Deep Learning rooted Particle Swarm Navigation in 3D-environments

G035 (s1653529, s1983630, s1965737)

Abstract

This article presents a strategy for the efficient navigation of particle swarms in three-dimensional environments with obstacles. The proposed strategy involves the utilization of proximal policy optimisation and three-dimensional convolution. The navigation task includes the sequencing of uniform control outputs for multiple particles in order to yield collision-free paths to a target position. The environments are pixel-discretised and generated randomly. The experiments demonstrate the algorithm's ability to avoid obstacles on maps of varying size and carry the particles safely towards the target position. We also explore the number of particles as an influencing factor on the reward performance and compare the computing time to traditional pathfinding algorithms. All experiments delivered satisfying results, especially in terms of computation times. Larger swarms showed some convergence issues with regard to collision avoidance.

1. Introduction

Deep Learning has become an established tool to realise autonomous decision making. Path planning tasks make up some of the most challenging decision-making problems. Besides the minimisation of the error to the target configuration, collision avoidance is often an essential component of mobile robot navigation and path planning. Deep Learning promises to be a robust and fast approach to solving the issues that arise in path planning tasks. State-of-the-art algorithms for discrete as well as continuous spaces include PPO (John Schulman, Aug 2017), TRPO (John Schulman, Apr 2017) and DDPG (Timothy P. Lillicrap, Jul 2019). For bigger environments with many obstacles learning algorithms often end up oscillating in their reward accumulation; potentially disabling the value function and policy to converge. This is especially the case for continuous observation and action spaces. A common approach is often to discretize these spaces. This paper utilizes a pixel-discretised 3D-environment. The task itself is defined as the navigation of multiple particles through an obstacle course and towards a target pixel. The particles have a size of one pixel. It aims to improve the convergence time and encourage the agents to stay close to the optimal paths by introducing a Dijkstra/A*-based reward system. Collisions of any kind should be avoided. The goal is to enable near-optimal navigation of agent swarms with a unified control input.

The project was mainly inspired by uniform intravenous drug delivery control (Ajeet Kaushik & Nair, Jul 2014). The paper by Ajeet Kaushik & Nair is based on the idea that so-called nano-carriers can be loaded with certain drugs. These ferromagnetic carriers can then be manipulated through alternations to local electromagnetic fields. This control mechanism applies its manipulations uniformly to all carriers. It facilitates targeted drug deliveries past potential barriers that tissues impose. The idea is to use computer tomography to acquire three-dimensional tissue scans that can then be fed into an agent that learns to deliver swarms of nano-carriers to a target position and past as many obstacles as possible. Other utilizations may include the uniform control of drone swarms. Many other applications are thinkable.

2. Data Set Generation

For the purpose of our project task, we need a data set containing 3D maps, start and endpoints on the maps and the shortest path calculated between the points. We decided to develop a random map generator and a pathfinder module which enable us to generate an unlimited number of different data sets. This was mostly due to a lack of available data sets for the task at hand but also enabled us to generate the data in such a way that we do not require any data pre-processing. Since we need differently sized maps for the experiments, the data set is generated on several parameters that are specific to an experiment. The number of maps in a data set is predefined and dependent on the requirements of the experiment. The map dimensions are specified with the format (n, m, l). Other generation parameters include the maximum number of obstacle objects on a map, the maximum obstacle size measured in the number of pixels it is made of, the respective minimum obstacle size and the search algorithm used to generate the optimal path of a single particle. The optimal paths are calculated with the A*(Hart et al., 1968) or Dijkstra(Dijkstra, 1959) algorithm for each map, the maps and paths are saved in separate binary files for use in the experiments. For simplicity, given each of the classical path-finding algorithms was only able to calculate a single path for a single agent in any instance, these algorithms were extended with a *walkable* function. This function could then be used to recalculate paths if obstacles were to be encountered by an agent of size (x, y, z), providing a more accurate representation of a path that would be found by the algorithm for a swarm of particles. This gives a benchmark against which the performance of our model can be measured in terms of both optimality and computational efficiency.

3. Methodology

3.1. Background

Proximal Policy Optimization (PPO) is a relatively new technique in reinforcement learning that can deliver state-of-the-art results combined with significantly reduced complexities in its implementation and tuning (John Schulman, Aug 2017). PPO solves several common problems associated with Policy Gradients (PG) where small differences in stochastic policies/actions across multiple episodes may create high variance and conflicting reward values which slow the optimization process and therefore harm convergence. While increasing the batch size can reduce variance, it can also create data inefficiencies.

This is because, for each new policy, a new trajectory must be generated. The sampled data for each trajectory is then discarded after a single gradient update. Considering the large amount of training steps which may be required for convergence of a model, this inefficiency can be significant and is compounded by any increase in batch size. PPO's solution to this builds on the idea of the replay buffer commonly implemented in Deep Q Networks (DQN) which allows for the use of data from previous trajectories to inform an agent's actions in the current episode (Mnih et al., 2013). This is not possible for on-policy networks such as Actor-Critic models (A2C) etc. where we obtain the gradient for each individual policy and distributions across separate trajectories may be too dissimilar to produce any meaningful benefits. PPO instead uses importance sampling to estimate the distribution of samples from the current distribution using samples from a previous distribution, weighted by the ratio of the two functions (p and q):

$$E_{x\sim p}[f(x)] = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right] \tag{1}$$

This is then combined with the objective function to produce a surrogate objective function which re-uses samples from previous distributions:

$$J(\theta) = E_{(s_t,a_t)\sim\pi_{\theta_{old}}}\left[\frac{\pi_\theta(s_t,a_t)}{\pi_{\theta_{old}}(s_t,a_t)}A(s_t,a_t)\right] \tag{2}$$

However, if $\frac{p(x)}{q(x)}$ produces a value distant from 1 - thereby indicating a high variance between the policy functions, more data may still need to be sampled from the current distribution.

Another problem PPO solves is the unstable updates that can occur within PG learning. This originates from one of the key differences between PG learning and other supervised machine learning algorithms where the samples for each episode are generated based on the previous policy. This means that, while too large an update step in the stochastic gradient descent (SGD) process can be easily corrected by subsequent updates based on the existent, labelled data in most supervised learning models, this is not the case for PG learning. Instead, a large update that generates a bad policy will then be used to generate/collect bad samples and this will result in a worse policy. Conversely, too conservative an update will lead to slower training times and convergence (Lapan, 2018).

PPO uses a form of Trust Region Policy Optimization (TRPO) to combat this. TRPO uses the KL Divergence between distributions to produce a constraint in order to make confident updates (John Schulman, Apr 2017):

$$\begin{aligned}&\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t}\hat{A}_t\right]\\&\text{subject to } \hat{\mathbb{E}}_t[KL[\pi_{\theta_{old}}(\ldots|s_t),\pi_\theta(\ldots|s_t)]] \leqslant \sigma\end{aligned} \tag{3}$$

This can also be represented as a penalty (β) in Lagrangian Dual form:

$$\underset{\theta}{\text{maximize}}\,\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t}\hat{A}_t - \beta KL[\pi_{\theta_{old}}(\ldots|s_t),\pi_\theta(\ldots|s_t)]\right] \tag{4}$$

Crucially, PPO does not use a hard constraint and instead uses a proximal penalty for divergent functions. This resolves the expensive Hessian Matrix computations that result from using TRPO in gradient descent and also negates the difficulty of finding an adaptive penalty (β) which optimizes results across multiple problems or a single, complex problem.

Finally, PPO aims to combat unstable updates by eliminating invariance. Invariance occurs when the difference between the output of the new policy function (π_θ) and the old policy function ($\pi_{\theta_{old}}$) is great and the value produced by this ratio within the above equation changes too quickly. PPO sets a limited range for this value (represented by $r_t(\theta)$) so that $1-\epsilon \leqslant r_t(\theta) \leqslant 1+\epsilon$, clipping the objective accordingly (Yu, 2018).

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = E_{r\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min\begin{pmatrix} r_t(\theta)\hat{A}^{\pi_k}_t, \\ \text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\hat{A}^{\pi_k}_t\end{pmatrix}\right]\right] \tag{5}$$

$$\text{where } r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{6}$$

PPO is commonly used in Actor-Critic (A2C) networks in the following form:

$$L^{CLIP+VF+S}_t(\theta) = \hat{\mathbb{E}}\left[L^{CLIP}_t(\theta) - c_1L^{VF}_t(\theta) + c_2S[\pi_\theta](s_t)\right] \tag{7}$$

Where, respectively, the three terms correspond to the surrogate objective function, a squared-error loss function for the "critic" and an entropy bonus to ensure sufficient exploration.

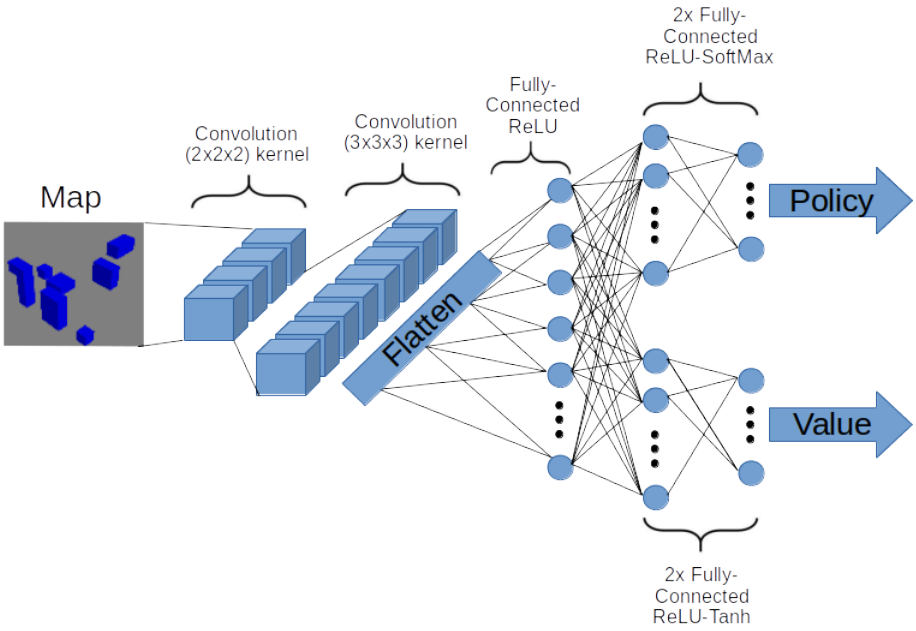


Figure 1. Network Structure: The map propagates through two convolutional layers followed by a fully-connected layer before proceeding to go through the actor and critic network to produce the policy and evaluation respectively

3.2. Environment

We define the action space $\mathcal{A} \equiv \mathbb{Z}/12\mathbb{Z}$ to be discrete and include 12 directions of movements. More specifically, each movement is an agent shift by one pixel. This includes all diagonal, horizontal and vertical movements.

The observation space $\mathcal{O} \equiv \mathbb{Z}_{+}^{n\times m\times l}$ is defined by the respective map data set that is generated beforehand. The target is always positioned on the most outer pixel, i.e. the position $t \equiv (n, m, l)$.

The reward function is defined as

$$R(S) = \sum_{s\in S}[100 \cdot a(s) + 20 \cdot b(s) - 5 \cdot c(s) - 5 \cdot d(s)] \tag{8}$$

$$\text{where } a(s) = \begin{cases} 1 & \textbf{if } s \textbf{ is terminal} \\ 0 & \textbf{otherwise} \end{cases} \tag{9}$$

$$b(s) = \begin{cases} 1 & \textbf{if } s \textbf{ is on optimal path} \\ 0 & \textbf{otherwise} \end{cases} \tag{10}$$

$$c(s) = \begin{cases} 1 & \textbf{if } s \textbf{ is in collision with obstacle} \\ 0 & \textbf{otherwise} \end{cases} \tag{11}$$

$$d(s) = \|s - t\|_2 \tag{12}$$

We say that S is the set of particle states, i.e. the cartesian pixel coordinates of each particle. A state s is terminal if it equals the predefined terminal state t. A state is on the optimal path if its coordinates coincide with any of the coordinates in the set of points that make up the optimal path resulting from the pathfinding module. The state is in collision with an obstacle if its coordinates define a pixel with value one. Particles that were successfully delivered to the target become static and are not affected by control commands anymore. The episode terminates when all particles have reached the target state or when the maximum amount of steps in an episode is reached, i.e. $n + m + l + 20$.

We generate the initial set of states via

$$S \overset{i.i.d}{\sim} \mathfrak{S}^3(\{0, 1, 2, ..., k\}), \tag{13}$$

i.e. S is sampled from the permutation group $\mathfrak{S}^3$. The state $(0, 0, 0)$ is always in S and we pre-define $k \in \mathbb{N}$. Experiment 4.3 uses $k = 3$ while the rest of the experiments choose $k = 2$.

3.3. Network Structure

Our implementation utilizes three separate networks. The first network will be referred to as the feature detection network. The other two networks are respectively the actor and the critic. The feature detection network consists of two consecutive three-dimensional convolutional layers, interleaved by ReLUs. Said network terminates in a flattening layer followed by a fully-connected layer. The stride is generally kept at one. The network propagation is also unpadded. The first convolutional layer scales up its one input channel to four output channels. These are doubled with the second convolutional layer. The final output is a feature vector of size 32 for the baseline system (4.1).

Both the critic and actor use the feature vector produced by the feature detection network and propagate it respectively each through a feedforward network consisting of two fully-connected layers that are interleaved by ReLUs.

Here, the original feature vector is steadily reduced. First from 32 down to 16 and then down to one for the critic and $|\mathcal{A}|$ for the actor. The critic network terminates with a Tanh-activation function while the actor-network ends in a Softmax-layer. We choose to use the Tanh-activation for the critic to prevent the evaluation score from diverging. The Softmax-layer in the actor is used to yield the policy probabilities for the respective actions. The ensemble of networks is displayed in Figure 1. For the experiments that follow on from the baseline system (4.1), we doubled the number of output channels for the critic and actor-network. The feature vector, coming from the feature detection network, was also doubled in size to compensate for the increased map size.

4. Experiments

4.1. Experiment 1 - Baseline System

Motivation: The purpose of this experiment is mainly to find sensible parameter configurations for other experiments to take place at a larger scale. It also provides sample data on performance. The goal of subsequent experiments will be to keep close to this baseline performance.

Configuration: The agent was trained over 30000 episodes on maps of size $n = m = l = 10$. The maps have up to 8 obstacles each occupying up to three connected pixel. Each episode had a length of 50 steps, i.e. 50 pixel shifts, for two particles. The maps do not guarantee that the agent can reach the target configuration without any collisions. We set the reward decay at $\gamma = 0.99$, the learning rate at $\alpha = 8 \cdot 10^{-5}$ and $\epsilon = 0.2$ as recommended in the original paper (John Schulman, Aug 2017). The rewards are normalized. The loss coefficients, in accordance with equation 7, are chosen to be $c_1 = 0.5$ and $c_2 = 0.002$.

Results: As Figure 2 shows the accumulated reward per episode increases quite rapidly within the first 500 episodes. This increase is especially attributed to the initial jump in the number of target hits as seen in Figure 3. The number of collisions also increase initially before slowly beginning to drop. The number of target hits converges to 100%. At the end approximately 70% of the multi-particle paths remain collision-free. It is unclear whether this is the minimum as we do not guarantee a collision-free reaching of the goal state. The loss rapidly increases in the first 1000 episodes. The critic loss seems to be substantially increasing proportionally to the policy loss. Another apparent feature of the loss graph in Figure 4 is that it oscillates quite heavily.

Discussion: The heavy oscillations in the loss average as seen in Figure 4 might indicate that the entropy regularization coefficient, i.e. $c_2 = 0.002$, is chosen to be too high. One can also assume that the initial increase of the loss is due to widely incorrect evaluations performed by the critic. One can additionally say that the loss is far from converging, meaning that a longer training time could be considered viable.

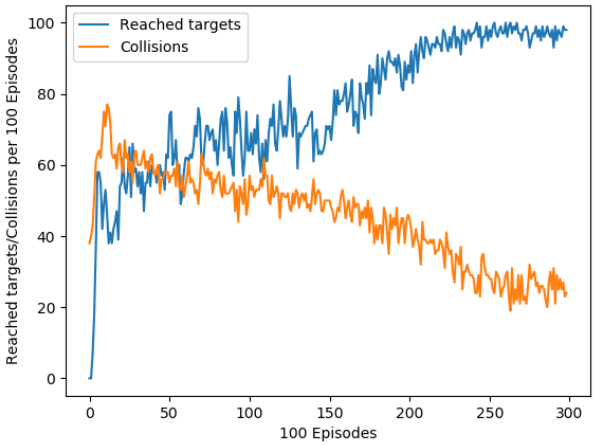


Figure 3. Number of reached targets, i.e. the number of times all particles reached the target position, against the number of collisions with obstacles per 100 episodes.

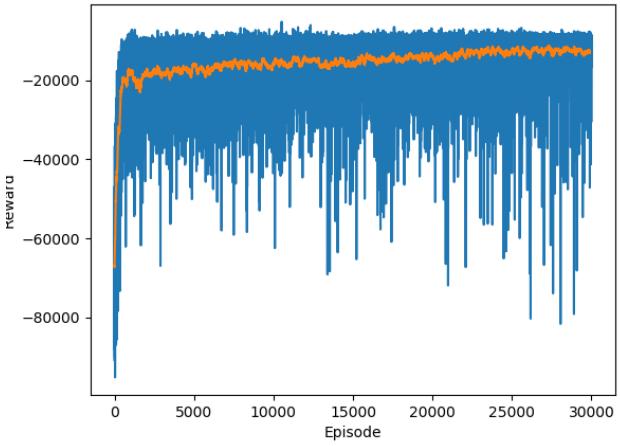


Figure 2. Reward accumulation throughout each episode: The orange graph is the running average over the past 100 episodes. The blue graph is the actual observed reward per episode.

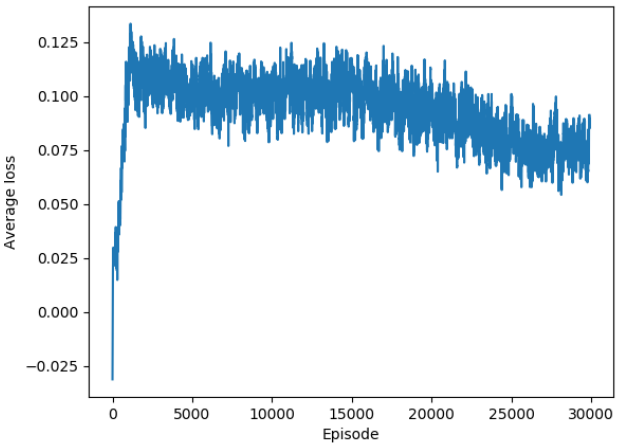


Figure 4. Average loss over the past 100 episodes in experiment 1, i.e. $L^{CLIP+VF+S}$ over the past 100 episodes.

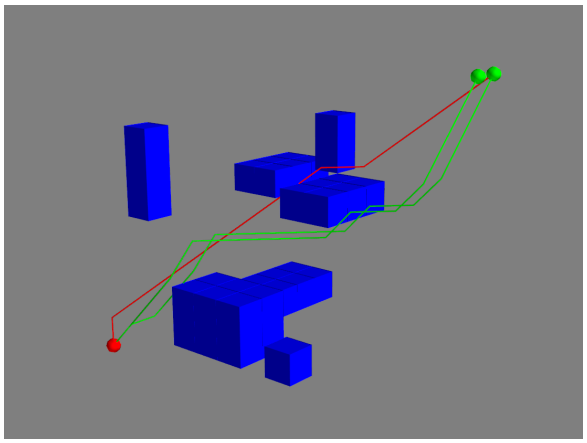


Figure 5. Example of model output in 10^3 space where the red path is the optimal path (A*), terminating at the sphere. and each particle's path is shown in green from the respective start point indicated by a sphere.

4.2. Experiment 2 - Environment Scale

Motivation: The purpose of this experiment is to check if our network is scalable and giving good results for a larger map environment.

Configuration: The agent was trained over 160000 episodes on maps of size $n = m = l = 20$. Since larger maps are used than the ones in experiment 1, more obstacles are needed for creating similar challenges. Therefore, the maps have up to 15 obstacles each occupying up to 4 connected pixels. For similar reasons, the number of particles was also increased to 3. Each episode has 100 steps. The same model parameters as the experiment 1 are used except the regularization coefficient which is chosen to be $c_2 = 0.0008$.

Results: Figure 7 shows the number of reached targets and the collisions. Figure 8 shows the loss function. The trend of these values are similar to the baseline as expected.

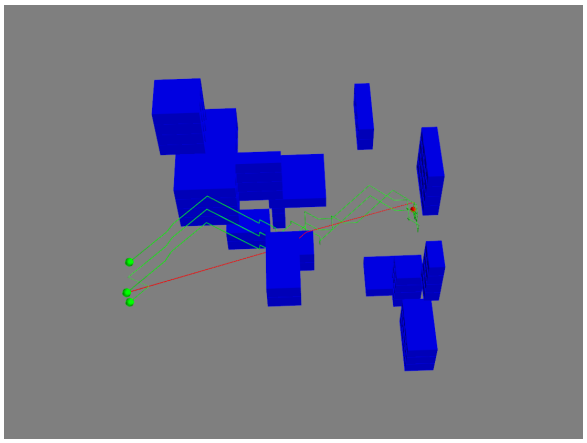


Figure 6. Example of model output in 20^3 space where the red path is the optimal path (A*), terminating at the sphere. and each particle's path is shown in green from the respective start point indicated by a sphere.

Discussion: Since the environment is larger in this experiment the training needs more episodes. Hence, it took 160000 episodes to converge for the number of reached targets/collisions. The trends are pretty similar to the baseline. It shows that the model can be scaled for the larger maps with more training.

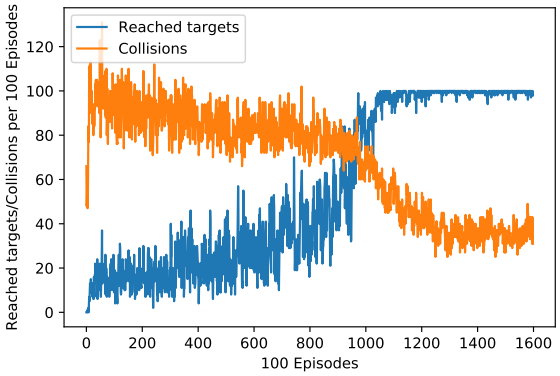


Figure 7. Number of reached targets, i.e. the number of times all particles reached the target position, against the number of collisions with obstacles per 100 episodes.

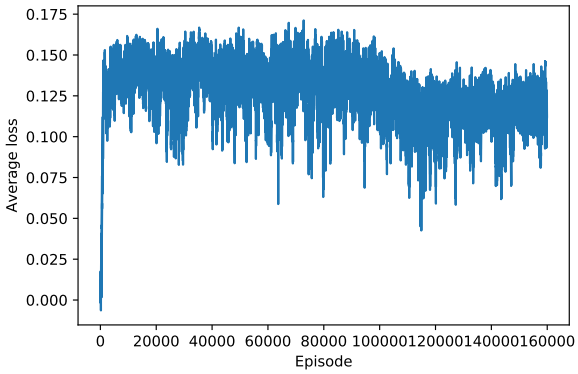


Figure 8. Average loss over the past 100 episodes in experiment 2, i.e. $L^{CLIP+VF+S}$ over the past 100 episodes.

4.3. Experiment 3 - Swarm Size

Motivation: This experiment was designed to measure the agent's ability to successfully guide particle sets of various sizes to the goal. While the previous experiments had demonstrated the agent's ability to navigate 2 or 3 particles simultaneously, these scenarios were designed to investigate scalability with respect to this parameter. This is useful to evaluate the system's viability for use in different domains such as drone control, where a large number of particles is unlikely, to biomedical agents which may have swarm sizes that are several orders of magnitude larger.

Configuration: All map generation parameters and model properties remained as in experiments 1. The only altered parameter was the initial particle set size for the agent during training. This was increased to 5 and then again to 10.

Results: Figures 7 and 8 show the number of reached targets and the collisions for a particle set size of 5 and 10 respectively in a 10^3 space. Loss and reward values were consistent with those from the baseline systems and all results were similarly replicated for the larger 20^3 environment. These have been omitted for brevity.

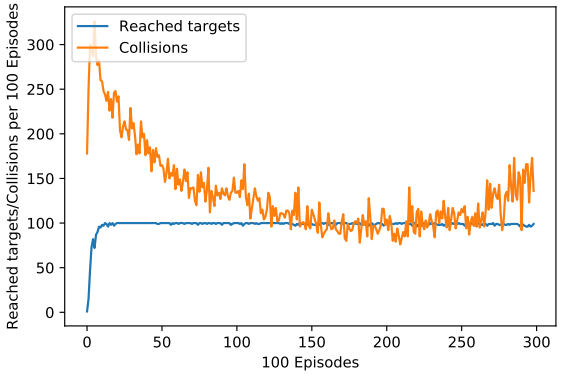


Figure 9. Number of reached targets vs. collisions for 5 particles in a 10^3 sized environment

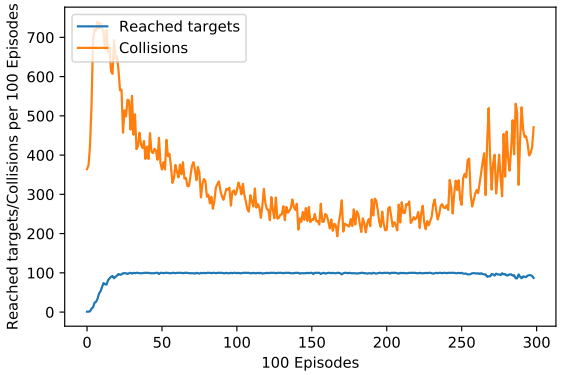


Figure 10. Number of reached targets vs. collisions for 10 particles in a 10^3 sized environment

Discussion: The results show that the initial time for the model to successfully converge, resulting in all agents reaching the goal, is increased relative to the particle set size. Both increases in the number of particles does not result in a loss of stability from the model once it has converged, the average number of episodes where all agents reached the goal remains relatively consistent at 100. This stability appears to be impacted to some degree by the average number of collisions as figure 10 demonstrates some fluctuations in this value as more collisions are introduced.

The number of collisions does appear to destabilise around episode 22500, indicating the agent has reached its minimum for the specified parameters and that its performance with respect to collision avoidance is now worsening. This would appear to be a result of over training since convergence is seen quickly in the smaller environment. Both results indicate a linear relationship between the average number of collisions per episode relative to the number of particles. This is true for both the initial values and minimal values which appear to double as the particle set size does. This shows that model require tuning to encourage better obstacle avoidance as, if the number of collisions can destabilise the agent's ability to reach the goal position, then convergence may be impaired as the model scales to larger swarms.

	Particles	Environment	
		10x10x10	*20x20x20*
Agent	*2*	0.0299	0.0598
	3	0.339	0.7380
	5	0.0738	0.1416
	10	0.1586	0.1675
	20	0.2114	0.2124
Dijkstra		0.2581	0.9271
A*		0.1915	0.3001

Table 1. Pathfinding runtimes in seconds for PPO agent vs. classical methods (single particle)

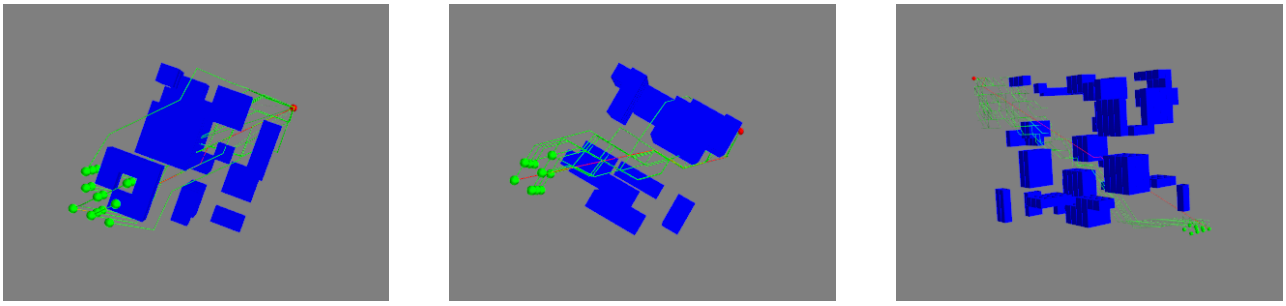
4.4. Experiment 4 - Computational Efficiency

Motivation: Previous experiments have shown the performance of our model in terms of successful path-finding and collision avoidance. In order to investigate other potential benefits of a trained path-finding agent versus classical path-finding methods, the runtimes for each were compared under a variety of conditions. This is particularly relevant in a three-dimensional space since the computational complexity of traditional graph search algorithms exponentially increases, proportional to the search-space dimensions.

Configuration: The pre-trained, 10-particle models from experiment 3 were used for each environment. Map conditions were again duplicated from experiment 1 and 2 for their respective environments. All other parameters remained unchanged. The agent then predicted a path for each map for either 2,3,5 or 10 particles. The average runtime of 100 predictions from the agent is shown in table 1. For comparison, the average runtimes of Dijkstra and A* for the same data but only a single particle are displayed.

Results: Table 1 shows the average runtime in seconds for each method, particle set size and environment described previously. All results were obtained from the same 100 maps.

Discussion: These results show that the proposed model offers significant improvements in runtime over classical methods, even when generating paths for larger particle numbers. Our agent is shown to reduce runtimes by up to



(a) 15 particles in 10^3 space with high occupancy near start

(b) 10 particles in 10^3 space with high occupancy near goal

(c) 20 particles in a 20^3 crowded space

Figure 11. Results from experiment 4/5 demonstrating the agents performance on increased particle set size and map occupancy

84.39% and 88.42% for A(*)/Dijkstra respectively in a 10^3 environment. This is replicated as the graph space increases, showing an 80.07% reduction for A* but a 93.54% reduction in search time compared to Dijkstra. Those figures demonstrate the improvement when the agent operates with the minimum number of particles (2) but, even at higher numbers, the agent shows a significant reduction in runtimes compared to those for a single particle, computed using classic, optimal path finding methods. At 20 particles the agent still achieves a reduction in search time in all but one instance. For 10^3 environments, the agent showed an increased runtime of 10.39% versus A*. This was contrasted by an 18.1% improvement over Dijkstra in the same environment and improvements of 29.22% and 77.09% in a 20^3 space for each method respectively.

In all of these instances the agents successfully navigated all particles to the goal despite the fact that, in the case of the 20 particle set, the agent had not been trained with particle sets of this size. This appears to show a robust and scalable learning of behaviour by the agent which is able to generalise to successfully accommodate new parameters and data. A similar example is shown in experiment 5 of this behaviour with regards to map environments/obstacles.

4.5. Experiment 5 - Crowded Environments

Motivation: This experiment was designed to measure the agent's ability to adapt to changing environments. This was motivated by the results of experiment 4 which showed the agent's was able to successfully handle an increase in particle set size beyond its trained parameters. To test if the agent was capable of dealing with other parametric changes, more challenging environments were introduce to agents that had not been trained on such data.

Configuration: The same two pre-trained models from experiment 4 were used with various swarm sizes between 10 and 20. The parameters for obstacle generation within in the maps were altered, increasing the number of obstacles to 15 for 10^3 environments and 38 for 20^3 environments. The maximum obstacle size was also increased to 4 and 5 connected pixels for each environment.

Results: Figure 11 illustrates several examples of the agent's performance in challenging environments regardless of particle set size. These results were observed broadly across a multitude of episodes under the parameters described previously. The samples chosen reflect the most commonly observed behaviour of the agent for these conditions.

Discussion: The results in figure 11 show examples of the agent successfully navigating maps of a crowded nature. These results indicate that even in challenging environments which it had not been subjected to during training, the agent was able to navigate all particles to the goal, while avoiding obstacles. Figure 11(a) and 11(b) demonstrate this to be the case even when the start and goal states are heavily occluded for some particles. Figure 10(c) shows that this was also seen to be the case in highly occupied environments for of size 20^3. These results support the evidence from previous experiments that indicate the agent is able to traverse significantly different environments to those it was trained on, either in terms of particle set size or in terms of the navigational difficulty of the environment.

5. Related Work

This paper is only loosely tied to existing research. Nonetheless, some articles may inspire improvements to the algorithms used in our project. The following two paragraphs present ideas on how to improve the generalised concept of the algorithm used in our project and how to increase its affinity in relation to our application in focus.

One of the biggest, remaining challenges in reinforcement learning involves the translation of algorithms to continuous action and observation spaces. In recent years significant advancements have been made. There are various papers that can serve as paragons for proceeding research (Zeng, Dec 2018) (William Koch, Apr 2018) (Sang-Yun Shin, Oct 2019). The paper by Zeng illustrates how the translation problem can be approached for mobile robots in two-dimensional environments. For the three dimensional case it is sensible to segment the actual planning

and control task into different tasks. We can divide the problem into a problem of determining the attitude of the particles and the specifying the magnitude of the movement. A helpful start is provided by William Koch with UAV attitude control. The paper uses various algorithms such as DDPG, TRPO and PPO to determine appropriate attitudes for three dimensional drone control. Lastly, Sang-Yun Shin presents multiple ways in which obstacle avoidance can be improved. A big part of their work concentrates on the setup of U-net based segmentation on RGB and depth maps. It offers a promising new approach on how to deal with more complex visual input.

There are several ways to proceed with the presented work, here. This paper has its target position fixed at (n, m, l). The sensible expansion would be to randomize the target positions. This might require a deeper network structure and a remodelling of the reward function. As intravenous drug delivery control (Ajeet Kaushik & Nair, Jul 2014) was the application in focus, it would be reasonable to continue with an application specific and more accurate model of the environment. This means that the controls should be expressed via changes in a magnetic field. Further, the environment should reflect the intravenous landscape of a human. This also means that perturbations such as blood flow should be included into the model. The next step would be to translate the approach to high resolution models, effectively increasing map size and detail. With the increased map size nano-carriers would also be modelled as patches rather than particle swarms. It would be useful to find a way to efficiently sample computer tomography images in such a way that it can be put through the algorithm.

6. Conclusions

We settled on PPO as our algorithm of choice as it has a reduced set of parameters compared to other methods such as TRPO or even DQN. It is less sensible to the parameter changes. This trend was confirmed in 4.1 as good results were easily achieved. It is also important to point on the algorithms dependency on the formulation of the reward function. We observed that the initial gain in cumulative reward was mostly attributed to the increased number of particle swarms reaching the goal. This is heavily influenced by the make up of the reward function as it puts more emphasis on the distance minimisation than obstacle avoidance. The most significant improvements occurred in the very first few episodes but PPO displayed its ability to continuously yield improvements until the end of the training session. As the original paper (John Schulman, Aug 2017) promises, we can confirm PPO's ability to avoid large oscillations in its reward accumulation graph. It fixed the high variance issue that other on-policy methods suffer from.

As Experiment 4.2 shows the algorithm was able to cope well with bigger maps. The training phase took significantly more time and the number of training episodes need to be increased to 160000. The algorithm behaved similarly to the baseline. We expect the algorithm to scale well with even larger maps.

The algorithm's ability to scale well translates to larger swarm sizes as Experiment 4.3 proves. As to be expected we reported an increased number of collisions. It showed tendencies of divergence towards the end of the training session. Possible approaches to fix this include a stricter measures of regularization. Reasonably we could apply a stronger entropy regularization or experiment with an increased weight decay. As the policy optimisation works via Adam, a smoothing of the reward function might also be an appropriate action to take.

In experiments 4.4 and 4.5 the model was shown to be able to generalise well when subjected to new conditions such as an increase in swarm size and map occupancy greater than any it had encountered during training. Experiment 4.4 also demonstrated the drastically reduced path finding runtime offered compared to classical methods. This is despite the fact the classical methods utilised were constrained to a compute only a single path whereas the agent was restricted to a minimum particle set size of 2. During experiment 4.5, the model exhibited encouraging behaviours and showed an ability to handle challenging scenarios despite its vastly reduced computational cost.

Our implementation presents the fundamental concepts required to design a three-dimensional particle swarm navigator. The experiments prove that our work is also a reasonable baseline to build future implementations on. We have given the reader many ideas on how to continue research and hope that this paper inspires new application-based path-planning algorithms.

References

Ajeet Kaushik, Rahul Dev Jayant, Vidya Sagar and Nair, Madhavan. The potential of magneto-electric nanocarriers for drug delivery. *Taylor & Francis Online*, Jul 2014. URL https://www.tandfonline.com/doi/abs/10.1517/17425247.2014.933803?journalCode=iedd20.

Dijkstra, Edsger W. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Hart, Peter, Nilsson, Nils, and Raphael, Bertram. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/tssc.1968.300136. URL https://doi.org/10.1109/tssc.1968.300136.

John Schulman, Sergey Levine, Philipp Moritz Michael Jordan Pieter Abbeel. Trust region policy optimization.

arXiv:1502.05477v5, Apr 2017. URL https://arxiv.org/pdf/1502.05477.pdf.

John Schulman, Filip Wolski, Prafulla Dhariwal Alec Radford Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347v2*, Aug 2017. URL https://arxiv.org/pdf/1707.06347.pdf.

Lapan, Maxim. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Packt Publishing, 2018. ISBN 1788834240.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. 2013. URL http://arxiv.org/abs/1312.5602. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

Sang-Yun Shin, Yong-Won Kang, Yong-Guk Kim. Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot. *MDPI*, Oct 2019. URL https://www.mdpi.com/2076-3417/9/24/5571/htm.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel Nicolas Heess Tom Erez Yuval Tassa David Silver Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971v6*, Jul 2019. URL https://arxiv.org/pdf/1509.02971.pdf.

William Koch, Renato Mancuso, Richard West Azer Bestavros. Reinforcement learning for uav attitude control. *arXiv:1804.04154v1*, Apr 2018. URL https://arxiv.org/pdf/1804.04154.pdf.

Yu, Ruifan. 15b: Proximal policy optimization, cs885. *University of Waterloo*, 2018. URL https://cs.uwaterloo.ca/~ppoupart/teaching/cs885-spring18/slides/cs885-lecture15b.pdf.

Zeng, Taiping. Learning continuous control through proximal policy optimization for mobile robot navigation. *ResearchGate*, Dec 2018. URL https://www.researchgate.net/publication/330105099_Learning_Continuous_Control_through_Proximal_Policy_Optimization_for_Mobile_Robot_Navigation/link/5c2dd53792851c22a356bead/download.