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:

# Abstract

# 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

Table of Contents

[Abstract 2](#_Toc17050643)

[Declaration 3](#_Toc17050644)

[List of tables 6](#_Toc17050645)

[List of figures 7](#_Toc17050646)

[List of acronyms and abbreviations 8](#_Toc17050647)

[Acknowledgements 9](#_Toc17050648)

[1. Introduction 10](#_Toc17050649)

[1.1 Theoretical background 11](#_Toc17050650)

[1.2 Purpose of the Study 13](#_Toc17050651)

[1.3 Significance of the Study 14](#_Toc17050652)

[1.4 Scope of the Study 14](#_Toc17050653)

[2. Literature Review 15](#_Toc17050654)

[2.1 Reinforcement learning 15](#_Toc17050655)

[2.2 Deep reinforcement learning 20](#_Toc17050656)

[2.2.1 Policy Gradient Methods 20](#_Toc17050657)

[2.2.2 Proximal Policy Optimisation 21](#_Toc17050658)

[2.5 Robotic Navigation in Urban Environments 22](#_Toc17050661)

[2.5.1 Robotic Navigation in Urban Environments using Reinforcement Learning 24](#_Toc17050662)

[3. Methodology 25](#_Toc17050663)

[3.1 Autonomous delivery robots today 26](#_Toc17050666)

[3.2 Unity – as a simulation engine for research in DRL 28](#_Toc17050667)

[3.3 The Environment 29](#_Toc17050670)

[3.3.1 Environment 30](#_Toc17050671)

[3.3.2 Robotic agents 31](#_Toc17050672)

[3.3.2.1 Uncertainty 32](#_Toc17050673)

[3.3.3 Rewards 33](#_Toc17050674)

[3.3.3.1 Extrinsic 34](#_Toc17050675)

[3.3.3.2 Intrinsic 34](#_Toc17050676)

[3.3.4 Pedestrians 35](#_Toc17050677)

[3.3.5 Crowded Areas 35](#_Toc17050678)

[3.3.6 Target 39](#_Toc17050679)

[4. Analysis 40](#_Toc17050680)

[4.1 The look of learning 40](#_Toc17050681)

[4.2 Learning to learn 42](#_Toc17050682)

[4.2.1 Curriculum Learning 43](#_Toc17050683)

[4.2.2 Exploring the environment under CL 45](#_Toc17050684)

[4.2.3 Maximum steps allowed for agent 46](#_Toc17050685)

[4.2.4 Shared Experience 50](#_Toc17050686)

[3.2.1 Observation stacking 53](#_Toc17050687)

[5. Results under certainty 55](#_Toc17050688)

[Insights: 56](#_Toc17050689)

[Thoughts: 56](#_Toc17050690)

[3.4 Discussion 57](#_Toc17050691)

[Stuff for the discussion: 57](#_Toc17050692)

[3.5 Future work 57](#_Toc17050693)

[4. Policy Evaluation 58](#_Toc17050694)

# List of tables

# List of figures

# List of acronyms and abbreviations

Net – neural network

Ppo – Proximal Policy Optimisation

RL

DL

DRL

# Acknowledgements

# 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 (Mirowski et al, 2018; Kahn et al., 2017; 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.

## 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), into large scale real-world applications.

* + 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* functionor 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 – Markov Decision Process*A screenshot of a cell phone

Description automatically generated *Credit: (Sutton and Barto, 2018)*

Any MDP is made up by a set of actions, , a set of states,, a set of rewards, , 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]](#footnote-1) MDP’s, which occurs rarely in practice, at least in any interesting applications.

The order in figure 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 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]](#footnote-2) leading up to the terminal state.

## 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., 2018) 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.

## Significance 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 (Zhou et al, 2019; Kahn et al., 2017). 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.

## 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., 2017).

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 contributions 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.

# Literature Review

The following proceeds by first reviewing RL, the emergence of DRL and *the[[3]](#footnote-3)* 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.

## 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[[4]](#footnote-4)* (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, , which is compared to an alternative successor action, drawn from the target policy, . 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, , the online update of the state action-value function is done according to (1).

Both policies are improved, with the target being updated greedily[[5]](#footnote-5) and the behaviour -greedy[[6]](#footnote-6), both w.r.t. , which simplifies the target as seen from (2).

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

*Figure 2 – Q-learning algorithm*  
A screenshot of a cell phone

Description automatically generated  
*Credit: (Sutton and Barto, 2018)*

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[[7]](#footnote-7), illustrated in (3).

The Q-learning algorithm showed in figure 2 can seem too simple to work on real-life large-scale control problems. However, it illustrates the high-level idea, about improve 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 (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 . 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 to , 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 . ReLU, , bounds the outcome to , which has shown to provide much faster training of deep nets (Lecun et al, 2015; Nair and Hinton, 2010; Krizhevsky et al., 2012).

* 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[[8]](#footnote-8) 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).

These advances along with the use of *experience replay[[9]](#footnote-9)* (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[[10]](#footnote-10) level in the boardgame GO, which was regarded as one of the grand challenges, because of its enormous state space consisting of 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.

## 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.

### 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[[11]](#footnote-11) 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 . How must to adjust, and so how to update the policy, is determined by the gradient of the scalar signal[[12]](#footnote-13) and called backpropagation[[13]](#footnote-14), seen from (4).

With being the learning rate and 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[[14]](#footnote-15), using a basic PG.

The structure is as follows; we receive an image, of size 210x160x3[[15]](#footnote-16), 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 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.

### 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.

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



## 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 obviously 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 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]](#footnote-17) module (Lidoris et al., 2009; Kümmerle et al., 2013; Maria Bauer et al., 2008; Trulls et al., 2011; Georgiev and Allen, 2004). 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; (Maria Bauer et al., 2008; Lidoris et al., 2009; Kümmerle et al., 2013) 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) (Trulls et al., 2011; Lidoris et al., 2009; Kümmerle et al., 2013; Maria Bauer et al., 2008).

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.

### 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.

# 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 1.3, 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.

State hypothesis?



## Autonomous delivery robots 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 (Merrit, 2019; Nichols, 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 (FedEx, 2019; Starship, 2019; Scott, 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.

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.

## 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 (Juliani et al, 2018; OpenAI, 2018; Sadeghi & Levine; 2016).

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.



## 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.

### 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.  
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 – The environment*  
A close up of a screen

Description automatically generated  
*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).

### 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.

#### Uncertainty

Uncertainty is introduced to the environment by adding standard Gaussian noise, , to the sensor observations received by the agent, when the agents gets within 20 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.

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%[[17]](#footnote-18), 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 using a Box-Muller transformation of a two-dimensional continuous uniform distribution, seen in (6), outputting a two-dimensional bivariate normal distribution (Givens and Hoeting, 2012; table 6.1). One of the Gaussian variables are chosen with an equal probability.

**Box-Muller transformation:**

|  |  |  |
| --- | --- | --- |
|  | *If and are uniformly and independent distributed between 0 and 1, then*  *and are independent .* | (6) |

### 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.

#### 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.

#### Intrinsic

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

A full treatment is given by (Pathak et al., 2017; Juliani, 2018), 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.

### Pedestrians

The pedestrians are *NavMesh agents* (Unity, 2019), 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.

### 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[[18]](#footnote-19) of (Yen and Hickey, 2004).  
As an alternative, the idea of including crowded areas arose. 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[[19]](#footnote-20), 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 1 and figure 4 & 5.   
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 1 - Population densities, London.** | |
| Max Ward: | 0.0289 |
| Max Boroughs: | 0.0164 |
| Overall:  **Re-create in LaTeX** | 0.0058 |

*Figure 4 – Population density, Boroughs, London  
A close up of a piece of paper

Description automatically generated*

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

The loss of data in figure 5 occurs because the population density data comes without boundaries, and the data needs to be merged onto a shapefile containing the boundaries.  
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[[20]](#footnote-21). 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.

*Figure 5 – Population density, Wards, London   
A close up of a piece of paper

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

### 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[[21]](#footnote-22).

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 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%[[22]](#footnote-23), chosen arbitrary and the effect of the relative size of the target could be a topic for future investigation.

# 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.

## The look of learning

Learning can be expressed in many ways (Juliani et al, 2018b), however, the most intuitive statistics is the ACR and the ALE, examples are seen from figure 7.

The ACR is expected to converge to the maximum obtainable level of reward within a single episode, when learning has taken place and the agent has solved the environment. 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 bound the maximum obtainable reward.   
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.

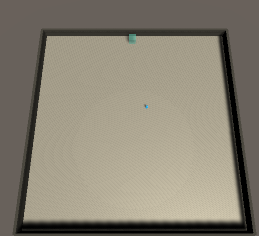
*Figure 6 – The simplest version of the environment.*  
   
*The simple environment contains only a target (darker green square) and the agent (light blue).*

Figure 6 shows the simplest possible environment, without any obstacles. Learning in this environment is straight forward, and this is seen from figure 7, 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 7 – ACR and ALE for the environment from fig. 3 and 6*  
*A close up of text on a white background

Description automatically generated*

The shapes of ACR and ALE seen in figure 7 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, 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 7 contains just 200.000 steps, which in general is regarded as few, and the purpose of figure 7 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 (3). The sensitivity just affects the length of *over time*, and some of ways to guide learning in 4.2 significantly reduce the sensitivity.

## Learning to learn

Depending on the environment, it is sometimes necessary to aid the learning process of an agent, as a result of what model-free RL basically are – a function mapping, often high-dimensional, from states to actions. In English, that means that too complex states can result in odd behaviour, if the quality of the function mapping is too poor at the time being.   
This section covers different ways to minimise the occurrence of destructive behaviour, by using aids to develop a high-quality function mapping, to facilitate meaningful learning.

### 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 8. For a discussion of each of the parameters in the curriculum, see (Juliani et al., 2018).

*Figure 8 – The curriculum of this study*A screen shot of a computer

Description automatically generated

The parameters of the CL are specified through a dictionary, and the keys of the dictionary referrers to the reset parameters[[23]](#footnote-24) of the academy. The dictionary contains at least one 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 . 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.

### Exploring the environment 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.

Figure 9 & 10 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.

*Figure 9 – Baseline comparison*A screenshot of a cell phone

Description automatically generated *The means and standard deviations of are calculated after 100,000 steps.*

One should be cautions 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 9 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[[24]](#footnote-26) and that there isn’t much difference between agent 2 and 3.

*Figure 10 – Efficiency in Difficult Areas  
A close up of a map

Description automatically generated  
The means and standard deviations of are calculated after 2,000 increments.*

Figure 10 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.

### Maximum steps allowed for agent

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.

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 9 and 10[[25]](#footnote-27). Figure 9 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 10 shows the actual number of steps for each episode of blue run, with one episode taking around 35.000 steps. Table 2 shows how many observations lies above certain thresholds. The threshold used on the red run, in figure 9, is , which from table 2, is equivalent to excluding 0.5% of the total observations.

*Figure 9 – The Effect of Restricting the number of steps*  
*A screenshot of a cell phone

Description automatically generated*

Excluding such a tiny fraction of the observations results in a lot more stable training, by reducing the skewness of the steps-distribution[[26]](#footnote-28).

*Table 2 – Fraction of observations lost with maximum steps invoked*A screenshot of a cell phone

Description automatically generated

Figure 10 shows two important findings. Each graph, in figure 10, 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.

*Figure 10 – Number of Steps Within Each Episode  
A screenshot of a cell phone

Description automatically generated*

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 3.

*Table 3 – Learning the difficult areas*  
A close up of a keyboard

Description automatically generated

Table 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.

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 is used.

### Shared Experience

Parallelised training is well-known is be beneficial in DL and DRL (Mnih et al., 2013; Silver et al., 2016; Lecun et al., 2015; 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 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 11. 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).

*Figure 11 – 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.*

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)[[27]](#footnote-29).

*Figure 12 – Shared Experience  
A screenshot of a cell phone

Description automatically generated*

The power in sharing experience is seen from figure 12. Figure 12 provides a detailed look of the AEL curve, in figure 26[[28]](#footnote-30), from the area bounded by the black line. Figure 12 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.

### 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, 2018).  
Observation stacking is an alternative to the use of recurrent nets proposed by (Hochreiter and Schmidhuber, 1997), and explored by (Mnih et al., 2015; Borsa et al., 2019), for example to achieve human-level control on 47 ATARI games.

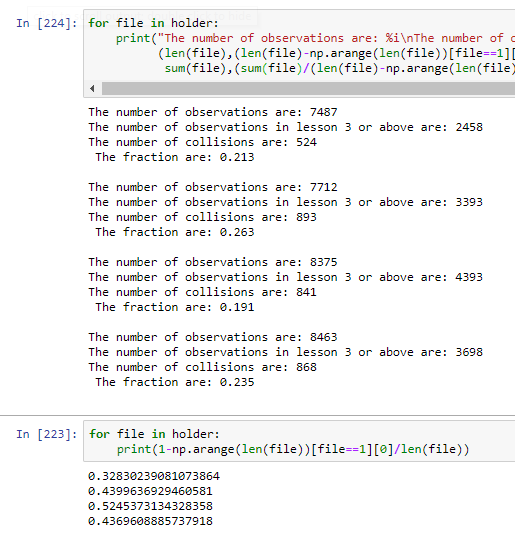
The effect of stacked observations is explored in this study, because recurrent nets have been found not to work well with continuous vector action spaces (Matter et al., 2018), which is used by the agent of this study.

The effect of stacking observations is interesting when the dynamic obstacles are present in the environment, illustrated in figure 13. Figure 13 compares the performance, in terms of ACR and AEL, of the baseline from figure 9 and two unbounded[[29]](#footnote-31) 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 13 is the stability of the ACR, but especially the AEL, from the two stacked runs (green and purple).

*Figure 13 - The Effect of Observation Stacking  
A screenshot of a cell phone

Description automatically generated*

Another interesting angle for comparison, between the four runs from figure 13, is to look at the number of collisions with pedestrians, see table 4. Table 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.

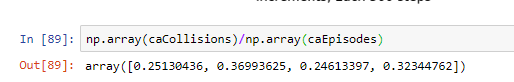
*Table 4 – Pedestrian collisions  
*

## Results under certainty

Hej

Table 5 – Training times & CA Efficiency.

Do in TeX



Remaining structure:

* ~~3.2.5 to section 3.~~
* Comparison of performance
* ~~Uncertainty section~~
* Discussion
* Conclusion

## Insights:

*Randomization of rewards usually poses a problem for the agent - it doesn't know what to do because it doesn't know what reward to expect.*

*.. the agent can't learn a perfect model of how observations correspond to rewards (i.e., they're "noisy")*.

Source: <https://github.com/Unity-Technologies/ml-agents/issues/1680>

## Thoughts:

If it turns out that, no matter the actions taken, it seems impossible for the agent to learn avoiding the sensor clouds, presumably because of the randomness in the penalties of the steps (as derivative of above), one idea can be to let the penalty be fixed but use the density to change timescale/speed.

## Discussion

## Stuff for the discussion:

* As RL is nothing more than a function mapping, either exact or approximative, it follows sort of trivially that randomness makes the mapping blurry, which is the reason why the agents has a hard time dealing with the sensor clouds.
* Indirect addressing the challenges of the difficult areas using CL and parallelisation of training areas.
* Similarity between RL and traditional boosting
  + How the optimisation work, and why we don’t need to run *many* runs of our RL application to get statistically significant results.
* Unity as a software of ABM

With the introduction of ML-agents, can Unity partly be regarded as the new kid in the class of software’s usable for Agent-Based modelling (ABM). The ML-Agents toolkit makes Unity an appealing contender to the traditional software’s used for ABM, by the fact that the scripting languages are C# and Python. The use of C# and Python implies low latency along with a wide variety of options for further data processing, through open source libraries, and user support, from the enormous communities surrounding the two languages.

## Future work

* Increasing the action space

Increasing the action space of the agent increases the degree-of-freedom of the agent, which be beneficial in navigation tasks with the presence of dynamic obstacles.

# Policy Evaluation



1. All sets of the MDP are finite. [↑](#footnote-ref-1)
2. This is not entirely true, as it is common practice to reward the agent with a minor negative reward at each state (also called *step*), to incentive fast learning. [↑](#footnote-ref-2)
3. A reference to the fact that this class of methods is the most used today (Karpathy, 2016). [↑](#footnote-ref-3)
4. see (Sutton and Barto, 2018) for a description of both methods. [↑](#footnote-ref-4)
5. A greedy policy is a policy, that chooses the action resulting in the maximum outcome. [↑](#footnote-ref-5)
6. A -greedy policy is a policy, which with probability chooses a random action and with probability chooses the greedy action. [↑](#footnote-ref-6)
7. See (Sutton and Barto, 2018), Chapter 9, section 4 for a full description. [↑](#footnote-ref-7)
8. See (Lecun et al., 2015) for a detailed description, of the architecture and each of the components. [↑](#footnote-ref-8)
9. N previous observations are stored, over many episodes, from which T *experiences* are randomly sampled from . Updates are done on the sampled experiences, and the agent chooses an action according to an -greedy policy, i.e. off-line learning, (Mnih et al., 2015). [↑](#footnote-ref-9)
10. Page 5, (Silver et al., 2016). [↑](#footnote-ref-10)
11. The first derivative of a function. [↑](#footnote-ref-11)
12. See (Lecun et al., 2015: P. 436, 2nd paragraph towards the bottom) for a straightforward explanation. [↑](#footnote-ref-13)
13. Training of nets using gradient of the objective function, see (Rojas, 1996: Chapter 7) and footnote 13. [↑](#footnote-ref-14)
14. A piece of software, integrating The Arcade Learning Environment (Bellemare et al., 2013) and the preferred script editor of the researcher. [↑](#footnote-ref-15)
15. Height, width and channels. [↑](#footnote-ref-16)
16. See (Lidoris et al., 2009) for a description. [↑](#footnote-ref-17)
17. The standard deviation divided by the distance at which the uncertainty occurs, (1 / 15) \* 100 = 6.6%. [↑](#footnote-ref-18)
18. 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. [↑](#footnote-ref-19)
19. Reflecting the work that would have been done by the low-level agent. [↑](#footnote-ref-20)
20. See *Population\_density\_graphs.ipynb* in the accompanying code file. [↑](#footnote-ref-21)
21. See appendix for more technical detail. [↑](#footnote-ref-22)
22. Equivalent to a size of 3 meters. [↑](#footnote-ref-23)
23. See appendix. [↑](#footnote-ref-24)
24. In terms of locating the target, seen from the fact that agent 4 scores highest average ACR. [↑](#footnote-ref-26)
25. See section 5.5 for a more extreme case. [↑](#footnote-ref-27)
26. See appendix 5.1 [↑](#footnote-ref-28)
27. ### Section *J.P. Morgan has been using reinforcement learning algorithms to place trades, even though this can cause problems.*

    [↑](#footnote-ref-29)
28. See appendix. [↑](#footnote-ref-30)
29. In terms of maximum steps allowed. [↑](#footnote-ref-31)