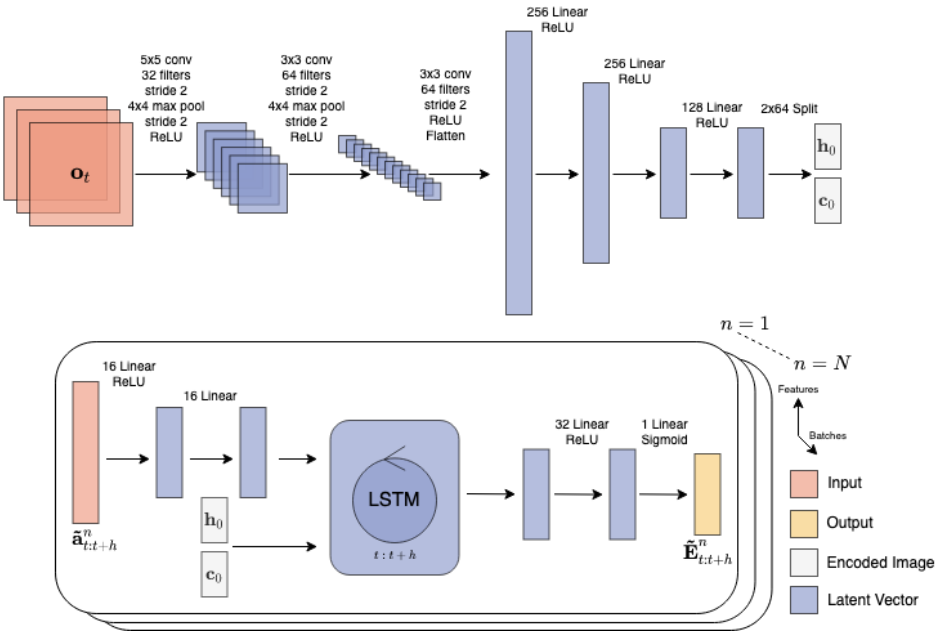

Figure 4.2: Detailed deep learning model architecture.

The neural network architecture used in this work is based on that used in [11], where figure 4.2 contains a diagram illustrating the structure. The model contains two inputs, a 3-channel colour image that has its values normalized between [-0.5, 0.5] (eq. 4.1) and a set of action sequences containing linear speeds and steering angles for each time step (more in section 4.2.2). The

model outputs a set of probabilities of the robot being in an unsafe state (detailed in section 4.3) for each step along the action sequence.

$$\mathbf{o}_t^{norm} = \frac{\mathbf{o}_t}{255} - 0.5 \tag{4.1}$$

The model first encodes the image using two convolutional and max-pooling layers followed by a final convolutional layer. After each max-pooling and the final convolution, a ReLU activation function is applied. The inclusion of max-pooling layers is common in CNN design as they reduce the number of features in the model. By reducing the number of features there is some loss in detail but this provides savings in computer memory and a reduction in computational cost.

The resulting data is flattened and processed by four linear layers further reducing the number of features, before being split in half to serve as the initial hidden $\mathbf{h}_0$ and cell states $\mathbf{c}_0$ of the LSTM. Using these states from a single image, a variable amount of action sequences can be processed. Before being fed into the LSTM, the sampled action sequences are expanded by a linear-ReLU activated layer and an unactivated linear layer. The LSTM then processes each sequence along the time axis and produces an output that is reduced with another linear-ReLU layer before producing a probability using a final linear-Sigmoid activated layer.

4.2 Action Planner

The action planner module carries out the sampling, updating and selecting of actions. The planner acts as the RL policy by using a continuously updated optimal sequence based on multiple exploratory samples to inform action selection. The module uses sample action sequences from a limited continuous action space to determine which actions are optimal given their likelihood of resulting in an unsafe state in addition to their task and operational costs.

4.2.1 Action Space

Similar to any other control or learning task, a range of permissible actions of the policy, often called the action space, must be defined. Acceptable actions are defined by a space that considers three constraints:

1. Dynamics of a non-holonomic car-like robot
2. Camera field of view
3. Walking speed of pedestrians

As this thesis was done in collaboration with Capra Robotics, a non-holonomic drive vehicle model was used to ensure compatibility with their Hircus platform. A non-holonomic drive vehicle is one where the number of degrees of

freedom (DoF) is more than the number of control inputs. In mathematical terms this means its current state is dependent on each of the previous states where the dynamics of the vehicle can be described by the following set of equations:

$$x_{k+1} = x_k + \nu_k \cos(\psi_k)\Delta t \tag{4.2}$$

$$y_{k+1} = y_k + \nu_k \sin(\psi_k)\Delta t \tag{4.3}$$

$$\psi_{k+1} = \psi_k + \frac{\nu_k}{l_f}\delta_k \Delta t \tag{4.4}$$

These equations define the motion of the vehicle using discrete-time approximations, where x and y are the 2D positions relative to an external origin, ψ is the global heading, ν is the linear speed, δ is the steering angle, l_f is the wheelbase length and Δt is the discrete time step interval. Using these equations along with the model in figure 4.3, the two control inputs, linear speed ν and the steering angle δ are used to control the robots 2D position and heading.

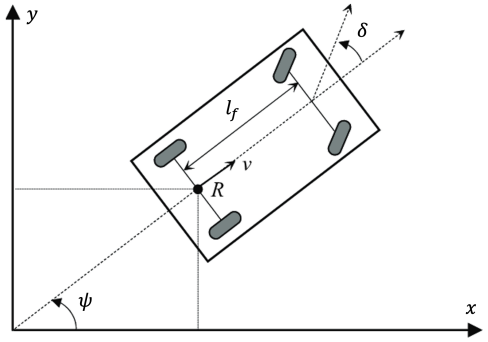

Figure 4.3: A car-like non-holonomic dynamical model [46].

This type of model defines a two-dimensional square action space, seen in figure 4.4, as the control inputs are independent of each other. The bounds for the square are determined by the remaining criteria. First, the field of view of the camera provides a limit on the amount the robot can turn in a single time step δ_{max}. This behaviour similarly presents itself in humans, where we generally first identify the destination we wish to achieve is safe, before advancing to that location, however having a fixed camera limits the viewable area. The robot is limited to doing the same by ensuring that any new position can be evaluated correctly based on, at minimum, partial observations from the image. Last, the maximum speed of the robot is restricted to reduce the risk for nearby pedestrians and improve controllability by limiting the distance travelled between action updates. Therefore, a maximum linear speed ν_{max} is set just above the typical pedestrian walking speed, to ensure safety and to still allow for overtaking when necessary. For this thesis, the assumption is

made that the proposed system is one of the multiple systems on the robot and that other safety stopping mechanisms are present that can override the controls in the case of an unexpected event. Therefore, the action space for this system does not have to include zero velocity. A bottom bound on linear speed, ν_{min}, can then be set to a small positive value as it could be unsafe to reverse blind and the system is only responsible for stopping upon reaching a terminal state.

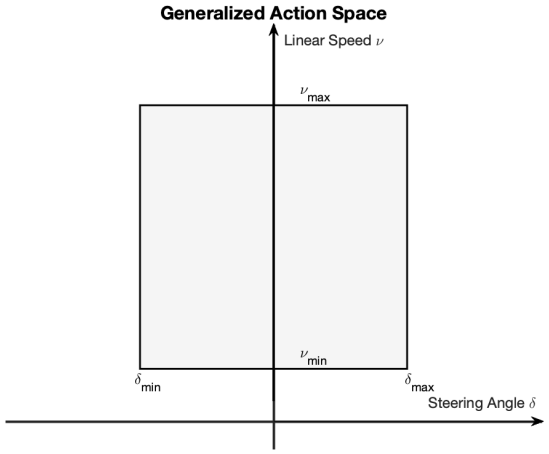

Figure 4.4: The generalized action space of HERDR.

4.2.2 Sampling and Updates

As mentioned before, the action planner uses a continuously updated optimal action sequence as the mean for a normal distribution. Between updates, samples are taken from this distribution and evaluated using the deep learning model. The algorithm used to update the optimal sequences is an adaptation of the one presented in [47] which combines ideas from model predictive control with Monte-Carlo approximation. The details of the algorithm are presented below in Algorithm 1.

Upon initialization of the planner, multiple parameters are set including the total number of samples N to be taken at each time step, a time horizon h for how many steps into the future to consider, and an initial optimal action sequence $(\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+h})$, sampling variances $\sigma_\nu, \sigma_\delta$ for each control input and finally two update parameters β and γ. The update parameter β is used to control the amount that consecutive actions are time-correlated, while γ is a weighting term for the reward of a sequence when updating the optimal. Using these values, at every time step while the task is not complete, the planner creates multiple action sequences $\tilde{\mathbf{a}}^n_{t:t+h}$ which are time-correlated to improve the smoothness of each sequence and are clamped to ensure they are within the defined action space $[\mathbf{a}_{min}, \mathbf{a}_{max}]$. During sample sequence construction,

the optimal sequence is shifted forward such that the action $\mathbf{a}_{t+1}$ is used as the mean for the first sample and $\mathbf{a}_t$ is only for time correlation. Similarly, at the end of the sequence, a new optimal action is added $\mathbf{a}_{init}$ that serves as a mean for the final time step distribution. This approach ensures that the actions are in line with the correct time step and the effects from updates are retained through time. The sampled sequences are then sent to the model to be evaluated and a corresponding reward R_t^n for each individual action $\tilde{\mathbf{a}}_t^n$ is returned. The rewards are then totalled for each sequence and used in an exponential weighting function to update the optimal sequence. Finally, the first updated optimal action $\mathbf{a}_0$ is sent to the robot.

4.3 Rewards and Experiences

Continuing along the pipeline from figure 4.1, the rewards, cost and experiences of the method will be discussed. The rewards that the method generates R_t^n

Algorithm 1 Model Predictive Path Integral Control (Modified)

n, N: Sample, Total number of samples;
t, h: Time step, Time horizon;
$\mathbf{a}_{init}$: Initialization action
$(\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{h-1})$: Initial optimal sequence;
$\sigma_\nu, \sigma_\delta$: Sampling variances;
$\boldsymbol{\epsilon}_t^n$: Sample noise;
β, γ, R: Update parameters;
while Goal not reached **do**
  for $n \leftarrow 0$ to $N-1$ **do**
    Sample $\boldsymbol{\epsilon}_{t:t+h}^n$ from $X \sim \mathcal{N}(0, [\sigma_\nu, \sigma_\delta] \cdot \mathbf{I}_h)$;
    for $\tau \leftarrow t$ to $t+h$ **do**
      $\tilde{\mathbf{a}}_\tau^n \leftarrow \beta \cdot (\mathbf{a}_\tau + \boldsymbol{\epsilon}_\tau^n) + (1-\beta) \cdot \mathbf{a}_{\tau-1}$;
      if τ is $t+h$ **then**
        $\tilde{\mathbf{a}}_\tau^n \leftarrow \beta \cdot (\mathbf{a}_{init} + \boldsymbol{\epsilon}_\tau^n) + (1-\beta) \cdot \mathbf{a}_{\tau-1}$;
      end if
      Clamp $\tilde{\mathbf{a}}_\tau^n$ to $[\mathbf{a}_{min}, \mathbf{a}_{min}]$;
    end for
  end for
  $R_{t:t+h}^{0:N-1} \leftarrow$ Evaluate $\tilde{\mathbf{a}}_{t:t+h}^{0:N-1}$ with model;

  $R^{0:N-1} = \sum_{\tau=t}^{t+h} R_\tau^{0:N-1}$

  $\mathbf{a}_{t:t+h} \leftarrow \frac{\sum_{n=0}^{N} exp(\gamma \cdot R^n) \cdot \tilde{\mathbf{a}}_{t:t+h}^n}{\sum_{n=0}^{N} exp(\gamma \cdot R^n)}$;
  Send $\mathbf{a}_t$ to Robot;
end while

are scalar values, that combines the predicted experience $\tilde{E}_{t:t+h}^{0:N-1}$ with the costs $\mathbf{c}_{t:t+h}^{0:N-1}$ for each action $\tilde{\mathbf{a}}_{t:t+h}^{0:N-1}$ based on equation 4.7. The costs are comprised of the euclidean distance to the goal position (eq 4.5) and the L2 norm penalty for the steering angle and relative velocity (eq. 4.6).

$$c_{dist} = \sqrt{(x_{robot} - x_{goal})^2 + (y_{robot} - y_{goal})^2} \tag{4.5}$$

$$c_{act} = \delta^2/2 + (\nu - \nu_{init})^2/2 \tag{4.6}$$

The inclusion of the euclidean distance to the goal causes the trajectories to have a bias towards the goal enabling the system to navigate to the desired location. The action cost is included to condition the system to make smaller adjustments smoothing control changes and reducing energy consumption. When added to the predicted state the two values are first normalized before being multiplied by the weighting vector $\mathbf{w}$. The weighting vector provides another tuning parameter for the system that allows for the behaviour to be changed. By increasing the weight for the goal cost the system becomes more goal-oriented and attempts to close the distance faster. Increasing the action cost, makes the system less sensitive as larger steering angles and speeds that deviate from the initial value become more costly.

$$R_{t:t+h}^{0:N-1} = -(\tilde{E}_{t:t+h}^{0:N-1} + \mathbf{w} \cdot \mathbf{c}_{t:t+h}^{0:N-1}) \tag{4.7}$$

The final piece of the pipeline is to determine what state the robot is in after taking an action. The proposed method uses a single binary state: safe or unsafe, where the difference can be defined by multiple criteria. For this work, with the focus on safe navigation around pedestrians, one of the criteria defines an unsafe space around pedestrians. Rather than using a traditional fixed radius from a pedestrian, the method defined an off-centre ellipse around the pedestrians to better model the social component of navigation among humans. This personal space model assumes people care more about the space in front of them while walking as opposed to behind them. This comes from the general bitterness people can have towards others when they feel they have been interrupted and made to adjust their trajectory because of the actions of another. The personal space model used is described by equation 4.8.

$$\frac{((y-h)\cos(\theta) + (x-k)\sin(\theta))^2}{a^2} + \frac{((y-h)\sin(\theta) - (x-k)\cos(\theta))^2}{b^2} < 1^2 \tag{4.8}$$

$$h = -s \cdot \sin(\theta), k = s \cdot \cos(\theta) \tag{4.9}$$

To use this equation, the global position $[x, y]$ and heading θ of all pedestrians are assumed to be known. This model creates an ellipse with a radius of a perpendicular to the heading direction and a radius of b parallel to the heading direction where $a < b$. The ellipse is shifted in the direction of travel

by s metres, relative to the pedestrian, leaving a smaller area behind and extending the area in front, similar to the shape for personal space found in [48]. A visual example of a safe space can be seen in figure 4.5. For this work, the parameters used to define this space are shown in table 4.1.

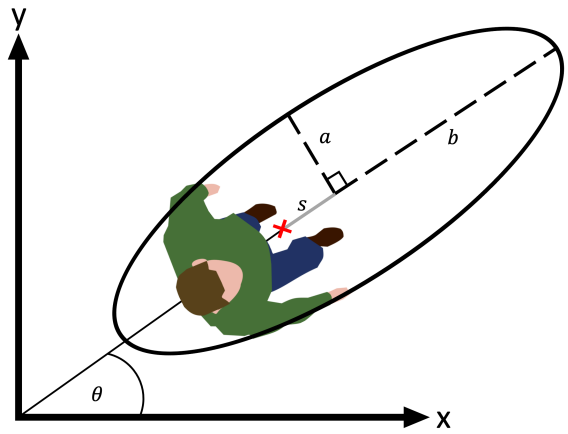

Figure 4.5: Generalized pedestrian safe space model.

a	0.8
b	1.5
s	1
Distance Behind	0.5
Distance In-Front	2.5

Table 4.1: Parameters for pedestrian safe space model, all values are in meters.

Additional criteria for unsafe conditions include, collisions and driving onto the road. Collision can be determined autonomously through measurements from an onboard accelerometer reaching a threshold or with other sensors and triggers like a bumper bar. The robot driving onto the road is measure through its angular velocity. The introduction of excess tilting as an unsafe state is novel to the application in this thesis, as navigating in urban environments the robot is likely to be on a raised curb relative to the road. The change in body angle caused by dropping a wheel off the curb can be measured by an onboard gyroscope and used to label unsafe states.

4.4 Training

While the method is designed to operate onboard a mobile robot, the system is trained offline on a larger, more powerful computer. To perform training a dataset containing numerous labelled images, actions and states are required. Given the unique control policy, there is no available dataset that can be used

for training the network, therefore data must be collected. The system is designed to collect data in a self-supervised manner, such that while the robot is driving it can always be collecting and labelling its own data. The data that is collected is considered to be off-policy such that any control policy, using the defined action space, can generate usable data for training, thus the system can be continually improving with increasingly more experience. Since the network is trained offline, data can be collected over multiple sessions and models are not required to be trained from scratch. During training, the problem is reduced to a classification task where the state acts as a binary class [0,1] – 0 indicating safe and 1 indicating unsafe. The network is trained with the binary cross-entropy loss function seen in equation 4.10)

$$l(x, y) = \sum_{n=0}^{N-1} -w_n[y_n \cdot logx_n + (1 - y_n) \cdot log(1 - x_n)] \tag{4.10}$$

This function is commonly used in binary classification training where x_n is the output of the network, in this case, a single probability and y_n is the labelled state. This function also includes a positive weighting term w_n to help with unbalanced datasets. This is useful for this project, as by design the system is expected to be in safe states significantly more often than unsafe ones. Therefore, the tuning of this parameter plays an important role in the ability to train the system. The value of the weighting term can be found by determining the ratio of safe to unsafe states, which could vary depending on the amount of data collected.

During training, the progress is monitored through several evaluation metrics that were recorded during each step as well as their averages after each epoch. The simplest measures to evaluate are classification accuracy and loss. These are necessary to verify if the network is still training or has over-fit. However, due to the large imbalance in the dataset, the overall classification accuracy could provide a false representation of the network's performance. As a result the network could be learning to classify all outputs to the more common label, while still producing high classification accuracy. To account for this, the accuracy of the less common label, or in this case, the accuracy for predicting unsafe states, is also recorded to confirm balanced learning. To validate the model is learning, it is possible to split the dataset and use a portion of data as a validation set which is not used in training. However, the performance of the model as a classifier is not directly correlated with the navigation and collision avoidance performance which is ultimately the desired results. As such, a validation set is not required and the complete dataset is used for training.

5 Implementation and Results

This chapter presents the results of the navigation system. The chapter is broken into three sections, each describing the test environment and scenario, followed by numeric results and discussion. In section 5.1, a simple low-realism simulator was used to verify the model's capability to learn the concept of a pedestrian's personal space when navigating in a crowded environment. In section 5.2 the system is deployed and trained in a highly realistic simulator where it was thoroughly tested in a populated urban environment. Finally, in section 5.3, the system was deployed in the real world using the top performing simulator model.

5.1 Crowded Personal Space

As this is a new application of the BADGR framework, it was necessary to first verify that the deep learning model could learn to avoid dynamic obstacles. To investigate this a simple simulation was created in the open-source simulator Webots [49]. This simulator was being used at Capra and a model for their Hircus robot and base controller written in Python had been made available. The simulation environment was kept extremely simple consisting of a Hircus robot, equipped with a single RGB camera and multiple stationary and walking pedestrians in a 10-meter radius circular arena as seen in figure 5.1. The pedestrian models and their controller were imported from an available Webots example to save on development time. The controller for the pedestrians was designed such that the pedestrians can only walk along a predefined path and are unable to correct their path when approaching an obstacle. This limited the effectiveness of the simulation and is discussed more later.

In efforts to keep the scenario simple, the number of pedestrians was set at six, where they were placed between the robot and the goal. The pedestrian's movement was also limited to either standing still or walking back and forth in a straight line. The task for the robot was then to navigate through the small crowd to the other side of the circle. A controller for the Hircus robot was created using the framework described in chapter 4 with the planner parameters seen in table 5.1.

The rationale behind these parameters was a result of computer hardware limitations and initial observed navigation performance. The number of sample trajectories had a significant effect on the method's execution speed and its navigation performance. Therefore a value of 50 was determined to balance these factors. For this simulation, the pedestrians walked at a speed of 1 m/s and the robot was given a maximum velocity of 1.2 m/s to allow for overtaking,

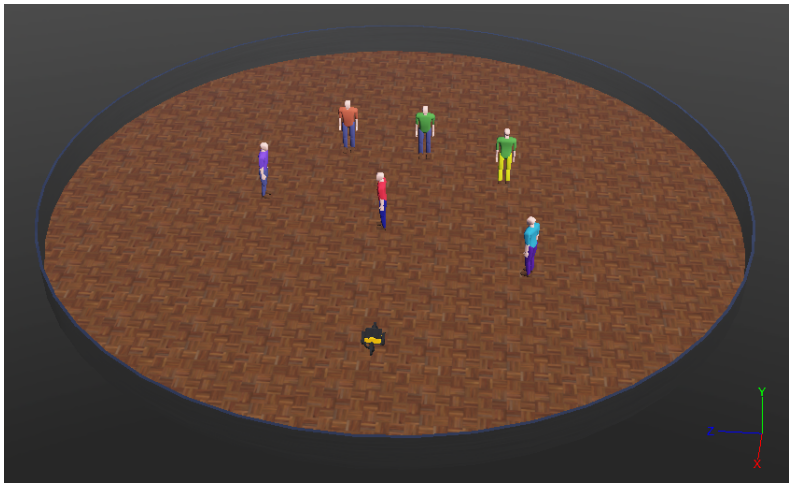

Figure 5.1: The webots simulation environment.

Sample Trajectories: N	50
Time Horizon Steps: h	10
Control Frequency: f_c	10 [Hz]
Update Parameters: β, γ	0.6, 50
Action Cost: c_{act}	0
Goal Cost: c_{dist}	0.2
Initial Velocity: ν_{init}	0.7 [m/s]
Velocity Sample Variance: σ_ν	0.5
Steering Sample Variance: σ_δ	1

Table 5.1: Webots action planner parameters.

if necessary. The time horizon and control frequency is a result of the limits of the simulator, as the controller operated synchronously and the simulator's physics would become inconsistent with time steps larger than 100 ms. With the time step having to remain low, actions were evaluated for 1 second into the future as predicting for more than 10 steps would increase the difficulty of the learning problem.

To train the model, data was collected from the scenario using an expert controller. This controller exploited the known states of the pedestrians and used an available object recognition algorithm to determine which pedestrians are within the camera's field of view. With the pedestrians identified and their states known, the expert controller would propagate the state of the robot forward based on all the sampled trajectories. The controller would then examine whether the robot would be within the personal space of the pedestrians for each of the time steps. The states for all trajectories were saved in addition to the current camera frame. This data collection method varies from that used in [11]; however, data is collected significantly faster, as

every action along all trajectories produces a ground truth value, as opposed to just the executed action. This provides a notable difference when training, as the model will be able to learn from outcomes of different actions for a given image. In addition, pedestrian orientation is required when determining the location of its personal space. In total, 1.05 million action-state pairs and 4,198 images were collected in about 1.15 hours of simulation time.

To evaluate the model's ability to learn pedestrian spaces and avoid moving obstacles, the method was compared against two other agents. The first uses an untrained model and the other uses the state-of-the-art method, optimal reciprocal collision avoidance (ORCA) [43], introduced in section 3.3. It should be noted that the ORCA method expects all agents in a system to be controlled using the same method. However, due to the implementation of the pedestrian controllers, the method had to be modified to ensure its compatibility, and thus its collision rate was not zero as defined by its ruleset. To provide a meaningful comparison of the three agents, commonly reported metrics for navigation systems in human environments were recorded during testing and are presented in table 5.2. Each of the methods performed 50 trial runs where the pedestrians' positions and orientations were randomized within a 4 x 4 (m) square area at the center of the arena.

	HERDR	Untrained	ORCA
Avg. Minimum Distance to a Pedestrian (m)	**2.28**	1.97	1.98
Collision Rate (%)	**0.47**	0.77	0.53
Time in Personal Space (ms)	**227**	265	312
Distance Travelled (m)	11.83	**8.27**	9.58
Distance to Goal (m)	8.14	8.14	8.14

Table 5.2: Performance comparison of HERDR and ORCA. Bold values indicate the best performance.

Based on the results in the table above, it can be seen that the HERDR method performs better than the ORCA method in nearly all the metrics. The results indicate that the method did learn a representation of personal space as able to maintain a greater distance to pedestrians and reduced time spent in their personal space. This is confirmed when compared to the untrained model which navigates directly to the goal. The extra distance travelled can be attributed to extra space needed to avoid the pedestrians' personal space. In addition to the numerical results, figure 5.2 shows a path taken by a HERDR

agent where it takes a wide route in front of the stationary pedestrian. This behaviour complies with the described model for pedestrian personal space. This image also shows the agent avoiding a moving pedestrian, whose paths are indicated by the blue dotted lines beginning in the center of the arena and ending off-screen.

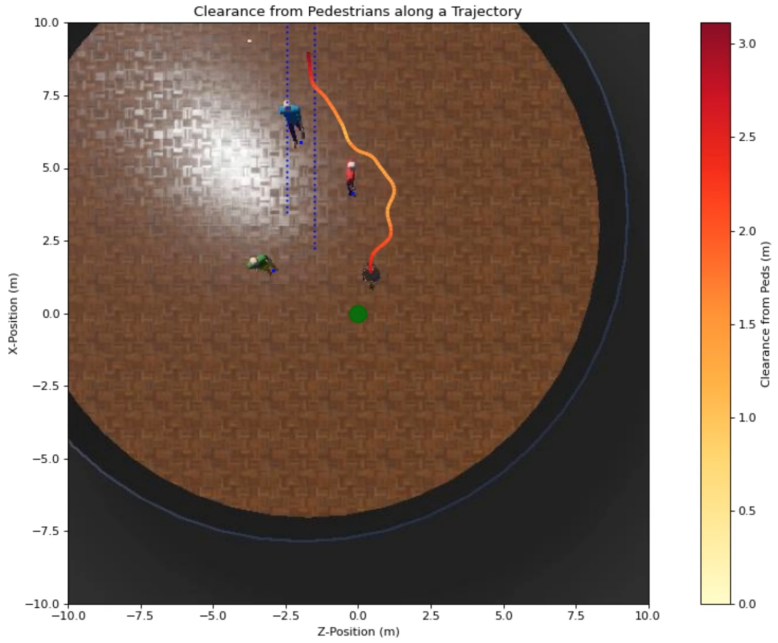

Figure 5.2: A trajectory showing desired behaviour.

Following these results, it was determined that the model was able to learn to avoid pedestrians while considering their personal space. When observing various trial runs, it was evident that the method actively attempts to avoid pedestrians and that in many cases it was the limited pedestrian behaviour collisions. An example of this shown sequence in figure 5.3. This observation indicated that a more realistic simulation environment, with more reactive pedestrians, was needed to properly evaluate the method.

5.2 High-Realism Simulation

Switching simulators allowed for the opportunity to both improve the behaviour of the pedestrians and the realism of the scenario. With both of these criteria in mind, a suggestion was made by Capra to use the open-source autonomous driving simulator CARLA [50]. This simulator is mainly used for autonomous road vehicle research; however, it provides high-quality renderings of complex urban environments and rule-based, AI-controlled pedestrians. In addition to high-quality graphical models the pedestrians exhibit human-like

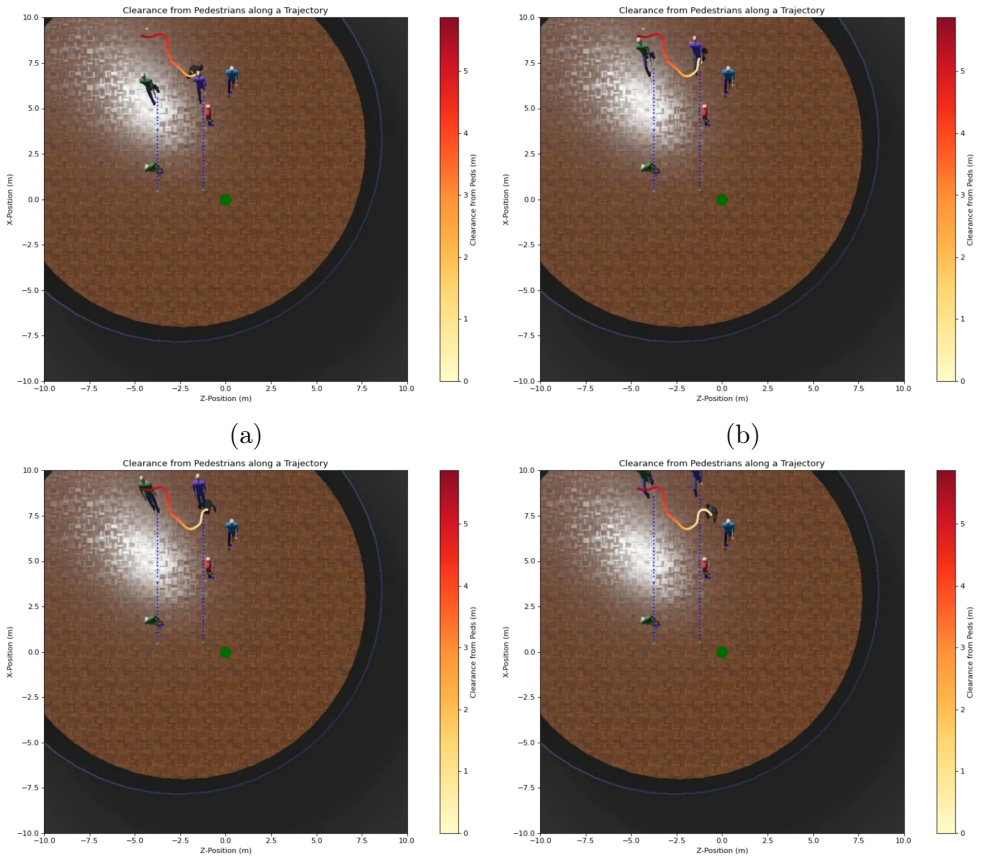

Figure 5.3: Example of a pedestrian not reacting to the evading robot.

behaviours when interacting with each other and the environment. For this work, the included environment - Town10HD - was chosen for its dense urban appearance with varied street designs from wide boulevards and office towers to narrow residential streets with cafes and row houses. The city, seen in figure 5.4, was divided into three sections, with the outer ring used for training and two of the inner blocks for testing. The outer ring was chosen for training as it provides the largest traversable area and contains a variety of environments from leisure areas, to an industrial area, and modern high-rise towers. The two inner blocks were selected to have one testing area with a similar aesthetic to the outer block, and the other be more distinct.

Initially, the model and parameters from the Webots simulation were tested in CARLA to evaluate their performance given significantly different visual input. However, when deployed in the CARLA simulator, the model performed poorly with an example output from the model shown in figure 5.5. In the image, the lines represent the sampled action sequences with the colour corresponding to the predicted unsafe state probability. The figure shows that as

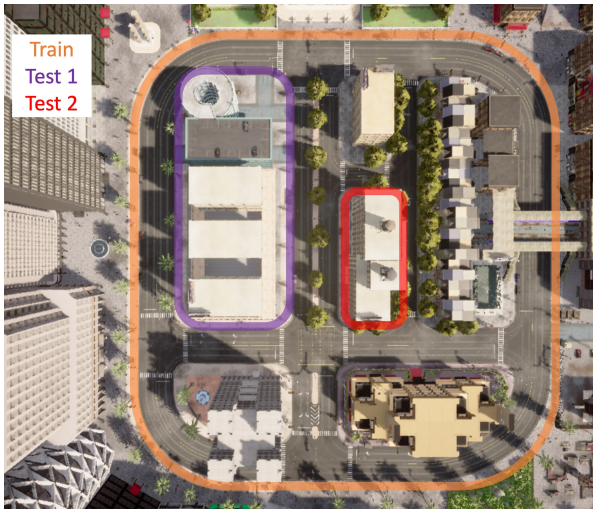

Figure 5.4: The urban environment where the different areas are highlighted.

the agent approaches a wall, it is unable to correctly label any states as unsafe, with all predicted probabilities of near zero. However, this was expected as the model was trained on significantly less complex images. As a result it is unable to distinguish hazards when presented with more complex scenes. From this outcome, the decision was made to train two models using new data: a freshly trained model using only CARLA data and a fine-tuned model using the Webots model as a base that is trained further on the new data.

Figure 5.5: The estimated states overlay on the input image.

The new simulation introduced a variety of new static obstacles for the system to navigate around. Unlike the pedestrians, the static obstacles such as light poles, garbage cans and bus shelters could not have their locations and geometries directly determined from the simulation. Therefore, the ground truth data was collected in a more traditional RL manner, by relying on ex-

periences such as collisions and excess vehicle tilt to determine unsafe states. For this scenario, both collision and excess tilt were used as terminal states where the simulation reset if these conditions were met. In addition to static obstacles, the positions and orientations of the pedestrians were exploited to determine when the agent was within their personal space, which was labelled as an unsafe state. However, these states were not considered terminal and the robot was allowed to continue after entering a personal space. The reasoning for considering these states non-terminal is that it may be possible that the agent is required to be close to pedestrians to progress. To facilitate data collection, a list of starting points on the outer ring of the city was created such that when a new trial began, an agent was placed at a random point from the list with a random orientation. The agent would then attempt to navigate to another random point selected from the list. Before beginning, around 200 pedestrians would also be added to the sidewalk and instructed to walk to random points along the outer ring. For each trial, the weather and time of day were selected from a portion of the preset list available within the simulator. Nighttime conditions were eliminated to reduce learning issues due to poor lighting. The conditions varied from the morning, noon, and sunset with weather conditions of varying levels of rain, clouds, and sunshine. Using this setup, an agent controlled by the Webots model collected data for around 20.5 hours of simulation time resulting in $363,000$ states and images. Unlike the Webots data, where whole action sequences were saved, the CARLA data only contained one action-state pair per image. During training this required that action sequences be created on a rolling basis. This was achieved using a similar method to [12], where action sequences were created by recalling the action-state pair along with the pairs that follow for the length of the time horizon. When the horizon extends past a terminal-unsafe state, the remaining number of actions-state pairs are generated with a small amount of noise added to the final action with all their states labelled as unsafe. This allows for the experiences from the terminal states to be propagated through to the beginning of the action sequence, similar to the rest of the data.

During training and testing, a bicycle equipped with a forward-facing camera was controlled using the HERDR framework and the action planner parameters were set according to table 5.3. In this scenario, the pedestrians walk at a speed of 1.4 m/s, which meant the initial velocity had to be increased to 1.5 m/s. The control frequency was reduced from 10 to 5 Hz while maintaining a time horizon of 10, allowing for planning 2 seconds into the future. The update parameter γ and the action and goal cost were set to 50, 0.25 and 0.25 respectively. These values were later varied to investigate their effects on performance, which is presented and discussed in section 5.2.2.

Following the training of the models on the newly collected data, both were tested on the two inner blocks from figure 5.4. The models were tested in two separate ways to evaluate both, their navigation ability, and their avoidance ability. Navigation performance was evaluated in a similar scenario to how

Sample Trajectories: N	100
Time Horizon Steps: h	10
Control Frequency: f_c	5 [Hz]
Update Parameters: β, γ	0.6, (20,50,80
Action Cost: c_{act}	(0.25,0.50,0.75)
Goal Cost: c_{dist}	(0.25,0.50,0.75)
Initial Velocity: ν_{init}	1.5 [m/s]
Velocity Sample Variance: σ_ν	0.3
Steering Sample Variance: σ_δ	1.5

Table 5.3: CARLA action planner parameters.

data was collected, by travelling between a limited set of points around the city block. In this case, only 50 pedestrians were added to the blocks to keep the pedestrian density closer to that of the outer ring when recording data. Next, the model's avoidance ability is measured by traversing structured crowded areas. Here, the planner parameters are varied to measure the impacts on avoidance. Finally, the performance of the models is compared against an ORCA agent.

5.2.1 Navigation Performance

The overall navigation performance results can be seen in table 5.4. The two models were each tested 60 times on both of the test block, totalling 240 trials.

	Block 1		Block 2	
	CARLA Only	Webots Pre-trained	CARLA Only	Webots Pre-trained
Avg. Min. Distance to a Pedestrian (m)	7.12	**7.52**	**3.69**	3.58
Success Rate (%)	24	**32**	**8**	7
Relative Time in Personal Space (%)	**4.0**	4.2	15.2	**9.3**
Distance Travelled (m)	37.3	**75.9**	11.5	**12.2**
Distance to Goal (m)	80.5	81.3	43.0	42.7

Table 5.4: Navigation performance results.

Upon review, the results indicate that the method navigates poorly with both models reaching the goal less than 35% of the time. However, when fur-

ther analyzing the success rate, two trends emerge indicating that the addition of a global waypoint planner would improve the results. In table 5.5 it can be seen that for both models the agent was 3 times more likely to reach the goal when the distance was less than 75 meters. This distance represents the Manhattan distance instead of Euclidean distance as is a more accurate representation of the minimal traversable path. On the given city blocks, 75 meters was roughly the point when the agent would be required to navigate around at least one corner. Thus it is conceivable that these further waypoints could be considered unreachable given the planner's only directional input is determined from the straight line distance. Similarly, the method struggles to reach the goal when obstacles in the path form a discontinuous boundary. This was a common point of failure that is best seen at the top of test block one (figure 5.4), where a circular car park forms an acute angle with the neighbouring building. When the agent's direct line to the goal falls within this angle, the agent almost always fails as it is unable to escape the closing walls. This is due to the model evaluating states on both sides of the agent to be unsafe and is unable to take corrective action. This indicates the method would likely perform better when goal locations are closer to being in the line of sight or when having intermediate waypoints at the apex of corners and away from dead ends. Another indication that closer waypoints would be beneficial was observed when the agent is facing directly away from the goal when still a significant distance away. Upon reviewing trials it was found that approximately 10% of time, the agent would enter a state facing away from the goal, causing it to struggle to determine whether to turn left or right. In this state it was unable to correct its heading before crashing or driving onto the road. This issue would also be addressed with the inclusion of a global planner that determines a closer waypoint that is not directly behind the agent, eliminating the directional uncertainty.

Manhattan Distance (m)	Trials	Success	Success Rate (%)
≥ 75	30	2	6.67
< 75	90	20	22.2

Table 5.5: Success rate with different goal distances.

Furthermore, the distance travelled in successful versus unsuccessful trials provides another area for improvement. Looking at figure 5.6, it can be seen that the method was more likely to reach its goal when it travelled farther than the first 25 meters. This is partly due to poor quality starting location, as they were selected to ensure the agent could begin without immediate collision. However, the agent's randomized orientation was not thoroughly considered. This is evident in figure 5.7 where the spawn locations, seen in red, are plotted with a heat-map of the terminal state locations where the lighter shades indicate more frequent terminal states. This result is most clear in the bottom right corner of block one, where three spawn points are located

within 10 meters of the most frequently recorded terminal location.

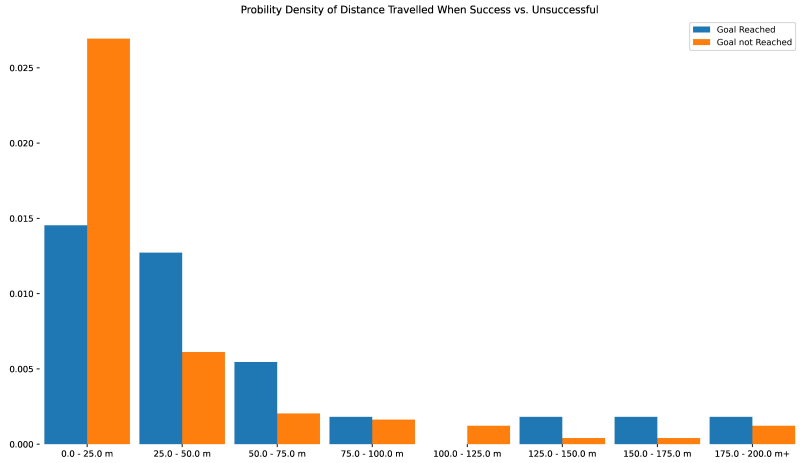

Figure 5.6: A probability density histogram of the distance travelled when successful or not.

Following this test, it was discovered that the pedestrians were unable to be collided with, when controlled by the available ruled-based controller. It was observed that when the agent moves within 3 meters of a pedestrian, the simulation automatically moves the pedestrian to avoid collision. This means that the training data likely does not have adequate information for the model to learn a representation of a pedestrian's personal space as it only extends a maximum of 2.5 meters. This provides an explanation for the difference in time spent in personal space on the narrower block 2, where the pre-trained model significantly outperforms the CARLA-only model. Another factor in the difference in performance is how the CARLA-only model struggled to differentiate between hazards on the sides of the image. It was observed that this model often predicted the same probabilities for each of the time steps in all sampled sequences. This produces poor performance as the planner is unable to make corrective updates to the optimal sequences when approaching hazards. By comparison, the pre-trained model was significantly better at predicting these hazards. This is likely due to the method of data collection as the pre-trained model had substantial training on a dataset containing multiple action sequences for each image. Whereas the CARLA dataset only contains one sequence per image, limiting its ability to learn to distinguish the difference between action sequences.

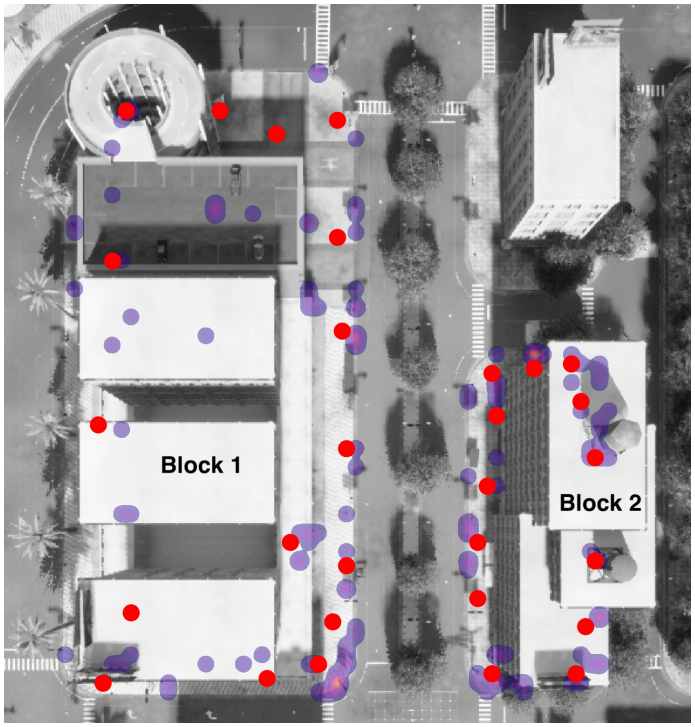

Figure 5.7: Terminal state location heat-map with test starting locations.

5.2.2 Pedestrian Avoidance Performance

The HERDR method's pedestrian avoidance performance was analyzed in two ways. First, the effects of the planner parameters were investigated using three key metrics: relative time in personal space, average minimum distance to a pedestrian and the magnitude of the steering commands per second. These metrics quantify the policy's reaction when encountering pedestrians. This provides insight into how to tune the parameters to maximize certain behaviour. Finally, the two models are compared to an agent using the ORCA method to examine their performance in a more realistic setting.

To investigate the effects of the planner parameters a test scenario was created in an open area of the map, where a HERDR agent would start along with multiple pedestrians controlled by the ORCA method. In this scenario, the agent is placed on a line next to two pedestrians while two additional pedestrians are placed on a parallel line 5 meters away. The agent and pedestrians are then each tasked to move to a random point on the line opposite to them, by passing through the on-coming traffic. An example of this scenario can be seen in figure 5.8. The three parameters chosen for testing, as stated previously in table 5.3, influence the updated actions in two different ways. The action and goal gain are applied to each of the actions in a sampled trajectory, accounting for a portion of the trajectory's reward. This differs from the γ

update parameter, as it changes the range of rewards that will have significant weighting when updating the optimal trajectory. The test was completed by running trials with all the combinations of each of the three parameters at three different values. Each combination was repeated 20 times in varying weather conditions and varying agent and pedestrian locations. The result of the test is shown in figure 5.9. Starting with goal gain, it is shown that a lower value provides a significant increase in the average minimum distance to a pedestrian; however, this was not a result of the desired behaviour. When reviewing recordings of the trials, it was observed that the agent could be easily directed away from the goal when a pedestrian was in proximity to the goal or due to a false unsafe state prediction. This often caused the agent to make a large loop away from the test area increasing the average separation. This behaviour can also explain the lower actions per second as the large loops are performed over a long time with small steering angles. A smaller goal gain did reduce the time spent in pedestrian personal space as its directional influence was not enough to outweigh the unsafe state predictions. The action gain and γ results were both observed to have effects on the magnitude of actions with lower γ values and higher action gain values producing small steering angles. This is in line with their design as the action cost increases the cost of larger magnitude action, reducing their weighting in the optimal sequence update. Whereas a lower γ can filter out more sporadic behaviour by widening the range of sequences that have a significant weighting. Despite the two parameters having an inverse relation for the magnitude of actions, it was found they have a proportional effect on minimum separation distance, with both increasing separation distance as their values increase. As γ increases, it can be expected that the actions could lead to more desired behaviour by reacting more to the pedestrians, as the planner becomes increasingly biased to the highest reward sequences, which are likely to be trajectories with low unsafe state probabilities.

The second test of pedestrian avoidance was to compare with the state-of-the-art ORCA method. Like the previous test, a HERDR or ORCA agent was placed in an open area with varying numbers of ORCA-controlled pedestrians, in a 5 x 5-meter square area. The agent and pedestrians were tasked with navigating the crowded environment and reaching a point on the opposite side. The test is first conducted with a single pedestrian located directly across from the agent. The test was then completed with groups of 2,4 and 6 pedestrians, where half of the pedestrians were placed on a line perpendicular to the agent forming an "L" shape as seen in figure 5.10. This forces them to cross the area perpendicular to the agent's direction of travel providing a different obstacle for the agent. The test was repeated 30 times for each group of pedestrians with pedestrians crossing from the left and right equally, totalling 120 test runs. For both models, the planner parameters were kept the same using, 0.50, 0.50 and 50 for action cost, goal cost and γ respectively. The results for both the trained models and ORCA agent are shown below in table 5.6. From the table,

Figure 5.8: Parameter evaluation starting position.

contrary to the simpler simulation, the HERDR models performed worse than ORCA in nearly every metric. Looking closer, it was found that the pre-trained model performed significantly better in keeping distance from the pedestrians, even as the space became more crowded. This is a result of the model having learned to predict unsafe states more effectively and being able to actively identify the appropriate actions to move away from the pedestrians. This is not the case for the CARLA-only model as it more often predicts uniform probabilities across all sequences at each time step. However, the behaviour of the pre-trained model does not appear to have translated into less time in the pedestrians' personal space. This is a result of it being possible for the agent to start within a pedestrian's personal space.Thus the time it takes the agent to accelerate causes an inflated percentage of time, in particular, if the agent collided within a short time. After examining the data, trials consisting of the agent in person space for more than 80% of the run time in addition to the run time being 5 seconds or less were removed as outliers. Then the data shows that the pre-trained model spent comparatively less time in personal spaces in tests with 2 and 6 pedestrians, spending only 6.61% and 17.41% respectively. When looking at the collision rate it should be noted that to use the ORCA method to control the pedestrians, the agent's state must be included for each control update step. This meant that a radius around the agent had to be specified in which the pedestrians would avoid. For the purpose of this test, the radius was set to 0.5 meters which is less than the length of the bicycle. The smaller radius was used as the ORCA pedestrians would be guaranteed to not collide with the agent if it were set to at least the length of the bicycle. This is the reason why the ORCA agent does not have a zero-collision rate as intended in its design. It was expected that the ORCA agent would collide less than the other agent and that for all agents, some of the collisions would be caused by the other pedestrians. This means that the number of collisions

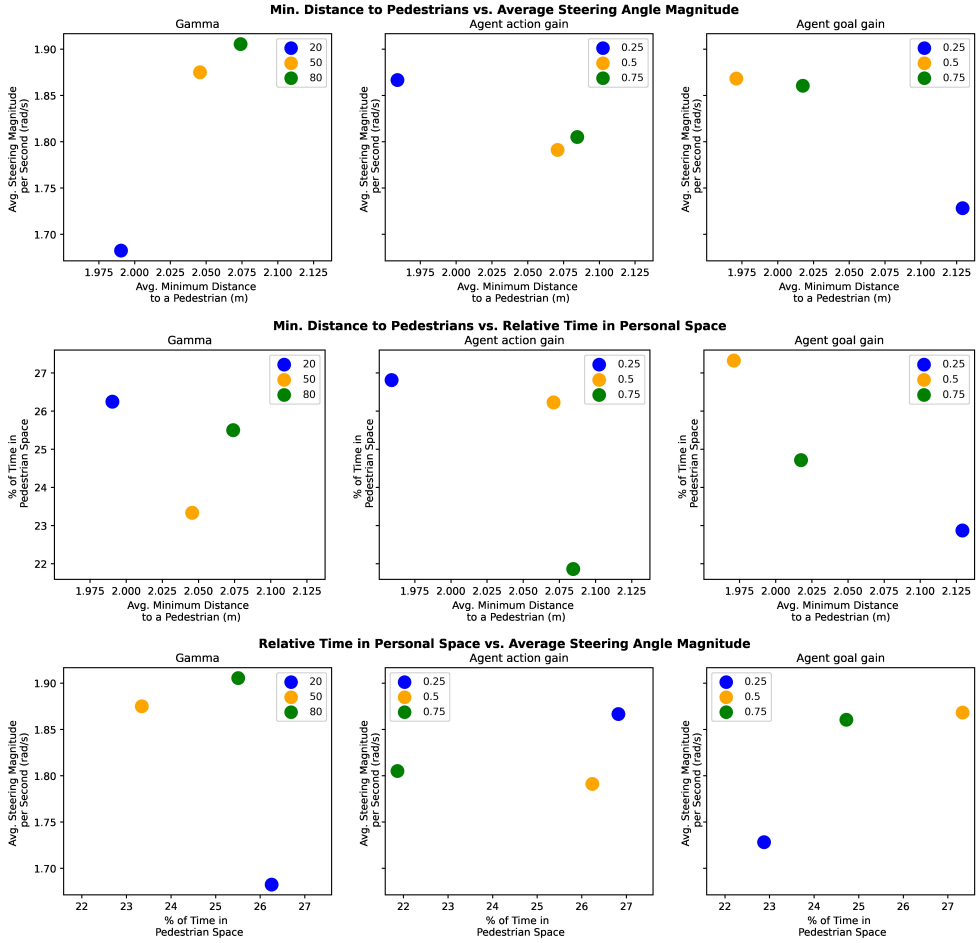

Figure 5.9: A comparison of action planner parameters.

that are the fault of the agents is likely lower.

5.3 Simulation to Real Life

The ultimate goal with any robotics project is to validate that the proposed system works in the real world. This required migrating from the simulation environment and onto a real platform. This was achieved by integrating with Capra's Hircus robot. The Capra Hircus system is built on a multiple docker container [51] ecosystem, with each running its own ROS [52] node to facilitate communication between hardware and software. This required a version of the HERDR framework to be built as a ROS package that can be executed within a docker container for it to be deployed on the robot. To achieve this, a new docker container was created that launches the HERDR system with one of the trained models using ROS publishers and subscribers to retrieve images and

Figure 5.10: Example testing scenario with 6 pedestrians in an L shape.

send commands to the robot. The system also outputs an image containing an overlay of the output of the model as seen in figure 5.11. Once built and configured, the container was able to be run off-line on a separate computer receiving images and sending commands via a wireless connection.

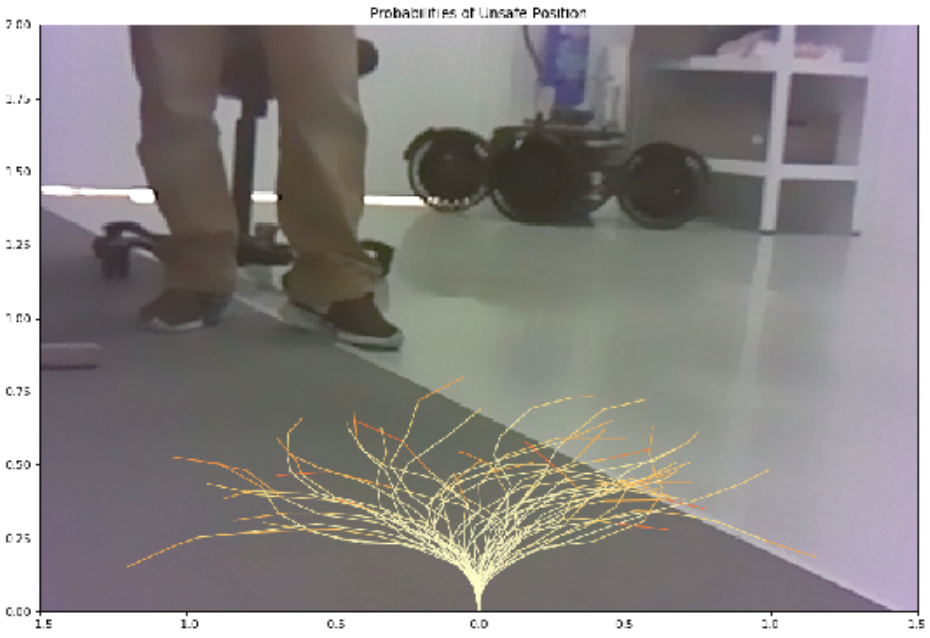

Figure 5.11: Sample output from deployment on a Capra Hircus robot.

The system was set to publish updated steering and velocity commands at a rate of 10 Hz, while the model and planner update the values of the

# of Pedestrians	Domain 5 x 5 (m)	1	2	4	6
Distance Travelled (m)	HERDR CARLA-only	5.42	4.92	4.71	4.63
	HERDR Pre-trained	6.31	9.84	9.68	7.66
	ORCA	**5.03**	**4.67**	**4.23**	**3.81**
Avg. Minimum Distance to a Pedestrian (m)	HERDR CARLA-only	4.07	2.92	2.57	2.44
	HERDR Pre-trained	**4.27**	**3.81**	**3.13**	**3.08**
	ORCA	4.18	3.17	2.72	2.41
Relative Time In Personal Space (%)	HERDR CARLA-only	3.43	**8.78**	14.62	20.52
	HERDR Pre-trained	2.99	12.39	**13.85**	23.09
	ORCA	**2.17**	10.9	14.16	**18.49**
Collision Rate (%)	HERDR CARLA-only	3.33	41.38	33.33	50.00
	HERDR Pre-trained	**0.00**	**30.00**	40.00	62.07
	ORCA	3.33	31.03	**27.59**	**41.38**

Table 5.6: Pedestrian avoidance comparison. Bold values indicate the best performance.

commands with every incoming image, which were subscribed to at 15 Hz. When run on a computer equipped with a capable GPU, the processing time of the model was found to be less than 2 ms allowing for the policy to run at a faster rate if necessary. However, due to time constraints, the system was not properly tested in an outdoor environment. Therefore, the system was only briefly tested indoors before the system was configured properly to run with a GPU. This limited the processing rate to 2 Hz. In a simple test where a person stood directly in front of the robot, it was observed that the system would set the steering angle to a larger magnitude, indicating it was attempting to manoeuvre around the person. Despite this effort, the system was unable to properly avoid as the commands were not being updated at an appropriate rate. Another issue when deploying to the robot was that the camera mounted off to one side of the robot. While this was not tested this would have likely caused an issue as the models are trained to evaluate actions from a centrally

mounted camera. Since this system does not directly take into account the shape of the robot the system would have a higher chance of colliding when turning to the side opposite the camera or when navigating through a narrow opening. A similar scenario was already observed in the simulation where the agent tried to turn through into a narrow passageway and failed to make a wide enough turn, clipping the corner. This is part of a general problem with the method as using only a single camera means the system has no peripheral information past its field of view, which is a significant disadvantage in crowded and confined spaces.

6 Conclusion

In this thesis, an autonomous navigation system was built using model-based reinforcement learning and deep learning that can avoid both static obstacles and pedestrians in an urban environment, while taking into account a model for human personal space. The method expands upon the framework used presented in [11], [12], by adding velocity control and utilizing highly realistic simulation and novel training techniques to train a system of safely navigating in low-density crowds. An analysis was completed for the parameters of the planner allows for more accurate behaviour tuning. The method outperforms a state-of-the-art method, showing a significant increase in consideration for pedestrians' personal space. Finally, early testing indicates the method has the potential to provide capable results in the real world when trained on high-quality simulation data.

6.1 Future Work

The system as a whole has many areas for improvement. As mentioned in section 5.2.1 the method would greatly benefit from a global planner that could provide higher quality waypoints, allowing for more robust navigation. The method could also be expanded to include multiple cameras to give the system better awareness in more crowded environments. This could be implemented in two ways, with either the images being concatenated forming one large image as the input to the deep learning model or to have different streams being fed to the same model and combining the reward from both sets of predicted states when updating the optimal trajectory. Another area that was out of the scope of this work and could be improved upon is the deep learning architecture. The current custom model is presumably not the most optimal, and replacing it with a more commonly used visual network backbone would potential improve predictions and performance. Finally, when deploying in the real world the model will likely need to be fine-tuned to account for the visual differences. This will require a data collection method that can determine the relative position of pedestrians visible to the camera and if possible, their orientation. One such solution would be to use a human detection algorithm and a depth camera to project the average depth onto the ground plane and use camera geometry to determine position.

Bibliography

[1] D. Muoio, *Here's how Tesla, Uber, and Google are trying to revolutionize the trucking industry*, Business Insider, Accessed: 2022-05-01, 2017. [Online]. Available: https://www.businessinsider.com/autonomous-trucks-tesla-uber-google-2017-6?r=US&IR=T.

[2] J. Schröder, B. Heid, F. Neuhaus, M. Kässer, C. Klink, and S. Tatomir, "Fast forwarding last-mile delivery–implications for the ecosystem," *Travel, Transport and Logistics and Advanced Industries, McKinsey & Company, July*, 2018.

[3] C. R. Aps, Accessed: 2022-05-01, 2022. [Online]. Available: https://capra.ooo/industries/smart-city/.

[4] iRobot Corporation, Accessed: 2022-05-01, 2022. [Online]. Available: https://www.irobot.com.

[5] J. K. Burgoon and S. B. Jones, "Toward a theory of personal space expectations and their violations," *Human communication research*, vol. 2, no. 2, pp. 131–146, 1976.

[6] L. A. Hayduk, "Personal space: An evaluative and orienting overview.," *Psychological bulletin*, vol. 85, no. 1, p. 117, 1978.

[7] T. A. A. Committee, "Di18.4 - automated micro-utility devices - accessibility feedback," Accessed: 2022-05-01, Nov. 2021. [Online]. Available: http://app.toronto.ca/tmmis/viewAgendaItemHistory.do?item=2021.DI18.4.

[8] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1989.

[9] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," *Advances in neural information processing systems*, vol. 31, 2018.

[10] R. Heinzler, P. Schindler, J. Seekircher, W. Ritter, and W. Stork, "Weather influence and classification with automotive lidar sensors," in *2019 IEEE intelligent vehicles symposium (IV)*, IEEE, 2019, pp. 1527–1534.

[11] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[12] ——, "Land: Learning to navigate from disengagements," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1872–1879, 2021.

[13] P. Anderson, A. Chang, D. S. Chaplot, *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[14] M. Sewak, *Deep Reinforcement Learning*. Springer, 2019.

[15] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, *Model-based reinforcement learning: A survey*, 2020. DOI: 10.48550/ARXIV.2006.16712. [Online]. Available: https://arxiv.org/abs/2006.16712.

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

[17] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[18] M. Stewart, *Simple introduction to convolutional neural networks*, Accessed: 2022-05-01, 2019. [Online]. Available: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 367, 373, http://www.deeplearningbook.org.

[20] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain, "Machine translation using deep learning: An overview," in *2017 international conference on computer, communications and electronics (comptelix)*, IEEE, 2017, pp. 162–167.

[21] E. Greenstein and D. Penner, "Japanese-to-english machine translation using recurrent neural networks," *Retrieved*, vol. 19, p. 2019, 2015.

[22] J. Zhang, H. Liu, Q. Chang, L. Wang, and R. X. Gao, "Recurrent neural network for motion trajectory prediction in human-robot collaborative assembly," *CIRP annals*, vol. 69, no. 1, pp. 9–12, 2020.

[23] A. Nguyen, D. Kanoulas, L. Muratore, D. G. Caldwell, and N. G. Tsagarakis, "Translating videos to commands for robotic manipulation with deep recurrent neural networks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 3782–3788.

[24] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[25] F. Gers, "Long short-term memory in recurrent neural networks," Ph.D. dissertation, Verlag nicht ermittelbar, 2001.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] C. Olah, *Understanding lstm networks*, Accessed: 2022-05-01, 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

[28] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[29] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[30] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE robotics & automation magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[31] T. T. O. Takleh, N. A. Bakar, S. A. Rahman, R. Hamzah, and Z. Aziz, "A brief survey on slam methods in autonomous vehicle," *International Journal of Engineering & Technology*, vol. 7, no. 4, pp. 38–43, 2018.

[32] F. Dayoub, T. Morris, B. Upcroft, and P. Corke, "Vision-only autonomous navigation using topometric maps," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 1923–1929.

[33] J. C. V. Soares, M. Gattass, and M. A. Meggiolaro, "Crowd-slam: Visual slam towards crowded environments using object detection," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 2, pp. 1–16, 2021.

[34] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 6015–6022.

[35] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5671–5677.

[36] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 31–36.

[37] Y.-H. Kim, J.-I. Jang, and S. Yun, "End-to-end deep learning for autonomous navigation of mobile robot," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, 2018, pp. 1–6.

[38] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun, "Off-road obstacle avoidance through end-to-end learning," *Advances in neural information processing systems*, vol. 18, 2005.

[39] P. Mirowski, R. Pascanu, F. Viola, *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[40] N. Hirose, F. Xia, R. Martıén-Martıén, A. Sadeghian, and S. Savarese, "Deep visual mpc-policy learning for navigation," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3184–3191, 2019.

[41] J. Kulhánek, E. Derner, T. De Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," in *2019 European Conference on Mobile Robots (ECMR)*, IEEE, 2019, pp. 1–8.

[42] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[43] J. v. d. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, Springer, 2011, pp. 3–19.

[44] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2017, pp. 285–292.

[45] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1343–1350.

[46] N. Ouadah, L. Ourak, and F. Boudjema, "Car-like mobile robot oriented positioning by fuzzy controllers," *International Journal of Advanced Robotic Systems*, vol. 5, no. 3, p. 25, 2008.

[47] G. Williams, A. Aldrich, and E. Theodorou, "Model predictive path integral control using covariance variable importance sampling," *arXiv preprint arXiv:1509.01149*, 2015.

[48] L. A. Hayduk, "The shape of personal space: An experimental investigation.," *Canadian Journal of Behavioural Science/Revue canadienne des sciences du comportement*, vol. 13, no. 1, p. 87, 1981.

[49] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004. [Online]. Available: `http://www.ars-journal.com/International-Journal-of-%20Advanced-Robotic-Systems/Volume-1/39-42.pdf`.

[50] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[51] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[52] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.