Robotic Navigation in Simulated Urban Environments

An investigation on the effect of uncertainty in the observed environment

This dissertation is submitted in part requirement for the Master of Science in Spatial Data Science and Visualisation at the Centre of Advanced Spatial Analysis, Bartlett Faculty of the Built Environment, University College London.

Candidate: Kristian Emil Lunow Nielsen

Date: 30/08/2019

MSc Spatial Data Science and Visualisation, TMSSDSAVIS01

Supervisor: Ed Manley

Word count:

Declaration

I, Kristian Emil Lunow Nielsen, hereby declare that this dissertation is all my own original work and that all sources have been acknowledged.

This dissertation is xxx words in length, from introduction to conclusion inclusive, excluding footnotes. Word count by Word.

Date: 30/08/2019

Kristian Email Lunow Nielsen

List of Figures

List of Tables

Acronyms and Abbreviations

DRL

RL

Notation

DRL

RL

Abstract

Abstract/Summary text

Contents

1. Introduction

The topic for this dissertation is robotic navigation in simulated urban environments, and the purpose of the study is to explore the use of deep reinforcement learning (DRL) in this context. Deep learning (DL) is an area with growing attention within urban analytics, as urban infrastructure is being transformed by the advances in artificial intelligence and robotics. This motivates exploring DL, through DRL, on a use case, with increasing focus from some of the world's biggest corporations [Nichols, 2019, FedEx, 2019, Scott, 2019]. That case of study of this research is DRL's ability to deal with some of the challenges faced by robotic citizens, the autonomous delivery robots (ADR's).

Urban environments are complex dynamics of interactions between objects, and navigation herein requires an ability to explore, foresee, adapt and plan. Successful prior work on robotic navigation in crowded urban environments (CUE's) rely on the use of particle filters, for building a probabilistic map of the environment to handle planning, and a combination of human interaction as well as goal-directed exploration for exploration and adaption of/to the environment [Lidoris et al., 2009, Kümmerle et al., 2013].
The need for human interaction limits the autonomous degree of the robot, potentially limiting the usages of the robot to a certain time period doing the day.
Recent advances in DL, implying the rise of DRL, could present a way for a robot to improve spatial awareness. Thereby circumventing the need for human interaction and allowing the robot to operate at any time. The first work on DRL for robotic navigation and obstacles avoidance have seen the daylight [Kahn et al., 2017, Mirowski et al., 2018, Zhou et al., 2019].
All three are bound to tackle static obstacles, and so do not address a major challenge of

CUE's, namely dynamic obstacles.

Therefore, the motivation and context of this study is to obtain insights on how a model-free DRL approach with continuous actions and partial observability, tackles the challenges of a simulated ADR navigating a crowded environment with static and dynamic obstacles. The study sheds light on two additional aspects; the effect of uncertainty about the observed environment, and how different training strategies can aid the learning process.

1.1. Theoretical Background

The fundamental aspects of robotic navigation can be boiled down to learning and planning, which is also the fundamentals of reinforcement learning (RL) through the tasks of control and prediction. Control tasks are concerned with learning the best policy, while prediction tasks evaluate the policy at hand.

Traditional RL literature distinguish between model-based and model-free algorithms, where model-based algorithms rely on planning and model-free algorithms rely on learning [Sutton and Barto, 2018].

This study is mainly a control task, on obtaining an optimal policy for locating a target while avoiding obstacles, and not about planning the optimal route to the target given a policy.

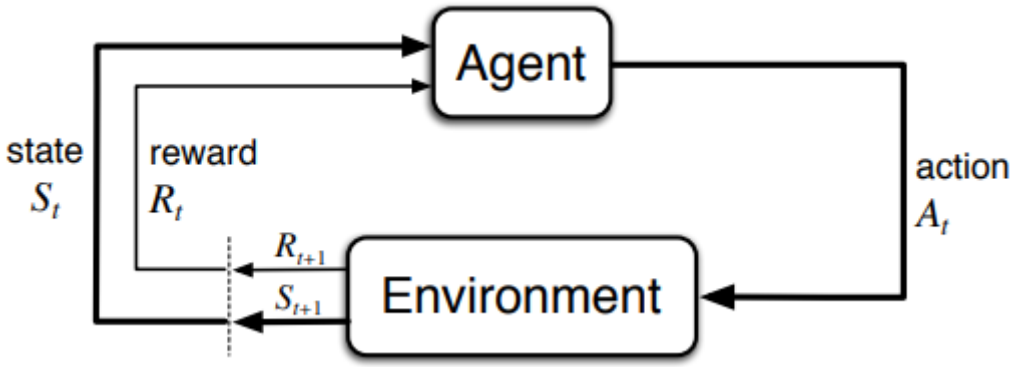
The RL algorithm used in this study is called Proximal Policy Optimization (PPO) [Schulman et al., 2017a, Schulman et al., 2017b], which belongs to class of policy gradient methods. This class of methods essentially extends traditional model-free control algorithms, as Q-learning [Watkins and Dayan, 1992][Watkins and Dayan, 1992], into large scale real-world applications.

1.1.1. The Reinforcement Learning Problem

The traditional RL set-up consists of an agent (the RL system) and environment in which it operates, see figure 1. Every RL problem is about solving the Markov Decision Process (MDP), in the sense of optimizing some objective function given the MDP, as the MDP fully characterise problem. This objective function can either be a policy function, value function or the advantage function (difference between the policy and value function), depending of the problem and algorithm.

The decision process is a Markov decision process because all history of the environment is captured in the most recent value;

Figure 1.1.: Markov Decision Process



Credit: [Sutton and Barto, 2018]

Any MDP is made up by a set of actions, $a \in A(s)$, a set of states, $s \in S$, a set of rewards, $r \in R$, and sometimes explicitly, yet rarely in practice, a set of state transition probabilities. The latter won't be elaborated any further, as they are only relevant for smaller finite[1] MDP's, which occurs rarely in practice, at least in any interesting applications.

The order in figure 1.1: The agent observes the initial environment and takes an action at time t, transitioning the environment to state t +1 and emitting a reward to agent. Based on the new observed state and the reward obtained, the agent chooses a new action, and this cycle repeat until the terminal state. An important thing to mention is that the reward does not say directly how good the action was, but how the good the resulting

[1] All sets of the MDP are finite.

state is.

The reward signal is often sparse, meaning the agent only receives a reward at the terminal state, i.e. at the end of an episode. If the terminal state is good, the agent receives a large positive reward, if bad, a large negative reward, and zero in all states[2] leading up to the terminal state.

1.2. Purpose of the Study

This research aims at addressing the challenges that are surrounding the application of DRL for navigation and obstacle avoidance tasks in CUE's. The implication hereof is that the conducted research is mainly methodological. This research should be regarded as a preliminary study, to be further extended to generalise to the real world, because of the complex nature of urban environments, and especially crowded areas. This study outlines the basis for filling the gap on DRL for dynamic obstacle avoidance, as this, to the knowledge of the author at the time of writing, is yet to be explored. This study was at the same time an opportunity to explore Unity and the toolkit ML-Agents by [Juliani et al., 2018a] for conducting DRL research, under realistic physical settings.

The intention of this study is not to promote DRL to replace existing methods in traditional robotic navigation, but hopefully to aid these, to achieve smarter and safer cities in the future.

The first objective of this study is to address the challenges emerging when using DRL for navigation and obstacles avoidance in dynamic environments.
The second objective of the study is to address ways to tackle these challenges and promote meaningful learning in the agent.
The final objective is to address how uncertainty around the observed environment affects the learning taking place.

[2]This is a modified truth, as it is common practice to reward the agent with a minor negative reward after each state (also called step), to incentive fast learning.

1.3. Significanse of the Study

This study contributes to the continuing development of DRL for robotic navigation in urban environments, by addressing some of the challenges still present and test novel design methods of the training phase. Addressing the ongoing challenges, hopefully enables focused future research [Irpan, 2018], avoiding rediscovering of known results. Most prior research in this area has been concerned with developing novel methods to tackle the challenges at hand, potentially neglecting the design of the training phase. Design of the training phase has previously been shown to have a significant effect [Bengio et al., 2009, Mirowski et al., 2018].

Two ways to design the training phase are curriculum and imitation learning (CL and IL respectively), where the latter is widely adopted in prior research on robotic navigation [Kahn et al., 2017, Zhou et al., 2019]. The former appears overlooked in the context of robotic navigation, especially in the specific context of robotic navigation in CUE's using DRL.

This study use CL in the training phase, and therefore addresses how the use of CL can aid to manage some of the challenges of robotic navigation in CUE's using DRL.

1.4. Scope of the Study

The study explores state-of-the-art methods and serves as a framework to evaluate the current state of the field of DRL for robotic navigation in CUE's through simulation. Simulation is done under realistic physical settings, intending to smoot the future generalisation to real-world applications. The aim is that even though the study is limited to simulation, it could serve as a baseline for future real-world applications.

The investigated DRL algorithm is based on what is available in the ML-Agents toolkit, and this algorithm is the state-of-the-art for continuous control tasks [Schulman et al., 2017a].

The literature review is not intended to be complete on RL/DRL nor robotic navigation. It serves to present the current state of the field, by the most important and latest con-

tributions relevant for robotic navigation in CUE's using DRL, in order to address the objectives of the study. Furthermore, fundamental concepts in RL, as well as key innovations leading to DRL, and the class of the method used in this study are outlined. That limits the literature review to viewing some concepts of model-free control, see [Sutton and Barto, 2018] for an extensive coverage of RL.

2. Literature Review

The following proceeds by first reviewing RL, the emergence of DRL and *the*[1] class of DRL algorithms, before reviewing the literature on robotic navigation in urban environments and the use of RL as well as DRL in this context. The order of the review is to provide some base knowledge of the chosen method, before reviewing the field of application surrounding this study.

2.1. Reinforcement Learning

Perhaps the most important discovery within RL is *temporal-difference learning* (TD), originating from animal learning psychology. TD was originally acknowledged in RL context by [Minsky, 1954, Samuel, 1959], and proposed in known format today by [Sutton, 1984, Anderson, 1986]. TD methods leverage on both *dynamic programming* (DP) and *Monte Carlo methods*[2] (MC), by using bootstrapping as DP, making TD online, and sampling as MC, making them model-free. Different TD methods for control tasks exists, some being *SARSA*, *Q-learning* and *Expected SARSA*, and they differ by the way they handle the estimate of the objective function in future states.

The following focuses on Q-learning, see [Sutton and Barto, 2018] for a review of the other two.

Q-learning is an *off-policy* learning method, which mean that the actual action is drawn from a *behaviour* policy, μ, which is compared to an alternative successor action, drawn

[1] A reference to the fact that this class of methods is the most used today [Karpathy, 2016].
[2] see [Sutton and Barto, 2018] for a description of both methods.

from the target policy, π. Off-policy learning methods are useful because they allow one to re-use experience from old policies, as we will see is useful, and to learn about the optimal policy while following an exploratory policy.

Given the two actions, $A_{t+1} \sim \mu(A_t|S_t)$ and $A' \sim \pi(A_t|S_t)$, the online update of the state action-value function is done according to 2.1.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)) \tag{2.1}$$

Both policies are improved, with the target being updated greedily[3] and the behaviour ϵ-greedy[4], both w.r.t. $Q(s, a)$, which simplifies the target as seen from 2.2.

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + \gamma Q(S_{t+1}, \left(\tbinom{\arg\max}{a'}\right) Q(S_{t+1}, a')) \\ &= R_{t+1} + \left(\tbinom{\max}{a'}\right) \gamma Q(S_{t+1}, a') \rightarrow \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \left(\tbinom{\max}{a'}\right) Q(S_{t+1}, a') - Q(S_t, A_t)) \end{aligned} \tag{2.2}$$

The substituted expression is just the current estimate of the optimal future value.

Q-learning converges to the optimal policy and action-value function, with probability 1, under the assumption that all states are visited infinite number of times[5], illustrated in (3).

$$Q(s, a) \rightarrow q^*(s, a) \; for \; \# \; of \; episodes \rightarrow \infty \tag{2.3}$$

The Q-learning algorithm showed in figure 2.1 can seem too simple to work on real-life large-scale control problems. However, it illustrates the high-level idea, about improve

[3] A greedy policy is a policy, that chooses the action resulting in the maximum outcome.

[4] A ϵ-greedy policy is a policy, which with probability ϵ chooses a random action and with probability $1 - \epsilon$ chooses the greedy action.

[5] See [Sutton and Barto, 2018, Chapter 9, Section 4] for a full description.

Figure 2.1.: Q-learning algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Credit: [Sutton and Barto, 2018]

an estimate in online fashion – which is essential before covering the policy gradient methods. Q-learning is in practical purposes couple with some minor tricks, as *batch updates*, *eligibility traces* and *function approximation* for more efficient scalable learning. Batch updates will be explained in a section to come, but the interested reader should consult [Sutton and Barto, 2018, Chapter 12] for a description of eligibility traces.

Doing convergence of 2.3, the function on the left-side is a so-called approximate state action-value function, which in the simple case is a look-up table with size $(S \cdot A)\, x\, (S \cdot A)$. In real-life applications, such a table is often impossible to store in memory. To ensure scalability, function approximations are used, implying the look-up table is replaced with a parameterised function, which could be a linear combination of features, a neural network or something third. The full description of function approximations are covered in [Sutton and Barto, 2018, Chapter 9,10 & 11], but note that the function approximation changes the state action-value function from $q(s, a)$ to $q(s, a, w)$, and updating the weights in the correct direction improves the approximate state action-value function in the desired direction.

Before moving on to policy gradient methods, which are mostly used in the industry today [Karpathy, 2016], it is worth to look at what drove the transition to deep RL.

Today's choice of function approximators are most often neural networks (nets), and the

deep part of DRL referrers to the structure of the net. A deep net consists of many layers, enabling learning of complex high-dimensional non-linear functions, as each layer learns different aspects of the data passed through [LeCun et al., 2015]. *DL*, which is the high-level designation for variations of deep nets, belongs to the class of general-purpose learning procedures [LeCun et al., 2015]. General-purpose learning procedures can learn good feature representations directly from the data, avoiding the need for hand-crafted, often non-generalisable, features and at the same time managing the *selectivity-invariance dilemma* (SID). SID in feature engineering is the ability of features to produces representation that are selective to aspects of the image that are essential for discrimination, but that are invariant to irrelevant aspects such as the pose of the animal [LeCun et al., 2015].

Some of the most important advances in DL, leading to DRL, is;

- The adaption of the rectified linear unit (ReLU) activation function as the standard.

  Activation functions are used to squeeze the output of neurons to a bounded range, typically $[0, 1]$. ReLU, $max(0, z)$, bounds the outcome to $[0, \infty[$, which has shown to provide much faster training of deep nets [Nair and Hinton, 2010, Krizhevsky et al., 2012, LeCun et al., 2015].

- Increased GPU power and the possibility of parallelised GPU training.

  More powerful GPU's allows increasing storage of data in memory, and parallel GPU implementations of deep nets allow for faster training and handling of larger amounts of data than ever before [Krizhevsky et al., 2012, LeCun et al., 2015].

- The raise of convolutional nets (ConvNet).

  ConvNets are built with an eye for processing data in the form of multiple arrays, and the typical architecture[6] consists of local connections, shared weights, pooling and many layers [LeCun et al., 2015]. ConvNets has been shown to training faster and exhibit greater generalisation ability than stacked nets of fully connected layers [LeCun et al., 2015]. The trend is to combine different types of nets in the final deep net, as seen in [Krizhevsky et al., 2012, Sermanet et al., 2013, Mnih et al., 2013].

[6] See [LeCun et al., 2015] for a detailed description, of the architecture and each of the components.

These advances along with the use of *experience replay*[7] [Lin, 1992], made the difference for the first successful deployment of a DRL model, to learn a control policy directly from high-dimensional sensory input [Mnih et al., 2013]. The agent of [Mnih et al., 2013] learned to play 7 ATARI games, with no adjustment between the games, and surpassed previous implementations on six of the games while obtained above human-expert level on three of them. The work of [Mnih et al., 2013] motivated two other important papers, [Mnih et al., 2015] and [Silver et al., 2016]. [Mnih et al., 2015] extend the work of [Mnih et al., 2013] to 49 ATARI games, beating all previous implementations, obtaining the level of a profession human game-tester across all 49 games and achieving above human performance on 23 games [Mnih et al., 2015]. [Silver et al., 2016], deploying a combination of deep nets and a tree search algorithm, obtained master level[8] in the boardgame GO, which was regarded as one of the grand challenges, because of its enormous state space consisting of 250^{150} possible moves.

Having reviewed a traditional RL method, as well as the key innovations leading to DRL, including a few key applications, it is now time to review a class of DRL methods. This class is called *policy gradient (PG) methods*, and the method used in this study, *PPO*, belongs to this class.

2.2. Deep Reinforcement Learning

This section starts out with a simplified example from [Karpathy, 2016], because the example gives a good intuitive idea about how PG methods work. After this example follows a review of the method used in this study.

[7] N previous observations are stored, over many episodes, from which T experiences are randomly sampled from, note $T \ll N$. Updates are done on the sampled experiences, and the agent chooses an action according to an ϵ-greedy policy, i.e. off-line learning, [Mnih et al., 2015].

[8] [Silver et al., 2016, Page 5].

2.2.1. Policy Gradient Methods

Modern PG methods seeks to learn a parameterised policy function to select actions, with some approximating a value function as well, to aid the learning process. The latter class are referred to as *Actor-Critic* methods, and the method used in this study belongs to this class.

The objective of PG methods is to learn the policy parameters based on the gradient[9] of some scalar performance measure, with respect to the policy parameters [Sutton and Barto, 2018]. To do that, the parameters of the approximated policy function, most often the weights in a net, are adjusted according to the reward signal $J(\theta)$[10]. How must to adjust, and so how to update the policy, is determined by the gradient of the scalar signal[11] and called backpropagation[12], seen from 2.4.

$$\theta_{t+1} = \theta_t + \alpha(\nabla \hat{J}(\theta_t)) \tag{2.4}$$

With α being the learning rate and $\nabla J(\theta_t)$ the gradient of the reward function.

The task in [Karpathy, 2016] is to learn an agent to play the ATARI game of Pong, from nothing more than the pixels from the emulator[13], using a basic PG.

The structure is as follows; we receive an image, of size $210x160x3$[14], and get to decide whether to move up or down. After every action, the agent is rewarded; $+1$ if the ball went past the opponent, -1 if the ball went past us and 0 otherwise. The objective is to beat the opponent, and so maximising the reward.

(Karpathy, 2016) approximate the policy function with a net, and adjust the weights based on the actions taken. Say an episode consists of 200 steps, implying 200 actions

[9] The first derivative of a function.
[10] The reward signal is a function of the policy parameters θ.
[11] See [LeCun et al., 2015, P. 436, 2nd paragraph towards the bottom] for a straightforward explanation.
[12] Training of nets using gradient of the objective function, see [Rojas, 1996, Chapter 7] and footnote 11.
[13] A piece of software, integrating The Arcade Learning Environment [Bellemare et al., 2015] and the preferred script editor of the researcher.
[14] Height, width and RGB channels.

to be taken. If just 101 of the actions are good actions, the outcome will be a reward of +1. Overtime, as we adjust the weights in favour of the good actions, the share of good actions within one episode increases, implying that the total number of episodes leading to a positive reward increase.

2.2.2. Proximal Policy Optimisation

PPO is the embedded DRL algorithm in the ML-Agents toolkit, and it has been shown to outperform far more complex PG methods while being more general and having better sampling complexity [Schulman et al., 2017b].
PPO optimises a surrogate objective function, based on sampled experience (batch updates), from which the policy is updated, and an action is chosen from the updated policy. Simple and effective[15].

With the theory outlined, it is now time to review the field of application for this study.

2.3. Robotic Navigation in Urban Environments

Besides limiting this part of the literature review to consider robotic navigation in urban environments (UE), it is also restricted to only consider navigation in unknown environments as it resembles to the model-free DRL approach studied in this study.
This part of the literature review is divided into two parts; the traditional literature, which does not include the use of RL, and the DRL-based literature.

Robotic navigation in UE's comes with a lot of challenges. The four main challenges surrounding robotic navigation in UE's are mapping, traversability analysis, localisation and planning.
Mapping concerns obtaining a high-level idea about the area om deployment.
Traversability analysis uncovers the potential challenges of the environment.
Localisation provides a belief about the relative position of the robot and the challenges

[15]See appendix for detailed technical explanation.

uncovered by the traversability analysis.
Planning seeks to determine the optimal route, from the current position to the target, given the three other components.

Common for all the papers examined on the traditional literature, is that each of the four challenges are handled by separate systems, within the robot.

The consensus way to handle mapping appears to be using particle filters, often using a SLAM[16] module [Georgiev and Allen, 2004, Lidoris et al., 2009, Maria Bauer et al., 2009, Trulls et al., 2011, Kümmerle et al., 2013]. This approach implies computing an occupancy grid; however, it can be troublesome in terms of memory to store large scale occupancy grid, which is something to consider design-wise.

Localisation is done in different ways; [Lidoris et al., 2009, Maria Bauer et al., 2009, Kümmerle et al., 20 uses sampled based methods, as Monte Carlo Localisation. [Georgiev and Allen, 2004] uses a combination of GPS coordinates, odometry and visual image processing for localisation and [Trulls et al., 2011] uses an online particle filter implementation based on a 3D geometric model of the environment.

Traversability analysis is usually done using a combination of horizontal and vertical lasers, to measure distance to objects, and potentially changing positions (needed to locate dynamic obstacles as pedestrians) [Lidoris et al., 2009, Maria Bauer et al., 2009, Trulls et al., 2011, Kümmerle et al., 2013].

Planning is done differently; [Lidoris et al., 2009, Maria Bauer et al., 2009, Trulls et al., 2011] use the A* algorithm to calculate the shortest path through the crowded area. [Kümmerle et al., 2013] uses a hierarchical set-up, which consists of three planners, a high-level agent planning the overall go-to-route, an intermediate-level agent which calculates waypoints and a low-level agent calculating the velocity of the robot based on the waypoints.

The take-aways from the presented literature review is that robotic navigation is a field which has been subject for much research. Some challenges are handled consensually while others are subject for experimentation of novel methods.

[16]See [Lidoris et al., 2009] for a description.

2.3.1. Robotic Navigation in Urban Environments using Reinforcement Learning

The literature on the use of DRL in the specific field of robotic navigation in UE's is currently sparse, but it is an area with growing interest from the research community [Zhou et al., 2019]. The implication is that no papers exists, at least to the knowledge of the author, which handles all four challenges, mentioned in the previous section, directly.

One paper from DeepMind by [Mirowski et al., 2018] presents an agent, which can navigate through a simulated city based in visual input – and transfer its knowledge to other cities. However, their agent operates with discrete actions and the environment possesses only static obstacles. Their agent uses two interesting additions to the traditional DRL architecture; A method named Long Short-Term Memory [Hochreiter and Schmidhuber, 1997] to incorporate memory in the agent, and the use of CL, enabling the agent to be introduced to increasing complexity of the environment doing training.

Another paper from OpenAI by [Kahn et al., 2017] focuses on obstacle avoidance using DRL, and they incorporate uncertainty-awareness in the agent, for safer navigation. The short comings of this article, compared to the traditional literature, is that they only consider static obstacles. Furthermore, they use a model-based DRL model.

The final paper considered here, is a paper by [Zhou et al., 2019] which uses a hierarchical agent to perform goal-directed navigation while being adaptive to changes in the environment. The high-level agent handles the goal orientation and the low-level agent takes care of changes in the environment. They train in a simulated environment and show that the agent generalises to the real world.

All the papers deliver promising insights, but two of them are especially interesting, in terms of this study; [Mirowski et al., 2018] uses CL successfully for navigation, and [Zhou et al., 2019] trains an agent, the low-level one, to be able to deal with changes in the environment, which also is able to transfer to the real world.

3. Methodology

Coming off the literature review, three points are worth to have in mind;

- Scalable RL, through the recent advances in DL, have accomplished promising results in other areas, from raw visual observations.
- Robotic navigation possesses four main challenges; mapping, traversability analysis, localisation and planning.
- The use of DRL for robotic navigation in UE's has so far been concern with avoiding static obstacles, using both visual and sensor observations.

This section outlines methods used to address the objectives of this research, and how they are investigated.

There are three objectives, as mentioned in section 1.2, namely;

- To investigate the ability of DRL to tackle the most common challenges of CUE's.
- To explore ways to ease learning of the robotic agent.
- To investigate how uncertainty about the observed environment affects the performance of the robotic agent.

The objectives of this study are mainly related to the challenge of traversability analysis and localisation in robotic navigation. Mapping of the environment is a consequence of the agent learning and planning is not considered in this study.

This study simulates a robotic agent, learning an environment which possesses some of the core challenges found in UE's, which are static, semi-static and dynamic obstacles. The robotic agent represents an ADR, and the environment represents an area which is

less congested with traffic, like an area of pedestrian streets or a campus university.
The current state of ADR's is briefly outlined in section 3.1 to, in affiliation with the insights obtained from the literature review, address the challenges that the simulated environment needs to possess.
To evaluate the suitability of DRL, to tackle the core challenges, a simulated environment is needed. The framework is developed in Unity, and a justification for choosing Unity is presented in section 3.2.
The simulated environment, and its content, is outlined with an eye for the objectives, and a more detailed description is found in appendix.

Robotic agents of varying complexity are compared, to evaluate the effect of different inputs on the quality of the decision making of the robot. Their performance is evaluated on their *average cumulative reward* (ACR), *average episode length* (AEL) and their ability to avoid obstacles.
The reason for evaluating different robotic agents is because more sophisticated agents comes with a higher computational cost, which is a concern when real-time decision are needed, as is the case with deployed ADR's.

Finally, the different robotic agents are compared on their ability to handle uncertainty in the observed environment, elaborated in section 3.3.2.1.

3.1. Autonomous Delivery Tobots Today

An increasing number of companies are putting they attention on autonomous delivery robots [FedEx, 2019, Scott, 2019], to meet increasing customer expectations of companies to ride the technology weaves, to enable low-cost-low-emission products. Starship, founded by two of the Skype co-founders, are regarded the leader in the ADR race, with their fleet of ADR's having logged over 100.000 kilometres and more than 25.000 deliveries under the wheels. The fleet has been deployed in pilot areas of cities like London, New York and Washington DC [Nichols, 2019, Merrit, 2019], and the fleet has gained enough experience to surpass the need for any handholding [Nichols, 2019].

Their newest launch is autonomous delivery of food and beverages at George Mason University, Maryland. The partnership is to accommodate the rising need for smart solutions in a high-paced-high-expectation environment, where nutrition sometimes is overlooked [Nichols, 2019].

The technology underlying the ADR's of Starship is proprietary, yet [Pärnamaa, 2018] reveals that neural networks does play a role in localisation and traversability analysis of their ADR's.

The specifications of [Scott, 2019, FedEx, 2019, Starship, 2019] serves as input to the development of the robotic agent, to ensure realism in the simulations, together with realistic physical settings, elaborated in coming section.

Common for the areas of deployment of today's ADR's is that they are areas dominated by pedestrians, and crowds are likely to emerge. The core challenges are static and dynamic obstacles, as well as a combination of the two, elaborated in section 3.3.1.

Having outlined the stage of which this research is conducted, in terms of a bounded pedestrian dominated area, and the challenges that such an area possess, it is time to access how this can be done in a safely manner.
Unity, a state-of-art game development platform and the simulation engine for this study, provides a framework in which interaction between agents can be safely investigated under realistic physical settings.
Utilising Unity enables this study to investigate how RL-agents handles some of the challenges faced by the current generation of ADR's, in a safe and realistic, yet simplified, setting.

3.2. Unity – as a Simulation Engine for Research in DRL

Unity is a game engine, that enables the use of sensor and visual input while providing the possibility of realistic physical rendering, which makes it an interesting application to conduct research on artificial intelligence (AI) on [Sadeghi and Levine, 2016,

Juliani et al., 2018a, OpenAI et al., 2018].

The underlying engine runs in de-synchronized fashion, implying low latency and distributed computing. This is important when developing software used for real-time decisions, as it is needed in ADR's.
Additionally, Unity provides various features to support reduced simulation time, such as the ability to run simulations at least 100 times faster than real time, while still maintain physics and frame rendering, and the option to deploy concurrent environments. Limiting simulation time as much as possible, without harming the performance, is important when working with complex environment, agents and methods. Depending on the setting, some applications requires millions of double-digit observations to converge to the optimal outcome [Hessel et al., 2017].
These computational features are appealing for this study, because part of the objectives in this study seeks to investigates the effect of using visual input in agents as well as parallelised training, which are two aspects that are computational demanding.

Unity is a serious candidate for modelling the complex dynamics of urban environments, via the ability to replicate real-life physical complexity, and thereby enabling realistic movement patterns as well as interaction between objects. This is essential if the objective is to generalise the results to the real-world, as higher similarity between the digital environment and real world increases the likelihood for generalisation.

It is not a direct objective of this study to generalise the results obtained to the real world, but it is hoped that this study can serve as a humble baseline for further research.

The analysis in section 4 proceeds by examining four slightly different robotic agents' ability to handle the challenges of the simulated environment, with and without any aids as well as under uncertainty about the observations received.
The analysis address all three objectives, and it is intended to do it in a diversified way, by examining different robotic agents.

The following section describes the elements of the simulated environment, and how they help to address the objectives of this study.

3.3. The Environment

The simulated environment is intended to capture the core dynamics of pedestrians-dominated areas, which implies that the simulated environment contains, besides the robotic agent, four elements; static obstacles, dynamic obstacles, crowded areas and a target. Each these are described in turn below, but a brief overview looks the following;

Environment: Pedestrian-dominated area, as an area of pedestrian streets or a campus university.

Robotic agent: An RL-based goal-seeking ADR.

Static obstacles: They represents everything static in the built environment, such as building, bins, mailboxes or even boundaries of pilot areas.

Dynamic obstacles: They represent pedestrians, sharing the walkable area with the robotic agent.

Crowded areas: They simulate areas with a higher probability of potential collisions, which requires increased effort of the robotic agent to pass through.

Target: It represents the delivery point of the ADR.

A full technical description of the environment is found in appendix, covering configurations of each element, training, training hyperparameters and the use of TensorBoard to observe training.

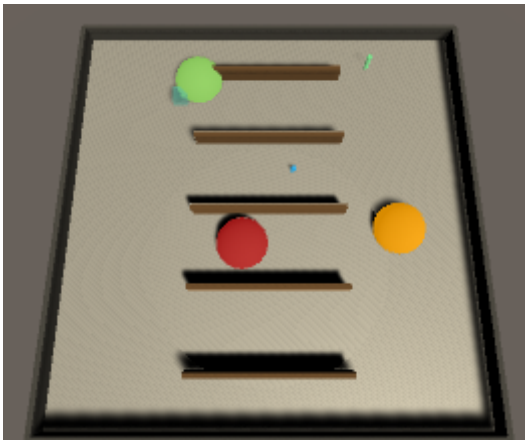
3.3.1. Environment

The environment under investigation is limited to reflect the core dynamics of pedestrian-dominated areas, and an example of a state in the environment is seen from figure 3.1.
A city example could be the area around Carnaby Street in Soho, London.
The simulated environment possesses difficult areas, located between the top/bottom static obstacle and the nearest wall, and it is difficult because it is narrower that the area between any two static obstacles. The difficult areas could represent varying street sizes

in a city area.

They present an interesting challenge, because they can help shed light on the effect of the spatial ordering of objects on learning.

Figure 3.1.: The Environment



The environment contains one moving pedestrians (light green), three crowded areas with varying density (red (high), orange (medium), green (light)), the target (dark green square) and the five static obstacles.

Simulating these somewhat simplified urban areas, i.e. pedestrian-dominated areas, is justified by the current state at which ADR's are today, i.e. still being in an early stage and facing challenges by the unpredictability of the real world [Nichols, 2019].

3.3.2. Robotic Agents

The robotic agents represent ADR's, and comprehensive technical description is available in appendix. The agents are RL-based agents powered by the ML-Agents toolkit by Unity. The toolkit embeds the DRL in the agent, by linking a Python API to Unity. For more detail, see appendix.

The performance of four different agents, with increasing complexity, are tested;

Agent 1 : Pure sensor observations.

The agent is equipped with 18 sensors, spanning 180 degrees in front of the agent, with a length of 50 metres. The effect of the length of span of the sensors is not investigated, and it is chosen arbitrary. The sensor returns the normalised distance to the detected object, with the distance normalised by the length of the sensor ray, 50 meters here.

The use of sensor observations resembles with the use of LIDAR sensors on real robots [Georgiev and Allen, 2004, Lidoris et al., 2009, Kümmerle et al., 2013].
This agent has as such no idea about the location of the target and needs to search for it. It can seem unrealistic, in terms of an ADR, but it provides a possibility to evaluate the effect of knowing the distance to the target.

Agent 2 : In addition to sensor observations, this agent is equipped with information on crowded areas nearby.

The agent is provided with the density and the normalised distance to crowded areas with a radius of 25 meters. This resembles with the use of radar information, and the idea stems [Pärnamaa, 2018].
The distance is normalised by the radius of the radar.

Agent 3 : In addition to sensor observations and information on crowded areas nearby, this agent is provided with the distance to the target.

The distance to the target is normalised by the length of the diagonal of the environment, as that is the maximum theoretical distance possible between the agent and the target.

Agent 4 : Sensor observations, Crowded-area information, distance to target and greyscale image observations.

The choice of using greyscale images, with a size of 84x84, is motivated by [Mnih et al., 2013], to limit the computational expense of using image observations.

All distances are normalised, to ensure better convergence properties of the policy net.

The action space of the agents consists of a single action, namely degrees to turn, implying that they are moved forward with a constant speed. An interesting topic for future research is extending the action space of the agents.

Investigating agents with varying complexity enables evaluation of the robustness of each agent, as well as the computational costs of the increasing complexity. The former is especially interesting when evaluating the effect of uncertainty, as is one of the objectives of this study.

3.3.2.1. Uncertainty

Uncertainty is introduced to the environment by adding Gaussian noise, $x \sim N(0, 1)$, to the sensor observations received by the agent, when the agents gets within 15 meters of a crowded area.
The uncertainty captures the uncertainty surrounding the actual presence of detected sensors, inaccuracy as a result of challenging weather conditions, or the use of cheap hardware to limit the cost of the ADR.
An example could be inaccuracy of the sensors from being covered by dust or snow.

The argument for investigating the effect of uncertainty on the agent' perception about the environment[1], instead of the on the rewards, which could be thought of as an alternative, is based on the theory underlying RL.
RL is a basically a function mapping on actions and states, and the goal is to obtain a quality policy which proposes a future action, based on the current action and the observed state of the environment. The quality of the policy, via the recurrent policy updates, is determined by the received reward signal of the environment. The reward signal seeks to incentive rewarding behaviour and disincentive undesired behaviour.
If the reward function is stochastic, the policy will in turn be stochastic, which in turn do not sustain any learning.

Modelling the noise as Gaussian provides deviations in both directions, which seems realistic given the potential causes. Choosing a standard Gaussian distribution implies that the expected distance is unchanged, and that the average deviation is around 6.6%[2] , which does not seem drastic. The choice of distribution is discussed a bit more in the discussion, and it is certainly a topic for future study.

The Gaussian noise is generated by performing a *Box-Muller transformation* of a two-dimensional continuous uniform distribution, seen in 3.1, outputting a two-dimensional bivariate normal distribution [Givens and Hoeting, 2012, Tabel 6.1]. One of the Gaussian variables are chosen with an equal probability.

[1] I.e. on the sensor observations.
[2] The standard deviation divided by the distance at which the uncertainty occurs, $(1/15) * 100 = 6.6\%$.

Box-Muller transformation

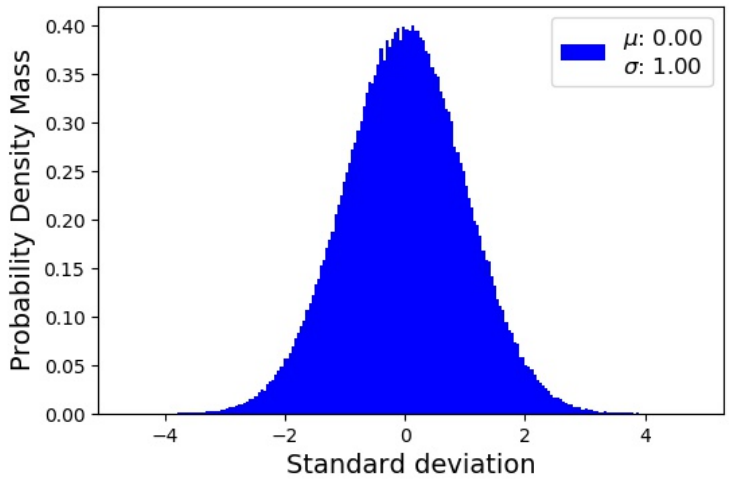
If U_1 and U_2 are uniformly and independent distributed, between 0 and 1, then

$$X_1 = \mu + \sigma\sqrt{-2logU_1}cos(2\pi U_2)\ \Lambda\ X_2 = \mu + \sigma\sqrt{-2logU_1}sin(2\pi U_2) \tag{3.1}$$

are independent $N(\mu, \sigma)$.

Figure 3.2 shows the distribution of sampled noise from the model, which verifies that the noise is in fact standard Gaussian distributed.

Figure 3.2.: Sampled noise



3.3.3. Rewards

Rewards are the fundamental driver of learning, irrespectively of the agent being real or artificial. Rewards occur with varying frequency, yet often distant, and they come in two types; extrinsic and intrinsic.

All characteristics of rewards, referred to as the reward function in RL literature, shapes the learned policy function. Headless treatment of the reward function results in poor policies, while careful design of the reward function enables generalisable high-quality policies.

3.3.3.1. Extrinsic

Extrinsic rewards come from outside the agent, and thereby emitted by the environment.

The environment emits a reward of -1 if the agent collides with static obstacles, walls or pedestrians, in hope to disincentive behaviour resulting in these collisions. Differently, collisions with the target are rewarded with $+1$, to incentive goal-seeking behaviour.

To motivate smarter decisions, and thereby quicker learning, the agent is penalised, -0.0005, for every step in the environment. The goal is to obtain a policy which provides the agent with navigational abilities, exhibiting characteristics that resembles with shortest-path-optimisation using A^* search, used in traditional robotic navigation research [Maria Bauer et al., 2009, Trulls et al., 2011, Kümmerle et al., 2013].

A subobjective of this study is to learn the agent, that crowded areas require more effort to navigate through. To do so, the agent is penalised for every step in the crowded area, proportionally to the density of the area, see section 3.3.5.

3.3.3.2. Intrinsic

Intrinsic rewards are internal rewards, encouraging curiosity-driven exploration.

A full treatment is given by , but the overall idea is to use two nets to predict the next action and observation respectively and compare the predicted values to the observed values. The difference is the intrinsic reward, and the bigger the difference, the bigger the reward.

The agents of this study are trained with curiosity-driven exploration, seen from figure 13, because of the sparse nature of rewards associated with navigational tasks.

3.3.4. Pedestrians

The pedestrians are *NavMesh agents* , which is Unity's build-in AI class, useful for spatial queries, as pathfinding. NavMesh agents can interact with other NavMesh agents, as well as avoid other moving obstacles, enabling a layer of social interaction with relative ease.

The behaviour of the pedestrians is that they every 10 seconds draw a feasible random point within 50 meters radius and move to that point. The time interval and radius can be set by the researcher through the custom script attached to the pedestrians, and the numbers are chosen because they fit well with the environment.
A feasible point, is a point which lies within the environment. Unfeasible points are possible because the points are drawn from within a circle, with the position of the pedestrian as origin and the radius specified by the researcher.
If the pedestrian is near a border of the environment, unfeasible points can be drawn. However, points are repeatedly drawn until a feasible point is drawn. In practice, that implies that the pedestrian from time to time stops for a very short duration, which is in accordance with typical human behaviour.

3.3.5. Crowded areas

The crowded areas represent an area with an increased number of challenges, which could be a public square or an area on a pedestrian street, where some entertainment unfolds. The common factor is that there are large number of pedestrians in a relatively small area, which complicates the traversability analysis tremendously.

The idea of crowded areas in the environment originated from a currently missing feature in the ML-Agents toolkit.
The initial idea was to have just static and dynamic obstacles present in the environment, yet having some areas with a higher concentration of dynamic obstacles (sort of equivalent to the crowded areas). Within this environment was at least two robotic agents meant to be deployed, a standard robotic agent with any of the configurations used in this study and a hierarchical robotic agent. The use of a hierarchical agent is inspired by [Yen and Hickey, 2004], and the study was intended to investigate the benefit of the hierarchical structure compared to a regular agent with the use of DRL, instead of RL as in [Yen and Hickey, 2004].
Unfortunately, it is currently not possible to training two agents, in the way needed by

the implementation[3] of [Yen and Hickey, 2004].

The idea of including crowded areas arose as an alternative. The idea is that if the high-level agent can learn to avoid the crowded areas, the need for the low-level agent decreases. Aspects of the initial idea could serve interesting applications for future work.

The robotic agent can pass through the crowded area, but a significant amount of additional effort is required[4], potentially making it beneficial to avoid interaction.

The robotic agent is penalised stepwise with the density of the entered area, representing the increased cost, either computationally or stepwise, of passing through. In this way, are the density used to shape the reward function, in such a way that the robotic agent is incentivised to go around if possible.

The crowded area is represented as a circle, and the size, as well as the density, of the area is two parameters which determines the complexity of the overall environment.

The attainable density values are based on empirical population density estimates, per square meter, for 2019 from London, see table 3.1 and figure 3.3 & 3.4.

The empirical data is used because it brings contrast to all the simulated data in this study. It is not as such the actual levels of the densities that shape the reward function, but the fact that they are significantly different, in between and to the regular step penalty.

The actual value of an area is randomly drawn from the list of possible values, and the density as well as the location of the area is changed every time a new episode begins.

Table 3.1.: Population densities, London.

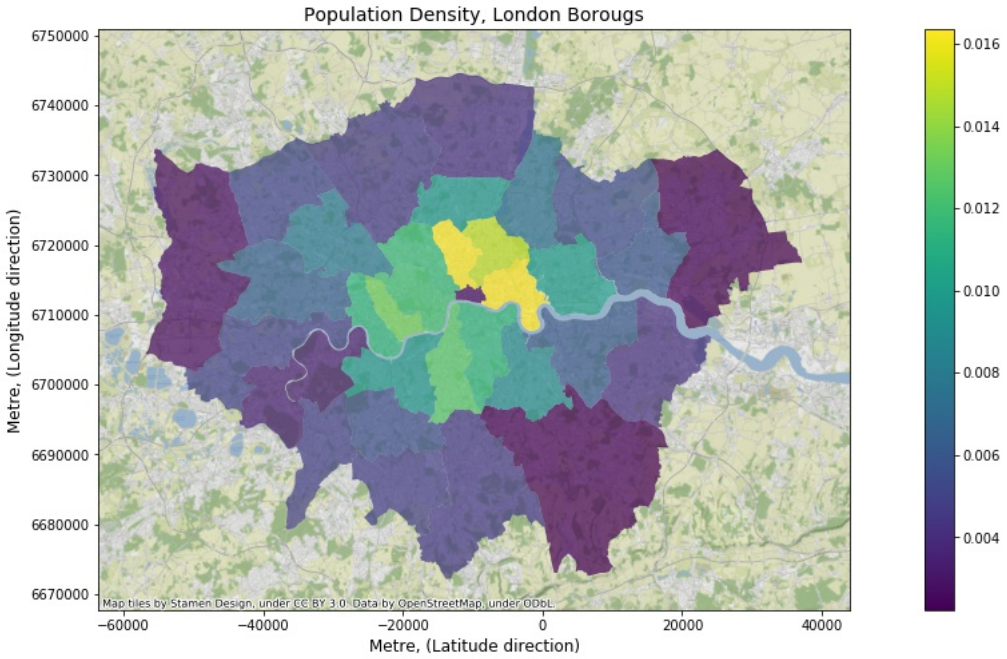
Area	Density
Ward, Maximum	0.0289
Boroughs, Maximum	0.0164
Overall	0.0058

The loss of data in figure 3.4 occurs because the population density data comes without boundaries, and the data needs to be merged onto a shapefile containing the boundaries.

[3]The actions of the first agent (high-level agent) is input to the next agent (low-level), which chooses an action. The high-level agent rewards the low-level agent, based on the low-level agent' ability to follow the direction of the action specified by high-level agent.

[4]Reflecting the work that would have been done by the low-level agent.

Figure 3.3.: Population density, Boroughs, London



"Contains National Statistics data © Crown copyright and database right [2015]" and "Contains Ordnance Survey data © Crown copyright and database right [2015]"

The population-density csv file and the boundaries shapefile has two columns in common, on which the merge can be performed, namely the name of the ward or the GSS code. The similarity of the two files, based on the two columns are around 85% and 71% respectively[5]. Furthermore, a combination of the two columns increases the similarity from 85.2% to 85.6%, which isn't that much of an improvement. The loss of wards is accepted, in lack of ideas to improve the similarity.

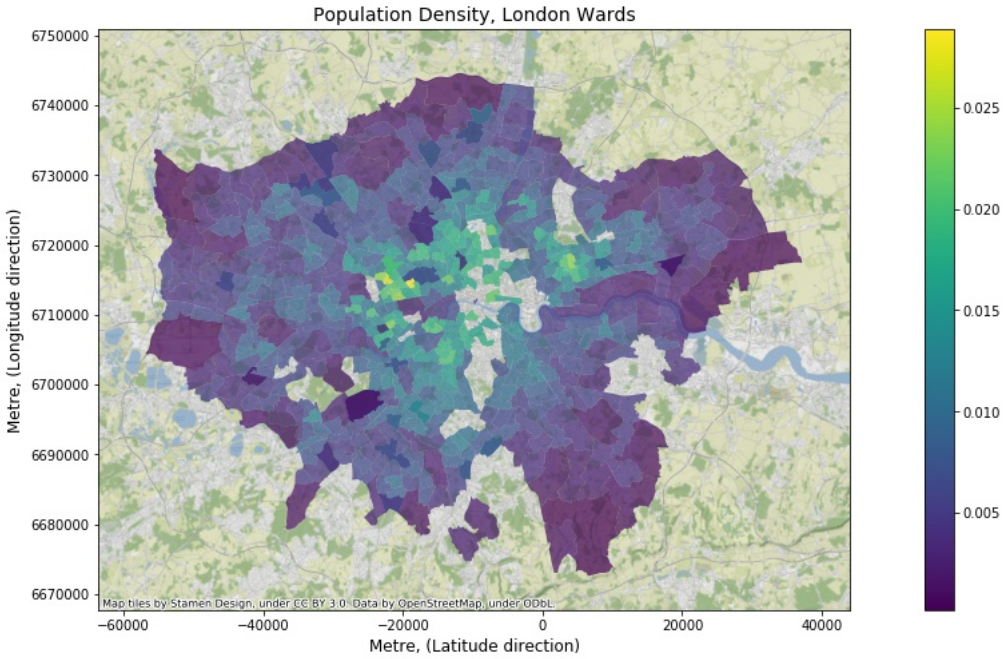
3.3.6. Target

The target is an important element of the environment, as it is the sparse positive reward signal, which the agent searches for. In this study, the target is a cube randomly positioned in the environment at the beginning of every episode[6].

Continuously changing the position of the target increases the likelihood of the learning to generalise to unseen environments. Furthermore, the target is not allowed to be located within a static object, which simulates the idea about locating a position outside

[5]See *Population_density_graphs.ipynb* in the accompanying code file.
[6]See appendix for more technical detail.

Figure 3.4.: Population density, Wards, London



"Contains National Statistics data © Crown copyright and database right [2015]" and "Contains Ordnance Survey data © Crown copyright and database right [2015]"

a building for delivery by the ADR.

As irrelevant as the geometry of the target is, as relevant is the size, relative to the environment. The relative size of the target affects how easily the target is located and can be thought of as the difference in locating a building on a street compared to a specific brick on a specific building. The size of the target is fixed here, at a relative size of 3.75%[7], chosen arbitrary and the effect of the relative size of the target could be a topic for future investigation.

[7]Equivalent to a size of 3 meters.

4. Analysis

Having outlined the methodology of the study, it is time to explore the environment, address the challenges and hopefully solve the environment.
When is the environment solved? The environment is solved when the agent can cope with the different type of challenges that the environment processes. As mentioned in section 3.3, the environment of this study processes four main challenges; difficult areas, static obstacles, crowded areas and dynamic obstacles.

4.1. The look of learning

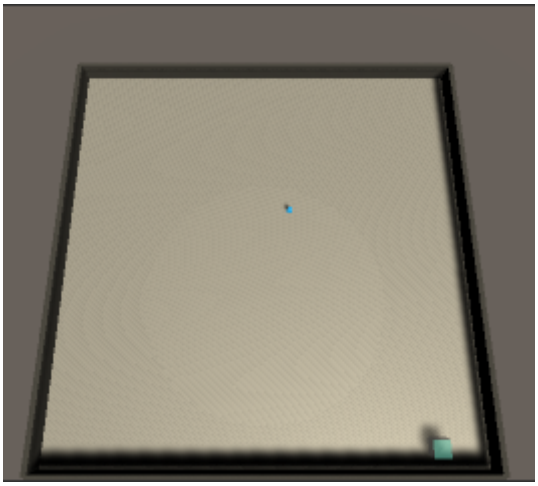
Learning can be expressed in many ways [Juliani et al., 2018c], however, the most intuitive statistics is the ACR and the ALE, examples are seen from figure 4.2.

The ACR curve indicates how well the agent performs, and it is expected to converge towards the upper bound of the reward, within a single episode, doing learning.
The ALE should converge to the minimum number of steps needed to obtain the reward. If a small negative reward is provided at each step, the ALE and ACR develop almost inversely, because the number of steps bounds the maximum obtainable reward.
Figure 7 provides an example.

The speed of which the convergence takes place is a result of the complexity of the environment and the degree of assistance provided to the agent, as will be clear doing this section.

Figure 4.1 shows the simplest possible environment, without any obstacles. Learning in

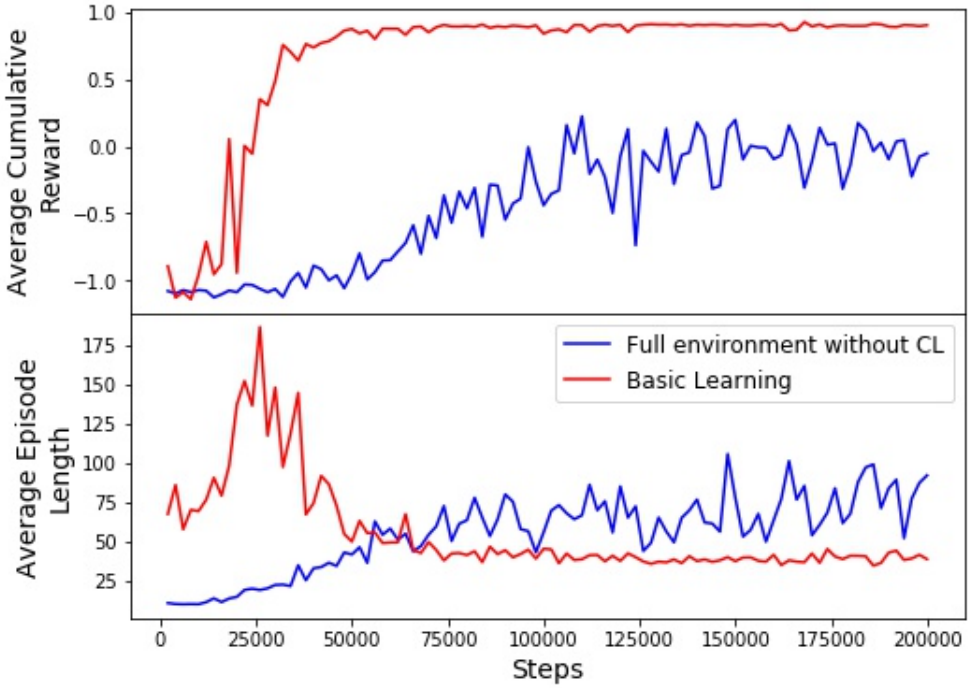
Figure 4.1.: The simplest version of the environment.



The simple environment contains only a target (darker green square) and the agent (light blue).

this environment is straight forward, and this is seen from figure 4.2, by the red curves. The ACR and ALE converges rapidly to their optimal levels, which are obtained after around 75.000 steps in this case.

Figure 4.2.: ACR and ALE for the environment from fig. 3.1 and 4.1



The shapes of ACR and ALE seen in figure 4.2 are ideals, and far from trivial to obtain with the slightest degree of complexity present. The road to meaningful learning requires careful design of the aid provided to the agent to ensure generalisation and so providing

the agent with the ability to handle unseen and challenging environments.
The blue curves are obtained from the environment from figure 3.1, and their shapes are quite different from the red counterparts.
Learning can take place in complex environments, seen from the increasing blue ACR, but it is not guaranteed to be meaningful or stable.
An ACR-curve fluctuating around zero indicates that the actions of the agent is dominated by random actions. It is not possible to say whether the actions of the agent would continue to be random, or if more learning would take place over time.
Figure 4.2 contains just 200.000 steps, which in general is regarded as few, and the purpose of figure 4.2 is to motivate the need for aids, to constitute meaningful and sustainable learning with the ability to generalise to unseen environments.

A final note is that learning can be sensitive to the initialisation of the environment, especially if avoidable objects are added randomly. However, over time, the policy does converge to the optimal policy, see 2.3. The sensitivity just affects the length of *over time*, and some of ways to guide learning in 4.2 significantly reduce the sensitivity.

4.2. Learning to learn

Depending on the complexity of environment, it is sometimes necessary to aid the learning process of an agent. Guiding the agent doing the learning process is necessary if the complexity of the environment fosters a sub-optimal policy, because it is too difficult to learn the optimal policy.
A brief example on this; Imagine an environment consisting of static obstacles and pedestrians. If the probability of colliding with a pedestrian is too high, because the pedestrian density of the environment is high, it could be suboptimal for the agent to bump into a static obstacle as fast as possible. The probability of locating the target does naturally correlate inversely with the probability of colliding with a pedestrian. The low locating-target-probability implies the need for taking a lot of steps.
The suboptimal policy would minimise the number of steps taken, by bumping into the

wall as fast a possible, receiving the associated penalisation, which would seem inevitable for the agent anyway given the high-pedestrian-collision-probability.

4.2.1. Curriculum Learning

The first step towards improving learning is to introduce a curriculum, for the agent to learn from, to gradually add complexity to the environment, as the agent learn to cope with novel challenges.

The use of CL for machine learning algorithms was formalised by [Bengio et al., 2009], and they specifically studied the use of CL on deep nets, with their experiments showing significant improvements in terms of generalisation. [Mirowski et al., 2018] used CL successfully for navigation using DRL, and reported great generalisation properties of their work. How to optimally design a curriculum is almost a study worth itself, and the curriculum used here is designed, such that it directly addresses three of the four main challenges equally, namely the static obstacles, the crowded areas and the dynamic obstacles. Specifying curriculums in Unity is done in a json file, seen in figure 4.3. For a discussion of each of the parameters in the curriculum, see [Juliani et al., 2018b].

Figure 4.3.: The curriculum of this study

```
{
    "measure" : "reward",
    "thresholds" : [0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7,0.7],
    "min_lesson_length": 125,
    "signal_smoothing" : true,
    "parameters" :
    {
        "NumberOfAreas" : [4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0],
        "NumberOfMovingSensors" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.0, 5.0, 5.0],
        "NumberOfSensorClouds" : [0.0, 0.0, 0.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0],
        "Radius": [0.0, 0.0, 0.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0],
        "ObstaclesToAdd" : [1.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    }
}
```

The parameters of the CL are specified through a dictionary, and the keys of the dictionary referrers to the reset parameters[1] of the academy. The dictionary contains at least one

[1]See appendix

variable for each of the three types of obstacles, along with an option to specify the number of training areas, allowing for parallelisation of the training. The latter is described in more detail in section 4.2.4.

The curriculum is designed such that the obstacles are introduced relative to their degree of complexity. The first challenge is the static obstacles, then the semi-static crowded areas and finally the dynamic pedestrians. Each type of obstacle is introduced over three lessons, and the same threshold is required.
The threshold should be set with two variables of the environment in mind, namely the AEL and the *step penalty* (SP), as well as the value of the *minimum number of episodes* (MNE). The AEL and SP form an upper bound on the level of obtainable ACR, and the relationship between the ACR and the threshold gets stronger the higher the value of MNE.
If the positive reward is $+1$, the AEL is 200 and SP is -0.0005, then the upper bound of the ACR is $1 - (200 \cdot -0.0005) = 0.9$[2]. Furthermore, if one training run consist of 200 episodes, and the MNE is chosen to be 200, then the threshold is equivalent to ACR. Setting the threshold and MNE too high implies no progress and setting them too low can result in unstable progress.
Signal smoothing can be enabled to increase the robustness of the signal.

A common misconception when working with reinforcement learning is the amount of data needed to sustain meaningful learning, the planning fallacy as of [Irpan, 2018]. Which means that the MNE and the threshold should be chosen at a level, which ensures stable learning which generalises.

CL is a necessity in this study, for the four different robotic agents to sustain any meaningful learning, within a reasonable time. The following section outlines the performance of each of the four agents, using just CL.

[2]Assuming the SP is the only penalty received.

4.2.2. Exploring the Envrionment under CL

This section outlines the first set of results obtained on all four agents, which serves as a basis for evaluating the effect of the sections to come.

Common for this section, section 4.3 and 4.4, is the results that is examined.
To begin with, standard metrics provided via TensorBoard are examined. The metrics are the ACR, AEL and the changes in lessons, and described at bit in section 4.1.
The second set of data examined is custom data, generated by the agents as they learn the environment, and the data is mainly collision data. The collision data is data on collisions with the target, crowded areas, difficult areas and the pedestrians, which enables comparison of the efficiency of the different agents.

Figure C.2 & 4.4 presents the results of $500,000$ training steps using CL. The means and standard deviations of are calculated after $100,000$ steps, excluding the burn-in period which is harder to justify belongs to the same distribution as the following observations.

Table 4.1.: Baseline Learning

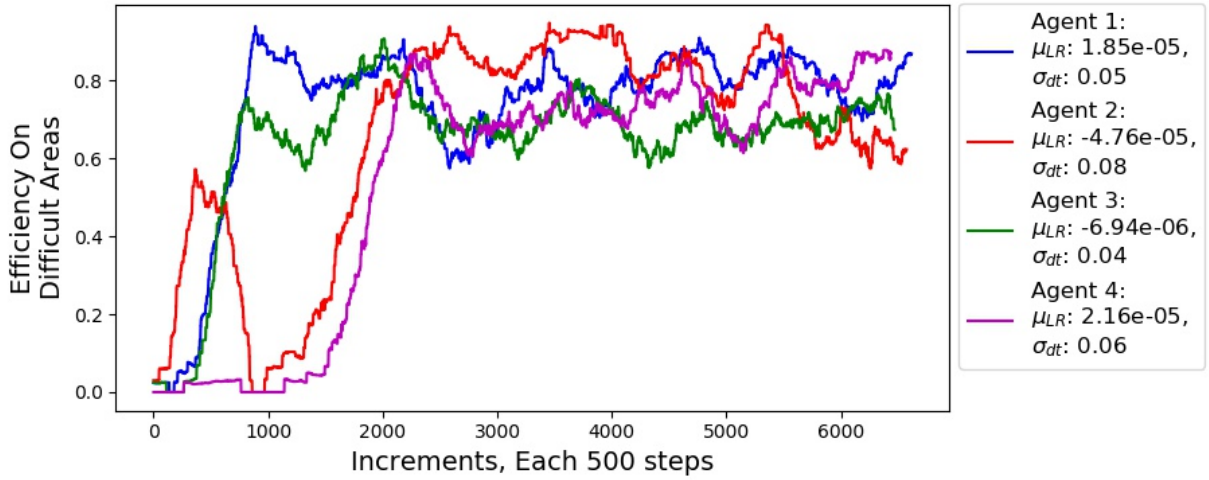
Level	Agent 1	Agent 2	Agent 3	Agent 4
	Learning Rates			
$\mathbf{Mean_w}$	$\mathbf{1.3 \cdot 10^{-2}}$	$\mathbf{1.3 \cdot 10^{-2}}$	$\mathbf{1.4 \cdot 10^{-2}}$	$\mathbf{1.1 \cdot 10^{-2}}$
SO	$2.3 \cdot 10^{-2}$	$2.2 \cdot 10^{-2}$	$2.7 \cdot 10^{-2}$	$1.7 \cdot 10^{-2}$
CA	$1.7 \cdot 10^{-3}$	$-1.7 \cdot 10^{-4}$	$3.2 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$
DO	NaN	NaN	NaN	NaN
	Standard Deviations			
$\mathbf{Mean_w}$	**0.22**	**0.36**	**0.25**	**0.27**
SO	0.22	0.28	0.24	0.17
CA	0.20	0.59	0.25	0.34
DO	NaN	NaN	NaN	NaN
	Weights			
SO	0.51	0.78	0.56	0.54
CA	0.49	0.23	0.44	0.46
DO	NaN	NaN	NaN	NaN

SO: Static Obstacles, CA: Crowded Areas, DO: Dynamic Obstacles

One should be cautious about interpreting too much on the initial results, as they are sensitive to initialisation, primarily because the episodes are unbounded, and they are

provided to serve as a baseline. Figure C.2 shows the ACR, AEL and changes in lessons for all four agents. Looking at the tables next to the ACR and AEL curves indicates that agent 1 is the most stable, agent 4 is the most efficient[3] and that there isn't much difference between agent 2 and 3.

Figure 4.4.: Dealing With Difficult Areas



The means and standard deviations of are calculated after 2,000 increments.

Figure 4.4 shows little difference at this stage, as all average efficiencies lies within one standard deviation from each other. However, this set the stage for investigating the effect of different aids in the training face.

4.2.3. Maximum Steps

The motivation of invoking a maximum number of steps for the agent to take, is that above a certain threshold, destructive paths is too likely to occur. Destructive paths occur because of too much complexity, which implies that unwounded behaviour becomes optimal at time.

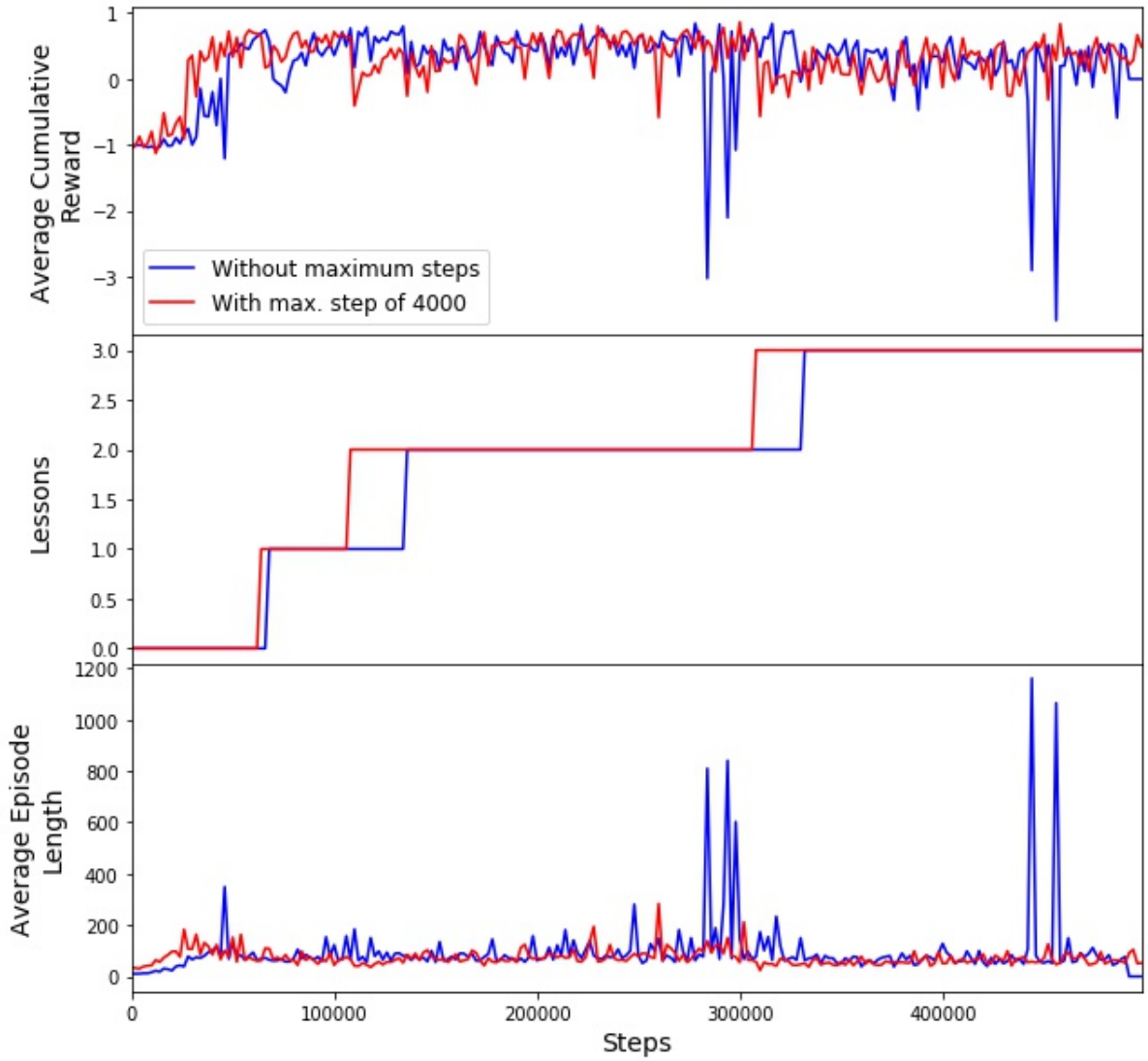
An example of this is that the agent runs in circles, because the target is located in area which is either associated with too high a probability of colliding with obstacles or unexplored, and so uncertain, and because of the size of step penalty, that implies running in circles is more attractive at the time.

[3]In terms of locating the target, seen from the fact that agent 4 scores highest average ACR.

Note, the threshold should be chosen such that it doesn't circumvents the agent for moving around the entire environment.

An example of the effect can be seen from figure 4.5 and 4.6. Figure 4.5 shows the ACR, AEL and the changes in lessons of two runs of CL-based training, in an environment with static and dynamic obstacles (pedestrians). The blue run does not have a maximum number of steps specified, which the red have. Figure 4.6 shows the actual number of steps for each episode of blue run, with one episode taking around $35,000$ steps. Table 4.2 shows how many observations lies above certain thresholds. The threshold used on the red run, in figure 4.5, is $max_{steps} = 4.000$, which from table 4.2, is equivalent to excluding 0.5% of the total observations.

Figure 4.5.: The Effect of Restricting the number of steps



Excluding such a tiny fraction of the observations results in a lot more stable training, by reducing the skewness of the steps-distribution[4].

Table 4.2.: Fraction of observations lost with maximum steps invoked

$\mathbf{Max_{steps}}$	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Share	0.729	0.534	0.371	0.252	0.146	0.092	0.066	0.051	0.042	0.035
$\mathbf{Max_{steps}}$	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**	**8000**	**9000**	**10000**
Share	0.035	0.013	0.008	0.005	0.003	0.002	0.002	0.001	0.001	0.001

Figure 4.6 shows two important findings. Each graph, in figure 4.6, contains the steps as points and a tiny bar, if the target of an episode where located in a difficult area and the target where reached.

The first important finding is, that the restricted agent (red) reaches targets in the difficult areas continuously throughout training. This indicates that restricting the number of steps to 4000 does not prevent the agent for exploring even the toughest parts of the environment.

The second important finding is, that the restricted agent seems to learn the difficult areas significantly quicker than its unrestricted fellow.

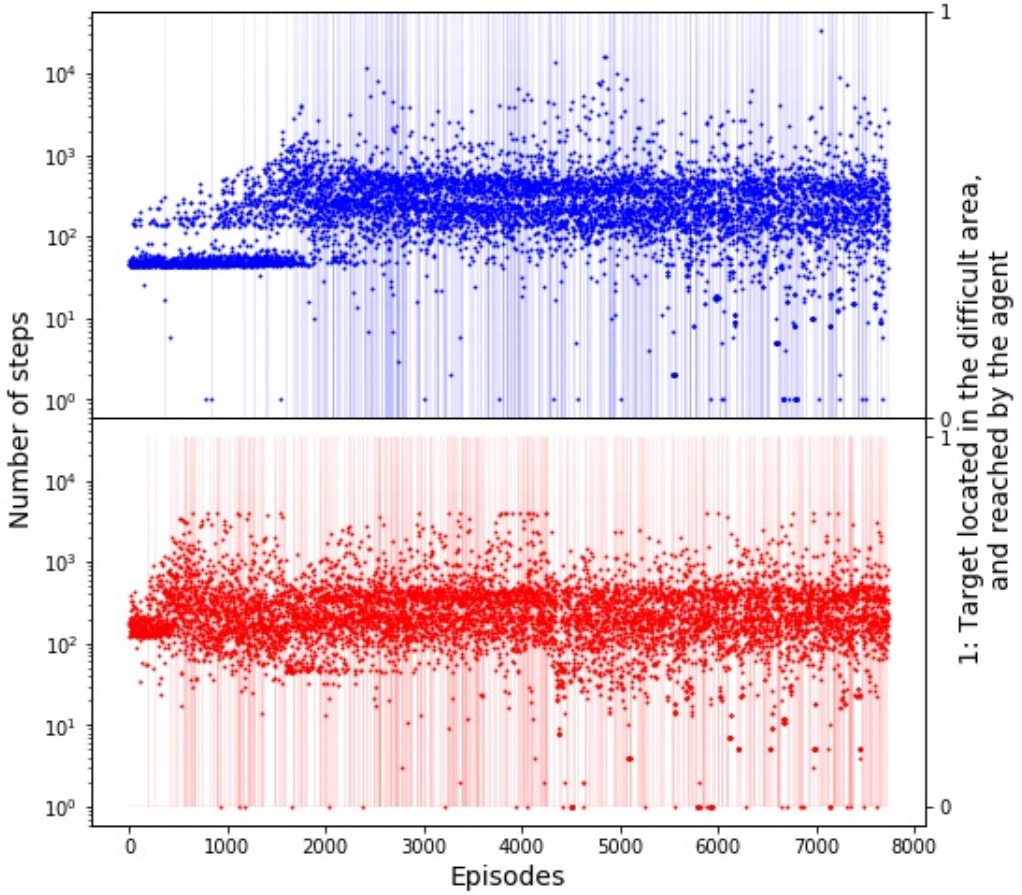
The last point is derived from the concentration of bars in the two graphs. The blue case takes around 2000 episodes before the concentration stabilise, where it, at worst, happens around 500 episodes for the red case. A stable concentration of bars indicates that learning, to locate the target in the difficult areas, has taken place.

Because the targets are randomly placed around the environment, we did expect to see roughly the same number of episodes, with the *target located in the difficult areas* (TLDA), and this is confirmed from table 4.3.

Table 4.3 reveals an interesting observation, namely that the efficiency, in terms of episodes with TLDA resulting in the target being reached, does not appear to differ, taking the differences in TLDA-shares into consideration. However, the steps needed to reach the target is on average 15% lower for the bounded agent, which results in better performance.

[4]See appendix

Figure 4.6.: Number of Steps Within Each Episode



Bounding the steps of the agent appears to result in the agent learning the environment quicker, and taking smarter actions, which over the longer run could materialise in overall improved efficiency because the quality of experience of the agent increases.

The take-away from this section is that avoiding potentially disastrous paths does influence learning, and going forward, a $max_{steps} = 4.000$ is used.

4.2.4. Shared Experience

Parallelised training is well-known is be beneficial in DL and DRL [Mnih et al., 2013, LeCun et al., 2015, Silver et al., 2016, Teng et al., 2019], to cope with the amount of data needed, and manipulating the training time, to obtain significant results. The main objective of parallelisation is to reduce training time, and where training of deep nets with hundreds of millions of weights and billions of connections between units took weeks

Table 4.3.: Learning the difficult areas

	Category	True	False	Share
No max. steps	TLDA	6765	722	0.096
	GR	349	373	0.517
Max. steps	TLDA	6914	822	0.106
	GR	372	450	0.547

two years ago, advances in parallelisation have reduced training time to a few hours [LeCun et al., 2015].

Part of these advances in parallelisation comes from novel algorithms, which benefit from parallelisation other than reduced training time. These novel algorithms incorporate experience of the agent to update the function approximation in question, thereby reducing the amount of data needed, and this is exactly what PPO does.

Parallelisation, at least in Unity, works by running concurrent training environments, either internally or externally. External refers to running multiple applications of the same environment, often used when training on GPU's via cloud computing. Internal refers to having multiple environments, with the agents being linked to the same brain, within one application, see figure 4.7. Internal concurrent training is a way to utilise parallelisation when training on a CPU, and training using GPU's isn't a possibility [Teng, 2019].

Running concurrent training environments, externally as well as internally, fills the same experience buffer with prior observations. The usual benefit is that training accelerate, because the share of favourable prior observations increases faster with more agents to harvest. However, it can of cause go the other way, if the environment contains too much complexity at the time being. Furthermore, if nothing prevents the agents from wandering around to infinity, a bad spiral unfolds. This spiral is powered by two events; firstly, the agents never benefit from sharing experience because they rarely finish an episode. Secondly, the shared experience is low quality of the wandering. In this way, sharing becomes a burden. This is referred to as the feedback-loop-effect by [Butcher, 2018][5].

[5]Section *J.P. Morgan has been using reinforcement learning algorithms to place trades, even though this can cause problems.*

Figure 4.7.: Internal concurrent training

Four similar environments prior to initialisation, each with an agent, a target and static obstacles. All the agents are linked to the same brain, thereby gathering experience to the same buffer.

The power in sharing experience is seen from figure 4.8. Figure 4.8 provides a detailed look of the AEL curve, in figure 26[6], from the area bounded by the black line. Figure 4.8 shows that bad experience of one agent often comes alone, i.e. spikes are often apart. The result is that the experience buffer contains a higher share of good experience to base the policy update on, implying an acceleration in learning, seen in figure 26.

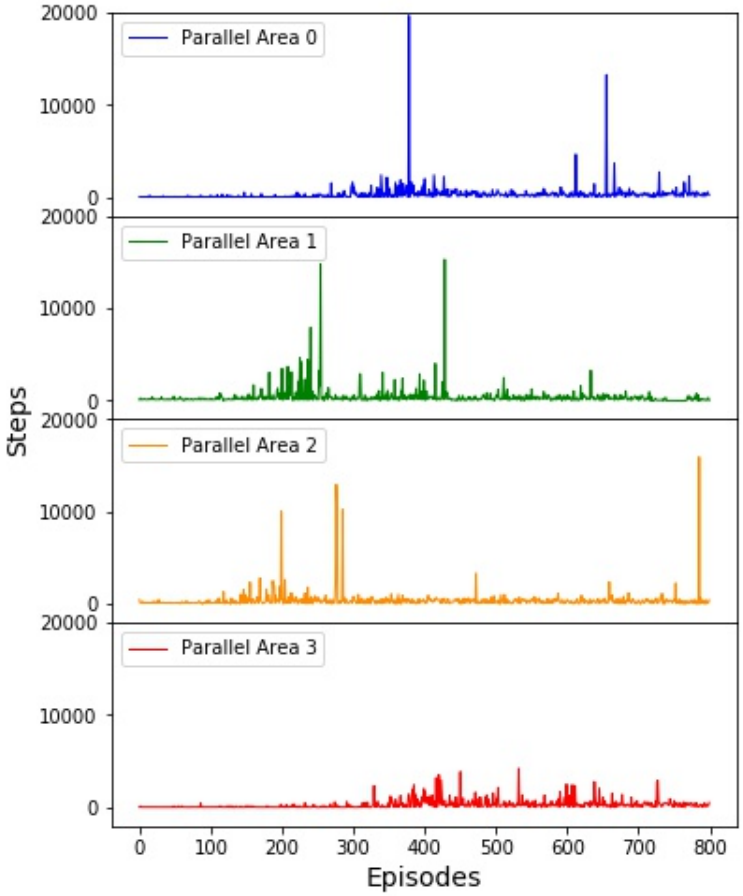
4.2.5. Observation Stacking

Stacking observations provide the agent with a short-term memory, which can be beneficial in dynamic environments where consecutive information enables the agent to better understand the consequences of its action, and thereby better foresee [Juliani, 2018a]. Observation stacking is an alternative to the use of recurrent nets proposed by [Hochreiter and Schmidhu and explored by [Mnih et al., 2015, Borsa et al., 2019], for example to achieve human-level control on 47 ATARI games.

The effect of stacked observations is explored in this study, because recurrent nets have been found not to work well with continuous vector action spaces [Matter et al., 2018], which is used by the agents of this study.

[6]See appendix

Figure 4.8.: Shared Experience

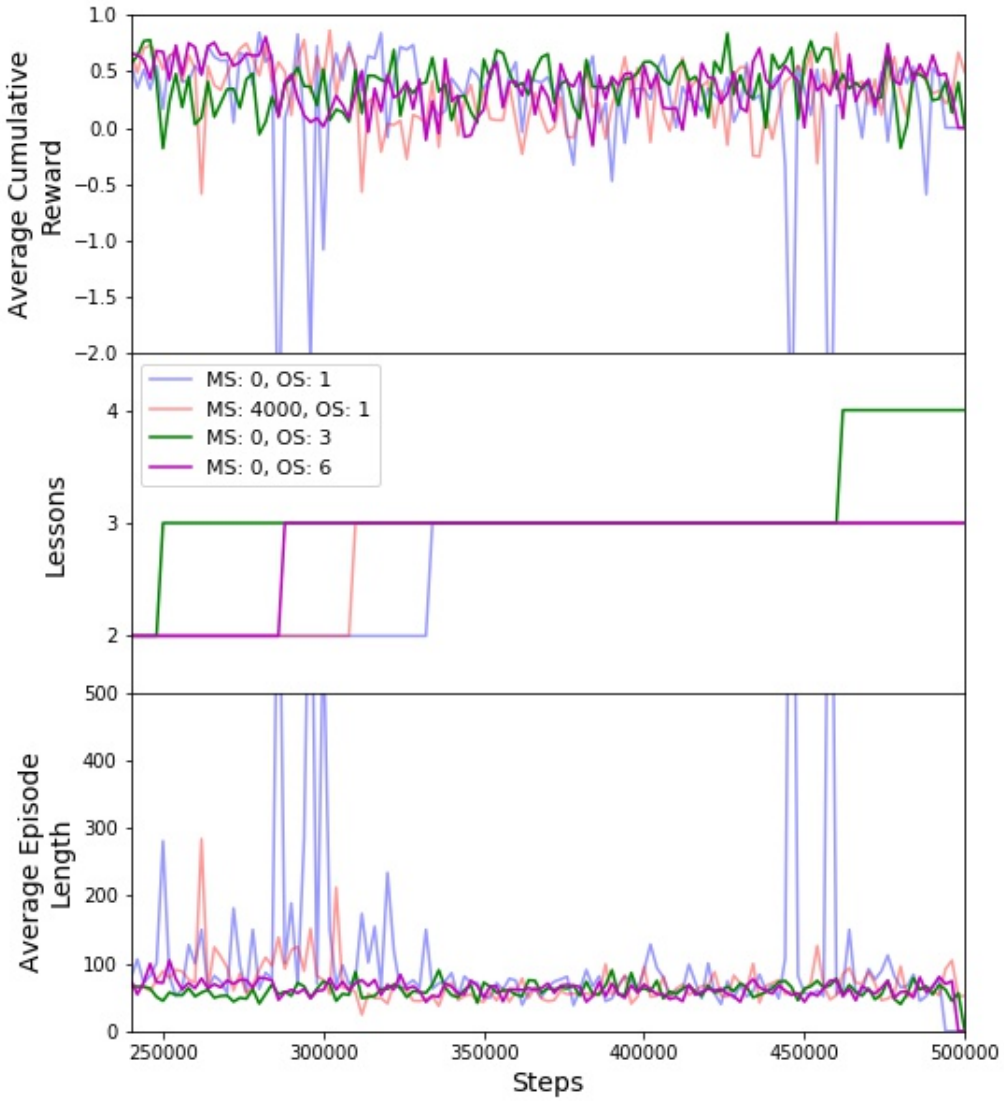


The effect of stacking observations is interesting when the dynamic obstacles are present in the environment, illustrated in figure 4.9. Figure 4.9 compares the performance, in terms of ACR and AEL, of the baseline from figure 4.5 and two unbounded runs with 3 and 6 observations stacked respectively. The comparison starts from the number of steps where the first run progresses to lesson 3, which is where the pedestrians are introduced in the environment.

The take-away from figure 4.9 is the stability of the ACR, but especially the AEL, from the two stacked runs (green and purple).

Another interesting angle for comparison, between the four runs from figure 4.9, is to look at the number of collisions with pedestrians, see table 4.4. Table 4.4 indicates how stacking consecutive observations, can improve the ability to avoid collisions with pedestrians. The green run spends over half of its episodes with pedestrians present and has the lowest collision rate.

Figure 4.9.: The Effect of Observation Stacking



Three ways to improve learning in the agents, as well as their individual effect, has been presented, and the following section evaluates the combined effect.

4.3. Results under certainty

This section outlines the results obtained after 1 million steps using the full set of aids previously described. The results presented in the following is directly comparable with the baseline results, except on the number of steps simulated, which matters for the quality of the policy obtained for each agent.

Figure C.3 shows similar pattern to figure C.2, with agent 1 being the most efficient and

Table 4.4.: Pedestrian collisions

Type	Obs	$Obs_{>=L_3}$	$Share_{>=L_3}$	Collisions	Share
MS: 0, OS: 1	7487	2458	0.33	524	0.21
MS: 4000, OS: 1	7712	3393	0.44	893	0.26
MS: 0, OS: 3	8375	4393	0.52	841	0.19
MS: 0, OS: 6	8463	3698	0.44	868	0.24

stable.

Agent 2 and 3 follows closely and agent 4, the agent with visual input, does not seem to benefit from the increased information.

Table 4.5.: Learning Rates Under Certainty

Level	Agent 1	Agent 2	Agent 3	Agent 4
	Learning Rates			
$\mathbf{Mean_w}$	$\mathbf{6.1 \cdot 10^{-3}}$	$\mathbf{6.2 \cdot 10^{-3}}$	$\mathbf{5.7 \cdot 10^{-3}}$	$\mathbf{5.9 \cdot 10^{-3}}$
SO	$5.1 \cdot 10^{-2}$	$5.8 \cdot 10^{-2}$	$5.7 \cdot 10^{-2}$	$5.2 \cdot 10^{-2}$
CA	$-3.2 \cdot 10^{-3}$	$6.2 \cdot 10^{-4}$	$-6.4 \cdot 10^{-3}$	$5.7 \cdot 10^{-3}$
DO	$6.4 \cdot 10^{-4}$	$2.2 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$	$4.9 \cdot 10^{-3}$
	Standard Deviations			
$\mathbf{Mean_w}$	**0.13**	**0.12**	**0.12**	**0.12**
SO	0.10	0.11	0.12	0.14
CA	0.10	0.11	0.10	0.12
DO	0.14	0.13	0.13	0.14
	Weights			
SO	0.08	0.07	0.09	0.10
CA	0.10	0.23	0.55	0.26
DO	0.82	0.70	0.36	0.64

SO: Static Obstacles, CA: Crowded Areas, DO: Dynamic Obstacles

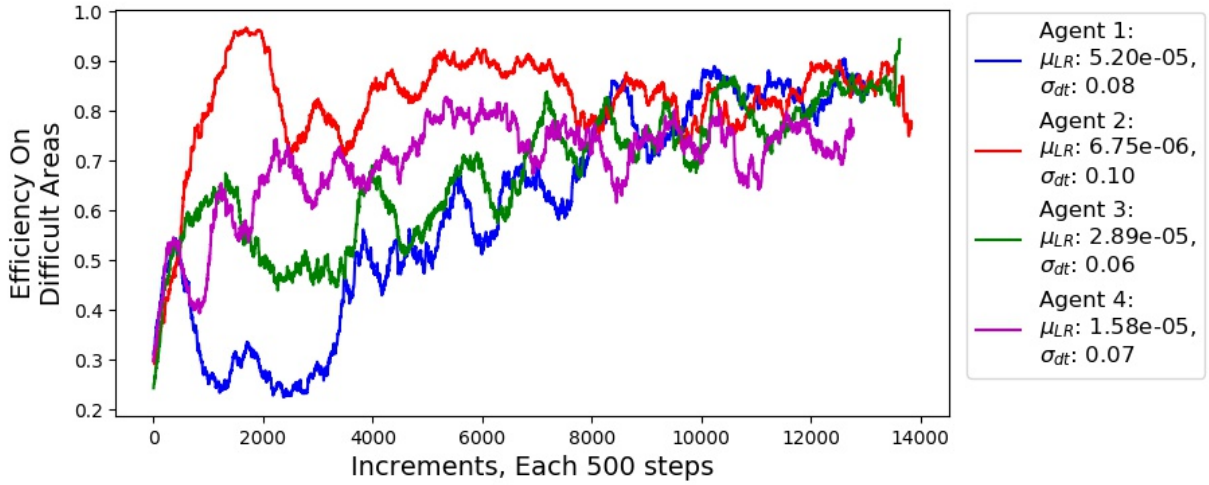
Figure 4.10 shows the ability to deal with the difficult areas, and here is little more dispersion the series in between, compared to figure 4.4. Agent 3 and 4 seems to have a harder time reaching the targets in the difficult areas, which is interesting because both have the distance to the target in their information set .

Table 4.6 shows the shares of collisions, with crowded areas and pedestrians. The numbers in parenthesis shows the shares of episodes, in which the agent has been exposed to the given obstacle. All shares are calculated from the episodes, and not from the steps. Evaluating shares makes comparison straighter, agents in between as well as the baseline

case and *certain* case. Table 4.6 shows significant differences the agents in between, in their ability to handle crowded areas. Most importantly, table 4.6 indicates that the agents learn to avoid the crowded areas. One way to see this, is by comparing the collision share of agent 1 and agent 4.

The collision share of Agent 1 are lower than the one of agent 4, with agent 1 having been exposed more to the crowded areas. More exposure should, all else equal, imply higher risk of collision, unless learning has occurred – this is discussed more in section 5.1.

Figure 4.10.: Dealing With Difficult Areas Under Certainty



The curves are averages from the four parallel areas, and the standard deviations are calculated on detrended series.

With the core of the results obtained under certainty outlined, it is time to look at the results obtained under uncertainty, which is the focus of the next section.

Table 4.6.: Shares of Collisions

	Crowded Areas			Pedestrians	
Agent	Baseline	Certain	Uncertain	Certain	Uncertain
1	25.1% (42.3%)	14.4% (88.0%)	21.6% (71.7%)	13.4% (77.2%)	10.3% (43.4%)
2	36.9% (15.6%)	17.4% (87.7%)	16.7% (87.0%)	13.7% (65.3%)	13.7% (75.4%)
3	24.6% (43.1%)	17.3% (79.3%)	18.5% (87.9%)	9.8% (32.4%)	8.9% (59.1%)
4	32.3% (27.2%)	22.6% (81.4%)	12.0% (62.7%)	14.5% (57.7%)	y%

Table 4.7.: Learning Rates Under Uncertainty

Level	Agent 1	Agent 2	Agent 3	Agent 4
	Learning Rates			
Mean$_w$	$\mathbf{5.9 \cdot 10^{-3}}$	$\mathbf{6.0 \cdot 10^{-3}}$	$\mathbf{4.4 \cdot 10^{-3}}$	$\mathbf{2.3 \cdot 10^{-2}}$
SO	$6.1 \cdot 10^{-2}$	$4.7 \cdot 10^{-2}$	$3.4 \cdot 10^{-2}$	$6.4 \cdot 10^{-2}$
CA	$-3.9 \cdot 10^{-7}$	$1.4 \cdot 10^{-3}$	$-5.5 \cdot 10^{-2}$	$3.1 \cdot 10^{-3}$
DO	$1.2 \cdot 10^{-3}$	$9.9 \cdot 10^{-4}$	$1.1 \cdot 10^{-2}$	$6.3 \cdot 10^{-4}$
	Standard Deviations			
Mean$_w$	**0.14**	**0.13**	**0.13**	**0.12**
SO	0.14	0.13	0.10	0.08
CA	0.14	0.09	0.11	0.08
DO	0.14	0.13	0.14	0.13
	Weights			
SO	0.16	0.08	0.07	0.09
CA	0.35	0.11	0.28	0.22
DO	0.49	0.81	0.65	0.69

SO: Static Obstacles, CA: Crowded Areas, DO: Dynamic Obstacles

4.4. Results under uncertainty

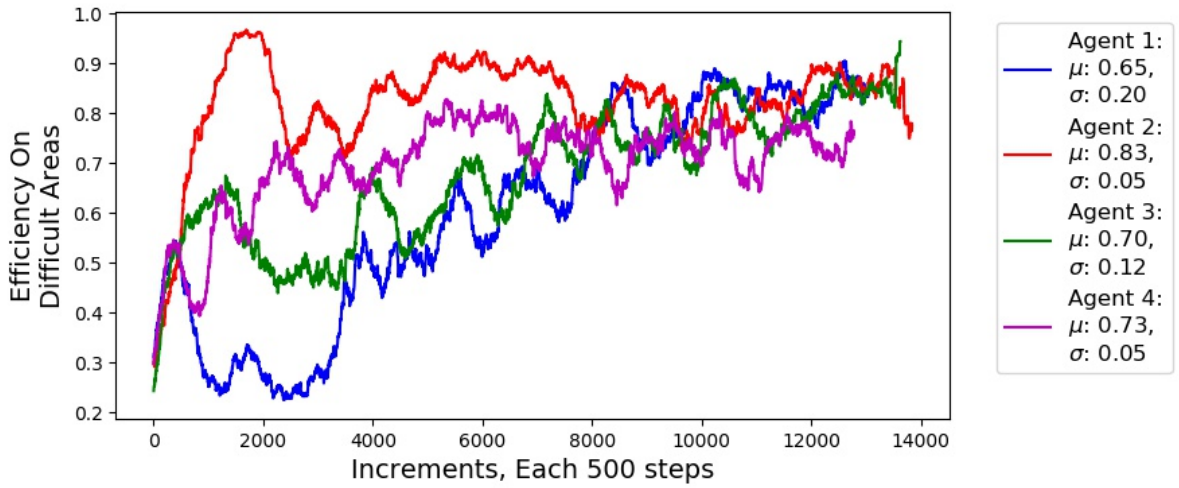
Hej

hej

Table 4.8.: Training Time

Agent	Baseline	Certain	Uncertain
1	42 min.	124 min.	138 min.
2	43 min.	129 min.	133 min.
3	44 min.	135 min.	148 min.
4	125 min.	667 min.	575 min.

Figure 4.11.: Dealing With Difficult Areas Under Uncertainty



The curves are averages from the four parallel areas, and the standard deviations are calculated on detrended series.

5. Policy Evaluation

5.1. Dicussion

Table 5.1.: Learning Rates - Accumulated Cumulative Reward

	Accumulated Cumulative Reward		
Agent	Baseline	Certain	Uncertain
1	$\mathbf{1.3 \cdot 10^{-2}}$ **(0.22)**	$\mathbf{6.1 \cdot 10^{-3}}$ **(0.13)**	$\mathbf{5.9 \cdot 10^{-3}}$ **(0.14)**
SO	$2.3 \cdot 10^{-2}$ (0.22)	$5.1 \cdot 10^{-2}$ (0.10)	$6.1 \cdot 10^{-2}$ (0.14)
CA	$1.7 \cdot 10^{-3}$ (0.20)	$-3.2 \cdot 10^{-3}$ (0.10)	$-3.9 \cdot 10^{-7}$ (0.14)
DO	NaN (NaN)	$6.4 \cdot 10^{-4}$ (0.14)	$1.2 \cdot 10^{-3}$ (0.14)
2	$\mathbf{1.3 \cdot 10^{-2}}$ **(0.36)**	$\mathbf{6.2 \cdot 10^{-3}}$ **(0.12)**	$\mathbf{6.0 \cdot 10^{-3}}$ **(0.13)**
SO	$2.2 \cdot 10^{-2}$ (0.28)	$5.8 \cdot 10^{-2}$ (0.11)	$4.7 \cdot 10^{-2}$ (0.13)
CA	$-1.7 \cdot 10^{-4}$ (0.59)	$6.2 \cdot 10^{-4}$ (0.11)	$1.4 \cdot 10^{-3}$ (0.09)
DO	NaN (NaN)	$2.2 \cdot 10^{-3}$ (0.13)	$9.9 \cdot 10^{-4}$ (0.13)
3	$\mathbf{1.4 \cdot 10^{-2}}$ **(0.25)**	$\mathbf{5.7 \cdot 10^{-3}}$ **(0.12)**	$\mathbf{4.4 \cdot 10^{-3}}$ **(0.13)**
SO	$2.7 \cdot 10^{-2}$ (0.24)	$5.7 \cdot 10^{-2}$ (0.12)	$3.4 \cdot 10^{-2}$ (0.10)
CA	$3.2 \cdot 10^{-3}$ (0.25)	$-6.4 \cdot 10^{-3}$ (0.10)	$-5.5 \cdot 10^{-2}$ (0.11)
DO	NaN (NaN)	$1.0 \cdot 10^{-3}$ (0.13)	$1.1 \cdot 10^{-2}$ (0.14)
4	$\mathbf{1.1 \cdot 10^{-2}}$ **(0.27)**	$\mathbf{5.9 \cdot 10^{-3}}$ **(0.12)**	$\mathbf{2.3 \cdot 10^{-2}}$ **(0.10)**
SO	$1.7 \cdot 10^{-2}$ (0.17)	$5.2 \cdot 10^{-2}$ (0.14)	$6.4 \cdot 10^{-2}$ (0.08)
CA	$1.6 \cdot 10^{-3}$ (0.34)	$5.7 \cdot 10^{-3}$ (0.12)	$3.1 \cdot 10^{-3}$ (0.08)
DO	NaN (NaN)	$4.9 \cdot 10^{-3}$ (0.14)	$6.3 \cdot 10^{-4}$ (0.13)

5.2. Future Work

5.3. Conclusion

A. Technical Details

The first part of this appendix is devoted to the technical aspect of the study, implying a full treatment of the elements of the environment and their configuration.

A.1. Developing digital environments in Unity

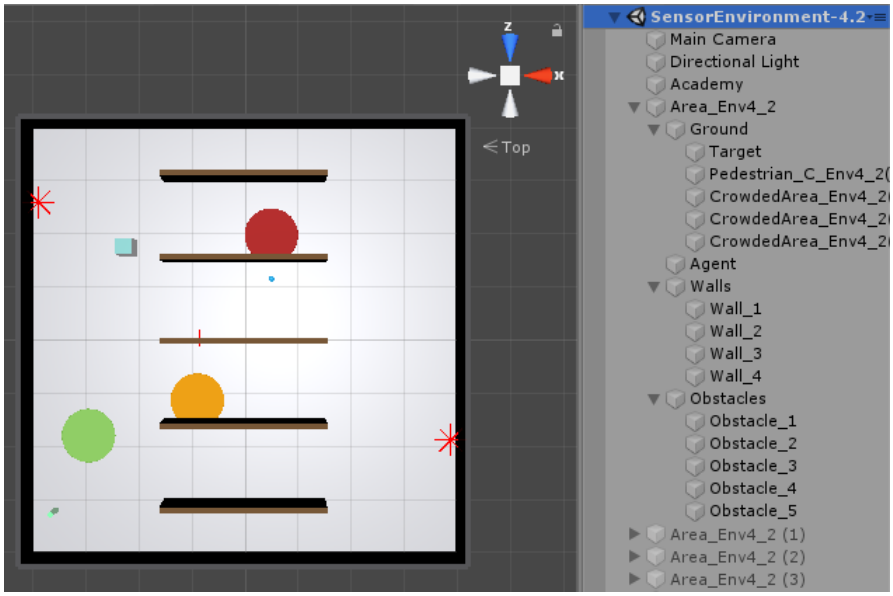
Every application build with Unity is made of *Scenes*, *GameObject's*, *components* and *scripts*. An application can contain an arbitrary number of scenes, and the application shown in figure A.1contains one, namely *SensorEnvironment-4.2*. Every scene contains GameObject's, in which components and scripts are attached, to sustain any form of behaviour imaginable. Components enables the use of all the built-in functionalities in the Unity Engine, and scripts provides the researcher with the option to take full control.

Developing an environment, formally known as a *Scene* in Unity, to facilitate the possibilities within the ML-agents toolkit requires some standard objects; an actual *environment* to explore, an *academy*, an *agent* and a *target*. Figure A.1 is an example of how everything could be organised within a scene.

One thing to note from figure A.1 is that all necessary elements are contained in a prefab named *Area_EnvX*[1]. Prefabs are pre-defined GameObject's, and the use of prefabs are a neat way of altering similar objects simultaneously. For the environment in question, containing all necessary elements in a prefab, is a way to utilise a parallelised set-up, allowing for faster training. The following describes key components of the entire environment, divided into the four categories made up by the basic objects required by the

[1]X referrers to the current version of the implementation.

Figure A.1.: A scene in Unity containing the components of the ML-Agents toolkit



The scene contains all available elements within the environment.

ML-Agents toolkit. All components left out of the following description can be found in the appendix, to ensure reproducibility.

A.1.1. Environment

The walkable area is labelled *ground* in figure A.1, and it serves two important purposes. Firstly, it defines the extend of the area through its scale, seen in figure A.2. The extend of the area is used to ensure that random placing, through scripting, of objects happens within the bounds of the traceable area.

Secondly, it serves as a container for the objects belonging to this training area[2]. Initialising new GameObject's as children of another GameObject is a way to ensure interaction with intended GameObject's. It allows the researcher to write generic scripts and not instances specific scripts, which are in general good practice, and especially desirable when working with parallelised set-ups.

Tags

Every GameObject within the training area is tagged, as seen in the right side of figure A.2. Tags is an elegant way to differentiate GameObject's from each other, especially

[2]Made up of the *Area* prefab.

useful in association with collision detection, collecting observations on the state of the environment and random placing of GameObject's.

Layers

Assigning different GameObject's to different layers is used to either include or exclude certain GameObject's from some sort of detection. This is useful in the two-brain set-up, ensuring that one brain handles avoidance of dynamic obstacles and one brain takes care of the general navigation towards the target.

Static Objects

To the right of the GameObject's name is the ability to mark a GameObject as static, which is used in connection with NavMesh agents. Static GameObject's are part of mesh in which a NavMesh agent can navigate [Unity, 2019].

Geometry of a GameObject

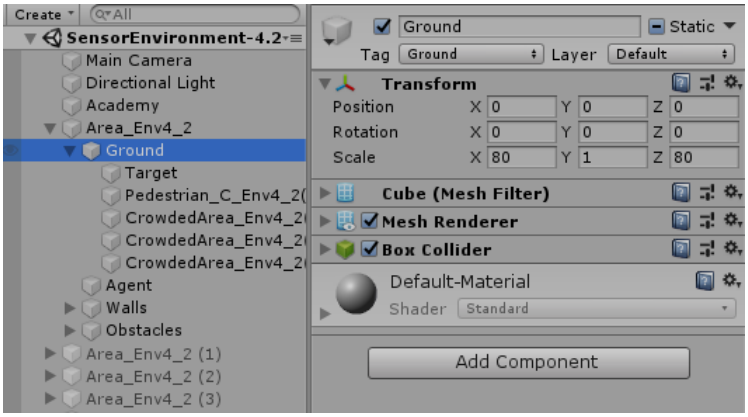
Any object having a shape contains a Mesh filter, defining the geometry of the object, and a Mesh renderer, which ensures rendering of the object at the position specified in the transform component. Figure A.2 shows that the *ground* element is a plane, having a size of $80 \times 80 \times 1$, positioned at $(0, 0)$. The height (size in the y direction) of the *ground* element is not as such important, if it is above 0, to sustain the plane rendering.

Within the bounded *ground*, not necessarily as child objects, is six types of objects placed, two of them elaborated in individual sections below, and the other four are *walls*, *obstacles*, *pedestrians* and *crowded areas*.

A.1.1.1. Walls & Obstacles

The walls as well as obstacles are, as GameObject's, identical to the *ground* GameObject, with a different tag along with the obviously different size and position. The walls and obstacles have different materials, to indicate that they represent something different.

Figure A.2.: The ground object



A.1.1.2. Pedestrians

The pedestrian GameObject is a prefab, which is attached to the academy from where it is initialised. The RL agent reset upon collision with them, as it is unacceptable for ADR's to collide with pedestrians. Figure 16 shows the components attached to the pedestrian prefab, of which four are interesting to elaborate on.

Collider

The collider together with the rigidbody component, is what that enables collision detection between the object and another object, with a collider and rigidbody component attached as well. The settings in the collider is irrelevant, as they are standard settings matching the scale of the GameObject.

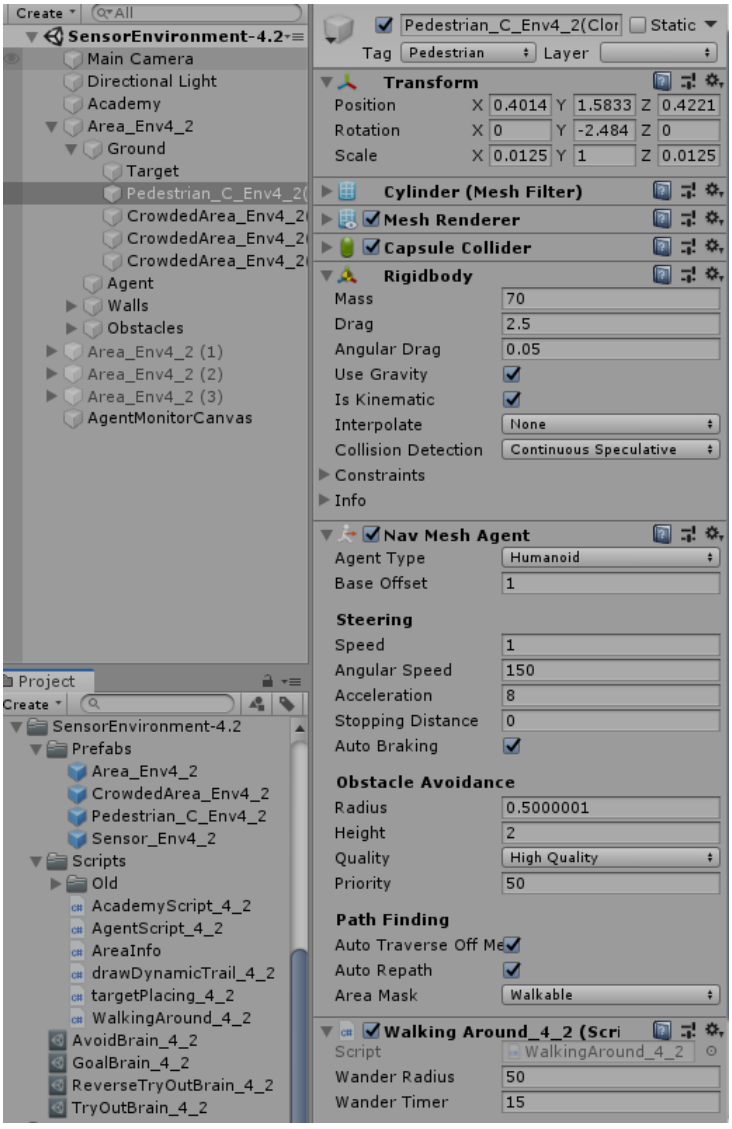
Rigidbody

The rigidbody is the component that enables the physics engine to take control of the movement of the GameObject. The mass of the GameObject is specified in kilograms and is set equal to a reasonable average value for a male.

RB: Drag

The drag is a force working in the opposite direction to the movement of the object, specifying at what pace the movement of the object is decreased. The value for the drag of the pedestrian prefab is calculated using A.1, to ensure realistic behaviour in the simulation, because the default value is zero which is not in accordance with realistic

Figure A.3.: The pedestrian prefab



behaviour.

$$drag = 1/2 \cdot \rho \cdot v^2 \cdot C_d \cdot A \tag{A.1}$$

Where ρ is density (kg/m^3) of the fluid that the object passes through, air in this case here, v is the speed (m/s) at which the object moves, C_d is the drag coefficient (unit less) and A is the cross sectional area related to the movement, which is the area of the object normal to the direction of the movement. Table 5 shows the values used, and the calculated drag.

Table A.1.: Drag

	Value	Unit
Density*, ρ:	1.225	kg/m^2
Speed, v:	1	m/s
Drag Coefficient, C_d**:	1.3	Unitless
Cross Sectional Area, A:	3.142	m^2

*: Density of air
**: Average of human body in upright position, and at the same time the coefficient of a short cylinder.

The *angular drag* is how much a rotation is slowed down, and it is kept at standard value, because the default value is within a realistic order.

The rigidbody is marked as being *kinematic*, which implies that the object isn't influenced by any forces. Why have the rigidbody attached then? Because it ensures better collision properties having both the collider and the rigidbody attached to an object, and all movement is handled by the NavMesh agent component below.

Nav Mesh Agent

The Nav Mesh Agent component is what turns an empty GameObject into a Nav Mesh agent. At least initially, is every parameter herein kept default, except speed and angular speed. The speed of the agent is set equal to $1\,m/s$, the same as the ML agent, which is roughly equal to a speed of $3.5\,km/t$ – the average chilled walking speed. The angular speed is set equal to 150, a bit above the default value of 120, as it is equal to the angular speed used in the ML agent.

Walking around

The final component is a custom script, written to ensure that the pedestrian walks around the training area continuously doing an episode. The script has two public variables, the radius and frequency. The radius is the distance from the agent, that a new target point is draw within. One target point is drawn within the frequency specified, and there should therefore strike a balance between the radius and the time, such that the agent has time to travel that distance within the specified time.

A.1.1.3. Crowded areas

The crowded area is a prefab and is initialised from the academy object.

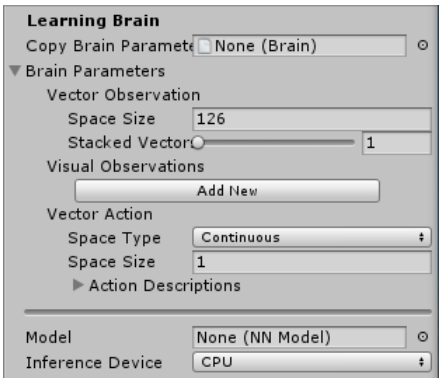
A.1.1.4. Academy

The academy is one of three cornerstones of the toolkit, and it serves to bridge actions and observations to the TensorFlow-based models in Python.
The academy is a Python API, and bridging is done through the brain. The academy has only a single component attached, seen from figure A.5.

Brain

There are two types of brains, learning brains and player trains. Learning brains learns the policy based on the net implemented in TensorFlow. See figure A.4 for the configuration of a learning brain. The player brains allow the researcher to test before invoking the learning brain, by giving the researcher the option to control the agent with keys on the keypad.

Figure A.4.: A learning brain

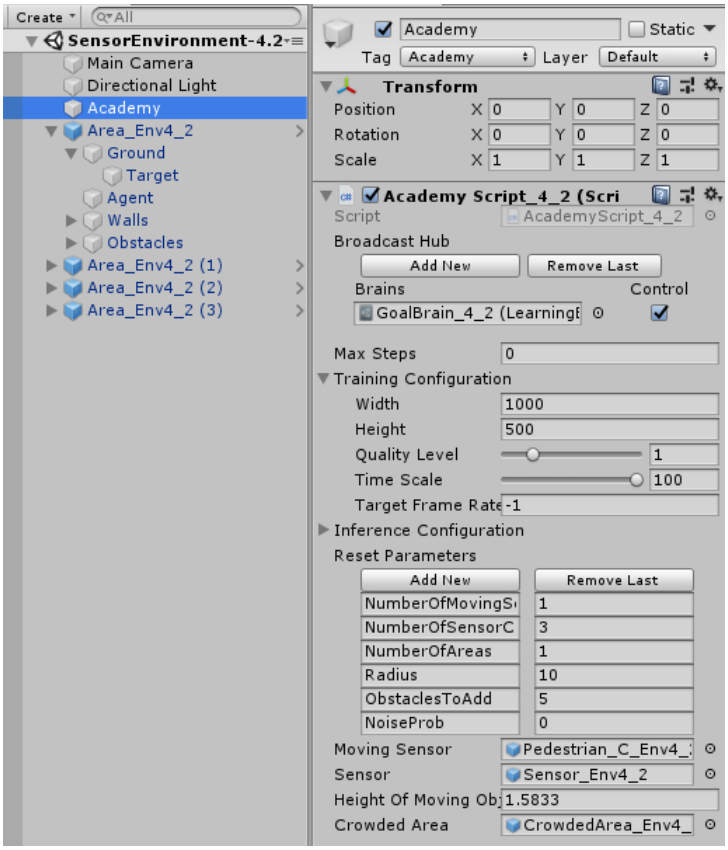


A brain can take vector observations as well as visual observations as input, and it outputs an action vector. The size of both the observation vectors and the action vector is specified by the researcher, and it is problem specific.
The learning brain used this case takes in vector observations, as visual observations requires more computational power than available for this paper.

The size of the vector is determined by the observations collected, with an example using sensor information provided in section 1.6.

Stacking observation vectors equips the agent with short-term memory, which can be beneficial when dealing with dynamic obstacles.

Figure A.5.: The academy



It is possible to add more than one brain to a single agent; however, it is not possible to train multiple brains on a single agent – yet. Training multiple brains on an agent would present some interesting possibilities, discussed at the end of this paper.

Varying complexity

An important purpose of the academy is to initialise and alter the complexity of environment. The academy script, except for some helper functions, contains two methods; InitialiseAcademy() and AcademyReset(). AcademyReset(), for this paper, is used with CL-based training, and the purpose is to update the environment with increased complexity at specific times. One could alternatively change location of certain objects from the academy, and thereby specify a maximum number of steps or call Done to reset.

However, changing location of certain objects, in this study, is done through the agent script.

Reset parameters

The reset parameters are the variables that enables changing the environment on reset, and they are input variables to the agent script as well, used when changing locations. The values of the reset parameters are being set by the curriculum[3].

Configuration

The width and the height determine the size of the application window when training is done outside the editor.

The quality level is the quality of the camera input, if visual observations are provided to the brain, and is so not relevant for this paper.

The time scale is the speed at which the simulation is carried out – 1 is real time and 100 is 100 times faster than real time. The actual level of time scale does not as such affect the performance of the training, only the training time, yet some physics calculations gets inaccurate with a too high time scale, and so affecting the performance. This should only be relevant if one has objects that travels at high speed, which isn't the case in this paper.

The time scale is set to 50, based on test simulations shown in appendix A showing minimal effect on performance and training time.

The final configuration parameter is the targeted frame rate, which is the rate at which Unity aim at rendering the frames, which shouldn't be altered unless one is using visual inputs.

Initialisation of environment

Below the reset parameters are the two prefabs, pedestrian and crowded area, attached in initialising of the environment.

[3]See section 4.2.1

A.1.1.5. Agent

The agent object is by far the most complex, in terms of the number of components and methods contained in the attached script. The content of the agent object is seen figure 19.

The agent GameObject contains a ray perception component along with two custom scripts, other than the familiar components as of the mesh, the rigidbody and the collider components. The purpose of the custom scripts is to draw movement trails and to hold the necessary methods needed, to leverage the ML-Agents toolkit.

The ML-Agents toolkit bridges sophisticated machine learning methods with the graphical interface and complex physical engine of the traditional Unity application, enabling a new setting to push the boundaries for DRL research (Juliani, 2018). The toolkit allows researcher to utilise pre-defined algorithms, based on TensorFlow, or define them themselves, via a Python API. In the light of the NavMesh class, the toolkit enables the researcher to take more control with the interaction, which carries a certain responsibility. It requires the researcher to exhibit a greater understanding of the task and modelling at hand, and so reduces the possibility of headless simulation – limiting the risk of another black box appearance.

The specifications of the agent are based on the specifications available on the current generation of ADR's. The ADR's appears to have a height around 0.5-1.5 metre, a width and depth of 0.5 metre, a total weight (including cargo) of 45-50 kgs and a speed around 5 km/h (FedEx, 2019; Starship, 2019; Scott, 2019).

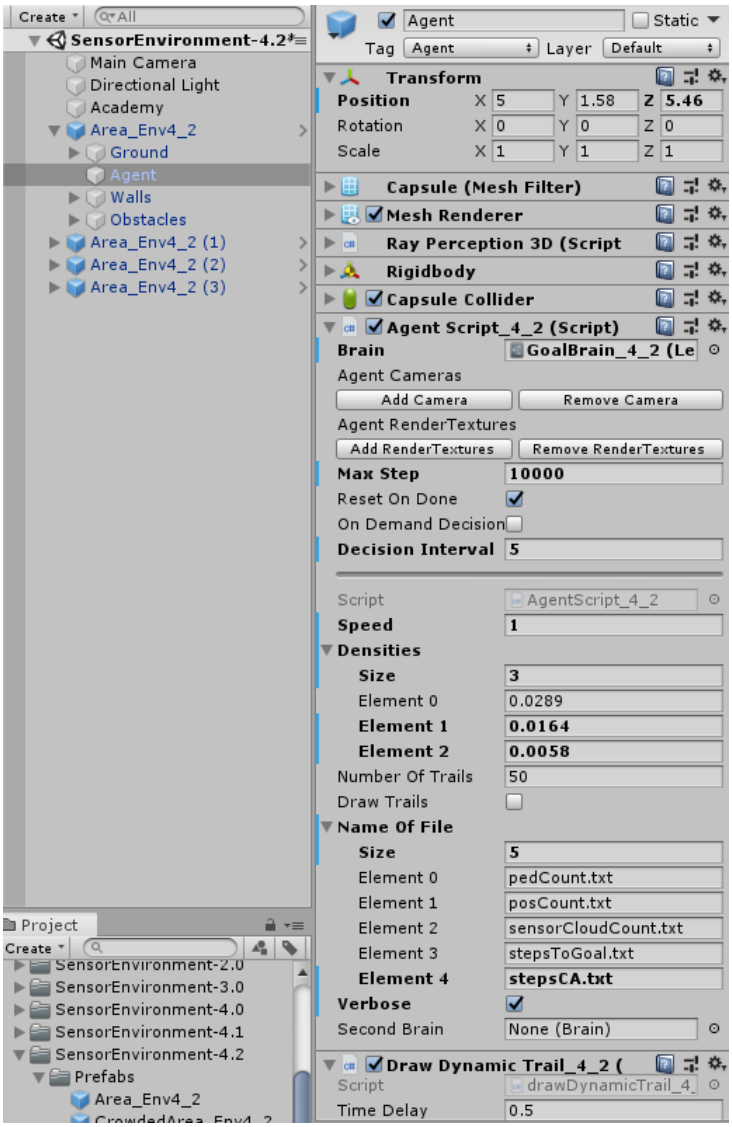
Ray perception

The ray perception component enables the agent to cast rays in specified length and direction and are here used to collect observations about the state of the environment. The rays resemble with LIDAR sensors commonly used for robots (Georgiev and Allen, 2004; Kümmerle et al., 2013; Starship, 2019; FedEx, 2019).

Agent Script

Any ML agent needs an agent script, to hold the agent-specific methods, just as the academy needed an academy script. An agent script contains some default variables and options, listed above the grey line in figure 19.

Figure A.6.: The content of the agent object



Brain

An agent needs a brain to control the movement of the agent, and it is the same brain as specified in the academy. Certain methods control the movement, more specifically; CollectObservations, AgentAction and MoveAgent.

CollectObservations serves to provide the agent with a vector of observations, sensor information in this case, and is only needed when the brain uses vector observations[4].

[4]Visual observations are provided directly to agent.

The agent has 180 degrees sensor vision, in steps of 10 degrees, spanning in front with a length of 50 meters.

The agent is provided with five tags to recognise, and a ray is casted for each of the degrees specified. For each ray is the following returned; A one-hot encoding, whether an object is detected or not (1/0) and the normalised distance to the detected object.

The one-hot encoding returns 0 or 1 for each detectable object, depending of which object was detected.

The observation vector has the dimension $(tags + 2) \cdot degrees \times 1$, which for the specific case here means that the observation vector is $(5 + 2) \cdot 18 = 126 \times 1$.

Describe partial/full observability.

The agent chooses an action, based on the observations about the current state of the environment. The action/-'s is chosen by the brain, and facilitated to the agent through the AgentAction method, in which the action signal is translated to actual movement via the MoveAgent method.

The action of the agent is degrees to turn, as the agent is moved forward with a constant speed. The speed of the agent is subject to change, via the public speed variable.

The speed of the agent is, by default yet subject for change, set equal to $1\,m/s$, which is roughly equal to a speed of $3.5\,km/t$ – the average chilled walking speed.

Max step

As with the academy, the option to specify a maximum number of steps is present. The agent will be reset when the number of steps surpasses the specified number, which is useful to break unfavourable movement patterns, as is shown later.

Reset on Done

Another way to reset the agent is to call Done at some point, which usually is after colliding with an object in the environment.

Both max step and reset on Done is used in this paper. Collisions with a wall, a static obstacle, a pedestrian or the target results in Done being called, to reset the agent, and start a new episode.

On reset, not only the position of the agent resets, but also the position of the target and the crowded areas. Re-positioning the crowded areas causes re-drawing the densities. For this reason, the option to specify levels and number of possible densities is available under the agent script.

Decisions

Decisions can be done either at a specific interval or on demand, and this paper here uses decision at a specific interval. The decision interval (DI) should be chosen with the complexity of the environment and the speed at which the agent moves in mind.

As with many of the other parameters of the environment, it is of interest to choose the level at a level which generalises. The default level of DI is chosen to be every fifth step but can be subject to change. It seems natural that greater complexity/speed should benefit from lower DI.

Camera/Render textures

The agent script contains the option to specify camera/-s and/or render textures, if the brain attached to the agent uses visual observations.

Draw trails

The agent script contains the option to enable drawing, and specify the number of trails drawn, which serves to visualise changes in learning patterns doing training. A custom implementation is used, contained in the second custom script of the agent, because the default implementation, trail renderer, does not consider the resetting of the agent. Trail renderer draws the jump from where one episode ends to the start position of the agent, which minimises the information obtained by visualising the trails.

The custom implementation simply draws a line for each episode, as either a new child element of the agent, if the number of trails is less than the specified number of trails, or by modifying the oldest trails.

Name of file

If desired, by checking verbose, the agent collects additional information, compared to the information provided via TensorBoard and writes it to specified files. The additional

information provides deeper insights into the progress of the training and highlights potential shortcomings. The additional information is information on the number of collisions with pedestrians and crowded areas, the steps used to locate the goal and the steps taken in the crowded area. However, this can be changed to suite the environment and the need of the researcher.

Second brain

The agent script is provided with an option to add a second brain, and potentially many more, which is useful to investigate the effect of separating tasks on individual brains.

A.1.1.6. Target

The content of the target is seen in figure A.7.
The target contains a custom script which serves two purposes; randomly setting a new position of the target and check for collision with static objects in the environment.

Continuously changing the position of the target increases the likelihood of the learning to generalise to unseen environments.

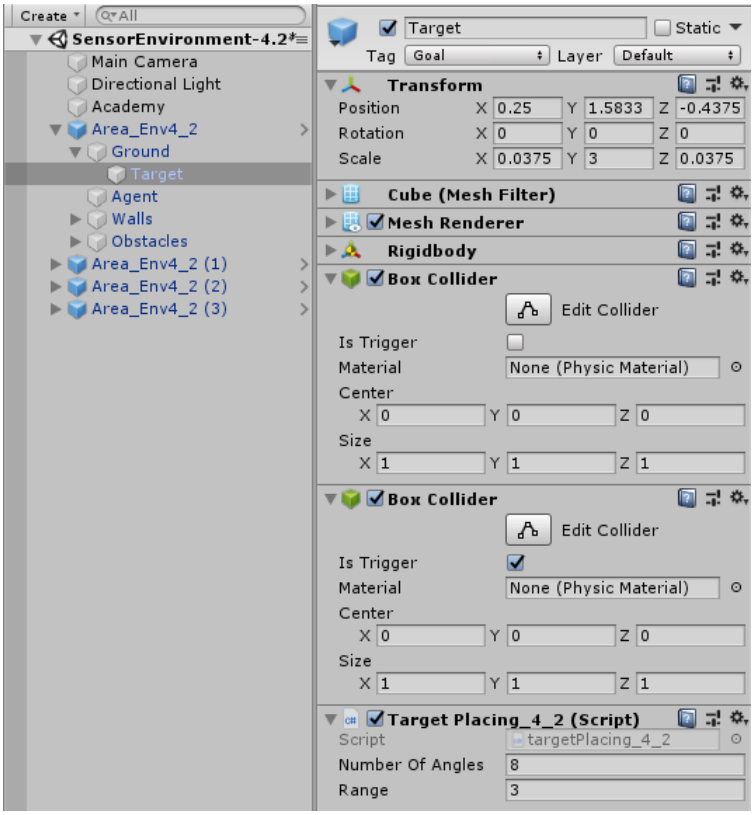
The target is not allowed to be located within another static object in the environment, to prevent conflicting collisions. Having the target located separately from other static objects, simulates the idea about locating a position outside a building for delivery by the ADR.

A.1.2. Training

Training can be done either in the editor or by running a build application of the environment. How to carry out training from the command line is described neatly by [Juliani et al., 2019b], and is so not described here.

Running training from a build application of the environment provides some desired possibilities, where two of them are running concurrent runs and concurrent environments.

Figure A.7.: The content of the target object



Concurrent runs are independent, which is beneficial for benchmarking. Concurrent environments are equivalent to having multiple training areas within one environment, which implies more experience being sampled, which should result in improve learning [Teng et al., 2019]. Running multiple environments is a way to speed up training beyond have multiple training areas. It is computationally demanding, and it is usually used together with cloud computing.

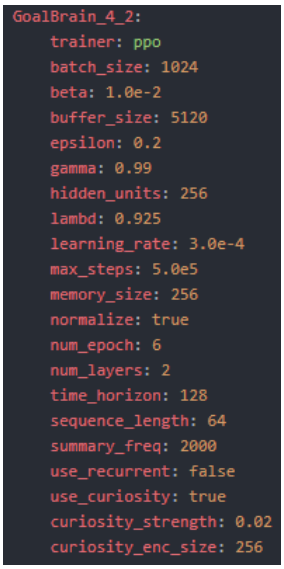
Cloud computing is not a possibility here, because of the associated cost, and this paper is so limited to the use of multiple training areas.

A.1.2.1. Training hyperparameters

As within other areas of machine learning, requires RL algorithms considerations on various hyperparameters. Tuning of the hyperparameters becomes increasingly important as the complexity of the task increases. However, when tuning the parameters to a specific task or environment, some generalisation is lost.

In this study, the hyperparameters are mostly kept at default values, with some being modified slightly to accommodate continuous actions instead of discrete. The full set of hyperparameters used is seen from figure 21, and for a discussion of the hyperparameters and how they should be set optimally, see [Schulman et al., 2017b, Juliani et al., 2019a].

Figure A.8.: Training Hyperparameters



A.1.3. TensorBoard

TensorBoard is used to visualise a wide range of statistics related to the training conducted, covering environment, policy and learning statistics. Each of the statistics available is described in detail in [Juliani et al., 2018c].

A.1.4. Effect Of Chosen Hyperparameters Of The Environment

There are three parameters that qualifies as hyperparameters of the environment, when looking back at the outlined environment. These three are; the *speed* at which both the agent and the pedestrians move, the *decision interval* of the agent and the *time scale* at this the simulations are carried out. The reason for these three parameters is because they naturally neither are part of a curriculum nor determined by the desire of having realistic physics in the simulated environment.

Table A.2.: Effect of Environment Hyperparameters

Speed	DI	TS	Training Time	Deviation	ACR, μ	ACR, σ	AEL, μ	AEL, σ
2	5	60	1650	22.2%	0.663	0.65	40.32	19.41
1	**5**	**60**	**1500**	**11.1%**	**0.920**	**0.59**	**39.55**	**15.59**
1	3	60	1350	0.0%	0.916	0.80	101.31	198.75
1	7	60	1750	29.6%	0.896	0.63	34.07	15.86
1	10	60	1800	33.3%	0.812	0.56	26.22	9.32
1	5	100	1350	0.0%	0.908	0.62	43.03	19.58
1	5	20	2020	49.6%	0.908	0.57	44.02	21.15

B. Theoretical additions

B.1. Proximal Policy Optimisation Explained

B.2. Reward Shaping

C. Analysis Support

This section contains some additional material to support the analysis, mainly by visualising the findings reported in chapter 4.

C.1. Supplement To Comparisons

This section contains graphs supporting the tables from section 4.2.2, 4.3 and 4.4. The tables serve to summaries the insights from the graphs found here.

ACR/AEL are smoothed to better visualise, by calculating a moving average of the past three observations.

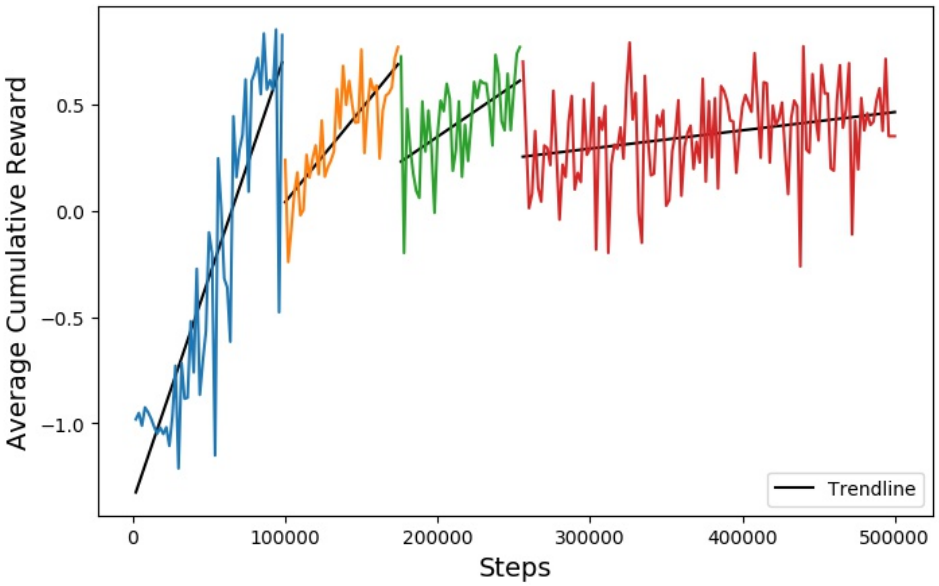
Weighted average growth rates are reported for the ACR curve, weighted by the time spent in each lesson. The growth rates are estimated by fitting a linear regression to observations in each lesson. The reported growth rates are per 2000 step, which is the summary frequency used doing training, see figure 21. The standard deviation is calculated on the detrended series.

The mean episode length and the standard deviation, calculated after 100,000 steps, are reported for the AEL curve.

C.1.1. Learning Rate Estimation

Figure C.1 shows how the assumption about each subsection following a linear curve isn't that off, as a result of the threshold and the minimum lesson length specified in the curriculum, seen from figure 4.3.

Figure C.1.: Subsets and trend estimation



C.1.2. Baseline

C.1.3. Learning Under Certainty

C.1.4. Learning Under Uncertainty

Figure C.2.: Baseline comparison

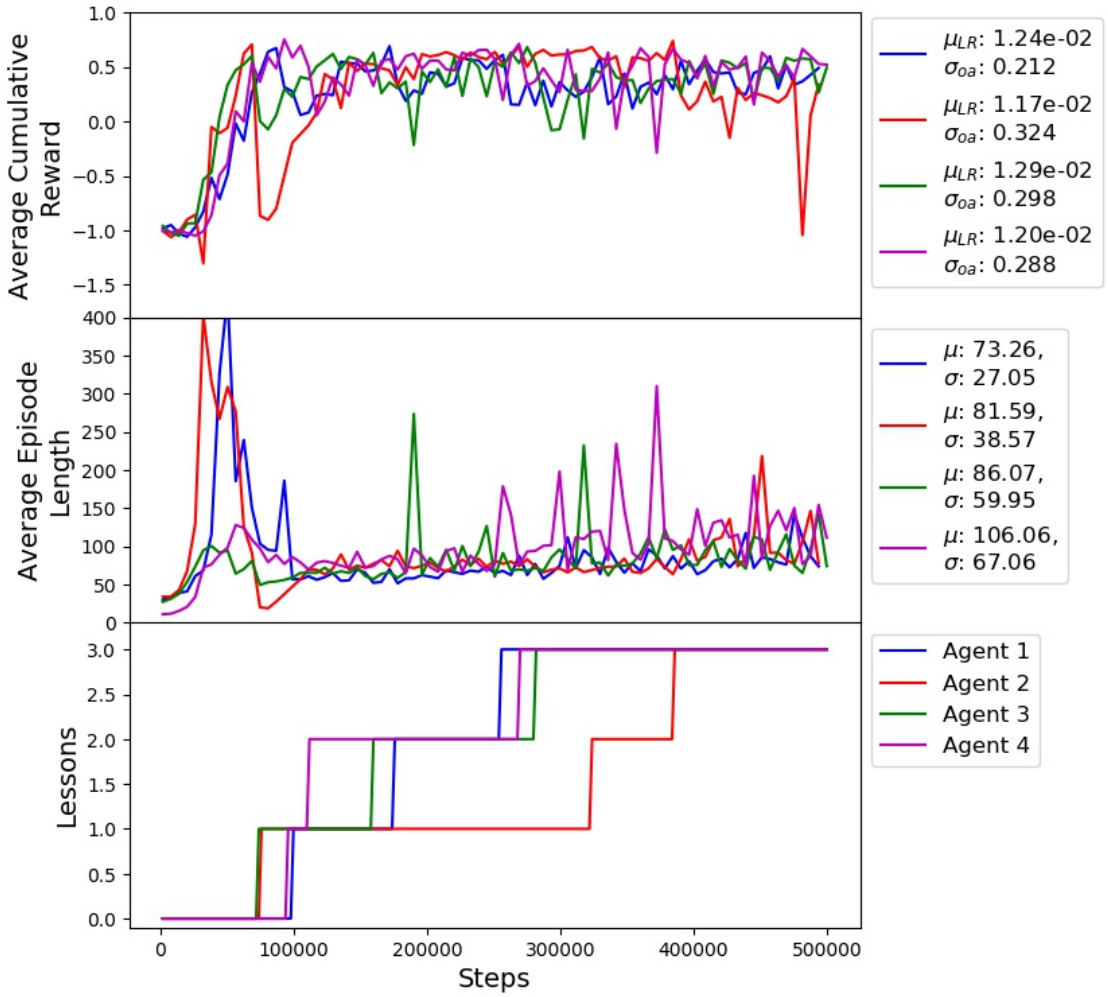


Figure C.3.: Full Set-Up Under Certainty

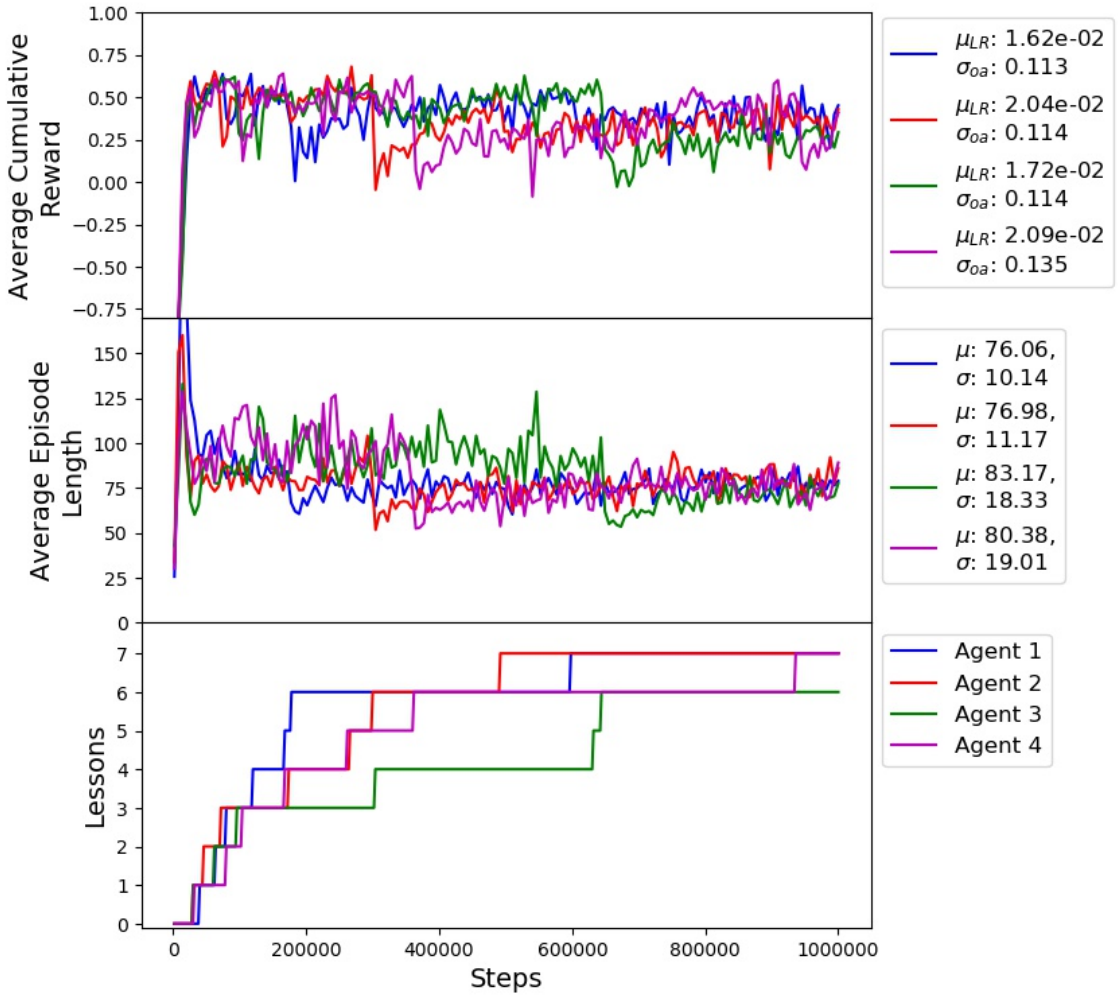
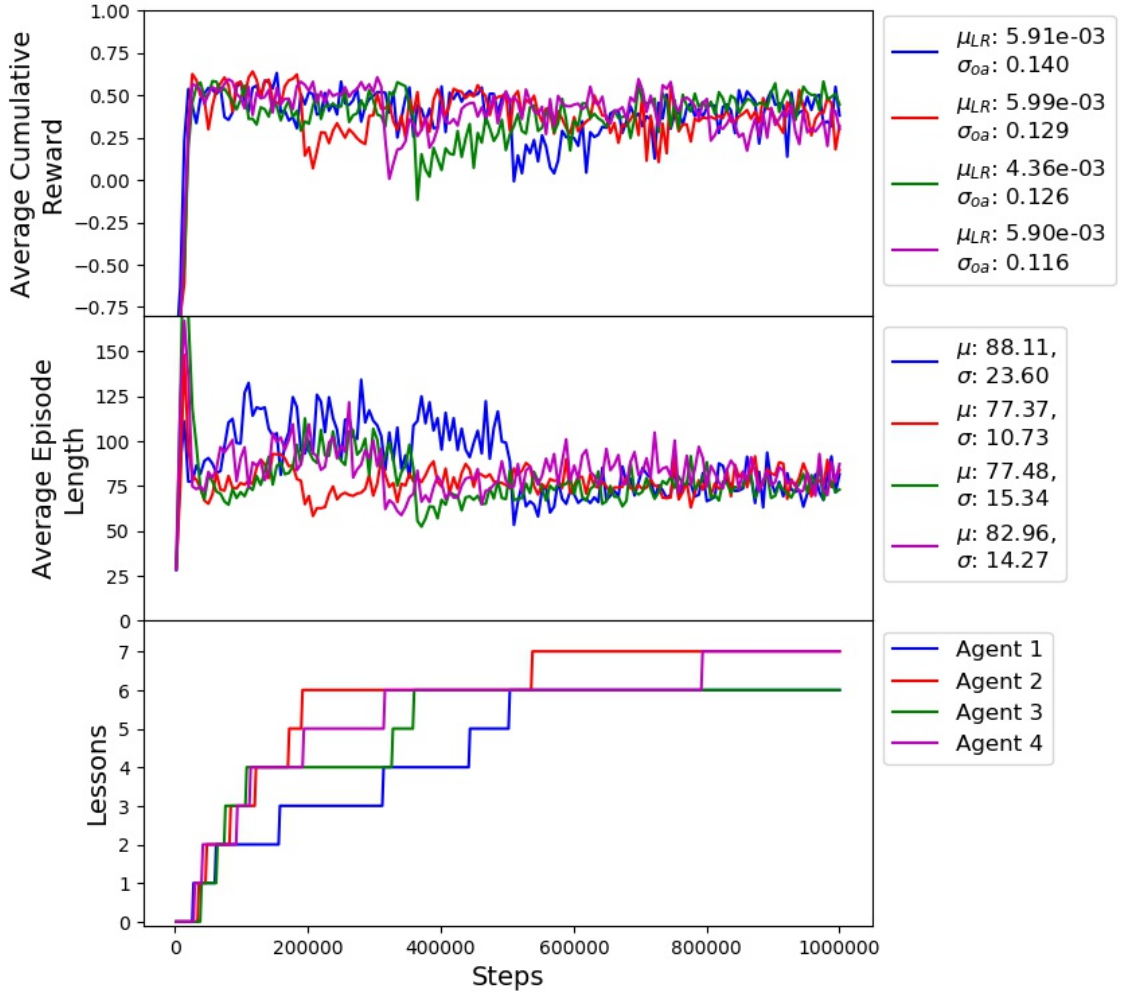


Figure C.4.: Full Set Up Under Uncertainty



Bibliography

[Anderson, 1986] Anderson, C. W. (1986). Learning and problem solving with multilayer connectionist systems.

[Bellemare et al., 2015] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2015). The arcade learning environment: An evaluation platform for general agents. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 4148–4152. AAAI Press.

[Bengio et al., 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA. ACM.

[Borsa et al., 2019] Borsa, D., Heess, N., Piot, B., Liu, S., Hasenclever, L., Munos, R., and Pietquin, O. (2019). Observational learning by reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 1117–1124, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Butcher, 2018] Butcher, S. (2018). Jpmorgan's new guide to machine learning in algorithmic trading.

[FedEx, 2019] FedEx (2019). Delivering the future: Fedex unveils autonomous delivery robot.

[Georgiev and Allen, 2004] Georgiev, A. and Allen, P. K. (2004). Localization methods for a mobile robot in urban environments. *Trans. Rob.*, 20(5):851–864.

[Givens and Hoeting, 2012] Givens, G. H. and Hoeting, J. A. (2012). *Computational statistics*. John Wiley & Sons, Hoboken, NJ, USA, 2 edition.

[Hessel et al., 2017] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Irpan, 2018] Irpan, A. (2018). Deep reinforcement learning doesn't work yet. `https://www.alexirpan.com/2018/02/14/rl-hard.html`.

[Juliani, 2018a] Juliani, A. (2018a). Ml-agents toolkit v0.3 beta released: Imitation learning, feedback-driven features, and more.

[Juliani, 2018b] Juliani, A. (2018b). Solving sparse-reward tasks with curiosity.

[Juliani et al., 2018a] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2018a). Unity: A general platform for intelligent agents. *CoRR*, abs/1809.02627.

[Juliani et al., 2019a] Juliani, A., Mattar, M., Teng, E., Berges, V.-P., Dong, R.-P., Elion, C., Ward, J., and Shih, J. (2019a). Training with proximal policy optimization.

[Juliani et al., 2018b] Juliani, A., Pang, D., Shih, J., Berges, V.-P., and Ward, J. (2018b). Training with curriculum learning.

[Juliani et al., 2018c] Juliani, A., Pang, D., Shih, J., and Ward, J. (2018c). Using tensorboard to observe training.

[Juliani et al., 2019b] Juliani, A., Peng, D., Berges, V.-P., Gao, Y., Mattar, M., Shih, J., Ward, J., Vckay, E., and Puida, M. (2019b). Training ml-agents.

[Kahn et al., 2017] Kahn, G., Villaflor, A., Pong, V., Abbeel, P., and Levine, S. (2017). Uncertainty-aware reinforcement learning for collision avoidance. *ArXiv*, abs/1702.01182.

[Karpathy, 2016] Karpathy, A. (2016). Deep reinforcement learning: Pong from pixels.

[Kümmerle et al., 2013] Kümmerle, R., Ruhnke, M., Steder, B., Stachniss, C., and Burgard, W. (2013). A navigation system for robots operating in crowded urban environments. In *IEEE International Conference on Robotics and Automation*, pages 3225–3232.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature International Journal of Science*, 521(7553):436–444.

[Lidoris et al., 2009] Lidoris, G., Rohrmüller, F., Wollherr, D., and Buss, M. (2009). The autonomous city explorer (ace) project - mobile robot navigation in highly populated urban environments. In *IEEE International Conference on Robotics and Automation*, pages 1416–1422.

[Lin, 1992] Lin, L.-J. (1992). *Reinforcement Learning for Robots Using Neural Networks.* PhD thesis, Pittsburgh, PA, USA. UMI Order No. GAX93-22750.

[Maria Bauer et al., 2009] Maria Bauer, A., Klasing, K., Lidoris, G., Mühlbauer, Q., Rohrmüller, F., Sosnowski, S., Xu, T., Kühnlenz, K., Wollherr, D., and Buss, M. (2009). The autonomous city explorer: Towards natural human-robot interaction in urban environments. *I. J. Social Robotics*, 1:127–140.

[Matter et al., 2018] Matter, M., Pang, D., and Shih, J. (2018). Memory-enhanced agents using recurrent neural networks.

[Merrit, 2019] Merrit, T. (2019). Top 5 delivery robots.

[Minsky, 1954] Minsky, M. (1954). *Theory of neural-analog reinforcement systems and its application to the brain-model problem.* PhD thesis.

[Mirowski et al., 2018] Mirowski, P., Grimes, M. K., Malinowski, M., Hermann, K. M., Anderson, K., Teplyashin, D., Simonyan, K., Kavukcuoglu, K., Zisserman, A.,

and Hadsell, R. (2018). Learning to navigate in cities without a map. *CoRR*, abs/1804.00168.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature International Journal of Science*, 518(7540):529–533.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA. Omnipress.

[Nichols, 2019] Nichols, G. (2019). Delivery wars: Amazon's new delivery robot vs starship's college munchie robot.

[OpenAI et al., 2018] OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2018). Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177.

[Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363.

[Pärnamaa, 2018] Pärnamaa, T. (2018). How neural networks power robots at starship.

[Rojas, 1996] Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlin, Heidelberg.

[Sadeghi and Levine, 2016] Sadeghi, F. and Levine, S. (2016). (cad)$ˆ2$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201.

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *International Business Machines Corporation*, 3(3):210–229.

[Schulman et al., 2017a] Schulman, J., Klimov, O., Wolski, F., Dhariwal, P., and Radford, A. (2017a). Proximal policy optimization.

[Schulman et al., 2017b] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

[Scott, 2019] Scott, S. (2019). Meet scout.

[Sermanet et al., 2013] Sermanet, P., Kavukcuoglu, K., Chintala, S., and Lecun, Y. (2013). Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 3626–3633, Washington, DC, USA. IEEE Computer Society.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature Internation Journal of Science*, 529(7587):484–489.

[Starship, 2019] Starship (2019). The self-driving delivery robot.

[Sutton, 1984] Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis. AAI8410337.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[Teng, 2019] Teng, E. (2019). Multiple environments when training: –num-envs vs multiple training areas within one application.

[Teng et al., 2019] Teng, E., Vckay, E., Harper, J., and Gao, Y. (2019). Faster training on real games.

[Trulls et al., 2011] Trulls, E., Murtra, A. C., Pérez-Ibarz, J., Tur, J. M., and Sanfeliu, A. (2011). Autonomous navigation for mobile service robots in urban pedestrian environments. *Journal of Field Robotics*, 28:329 – 354.

[Unity, 2019] Unity (2019). Navigation and pathfinding.

[Watkins and Dayan, 1992] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. In *Machine Learning*, pages 279–292.

[Yen and Hickey, 2004] Yen, G. G. and Hickey, T. W. (2004). Reinforcement learning algorithms for robotic navigation in dynamic environments. *ISA Transactions*, 43:217–230.

[Zhou et al., 2019] Zhou, X., Gao, Y., and Guan, L. (2019). Towards goal-directed navigation through combining learning based global and local planners. *Sensors*.