Single-Agent Policies for the Multi-Agent Persistent Surveillance Problem via Artificial Heterogeneity

Thomas Kent$^{1[0000-0002-7294-1702]}$, Arthur Richards$^{1[0000-0001-9500-5514]}$, and Angus Johnson^{2}

^{1} University of Bristol, Bristol, UK
{thomas.kent, arthur.richards}@bristol.ac.uk
^{2} Thales UK, Reading, UK
angus.johnson@uk.thalesgroup.com

Abstract. Modelling and planning as well as Machine Learning techniques such as Reinforcement Learning are often difficult in multi-agent problems. With increasing numbers of agents the decision space grows rapidly and is made increasingly complex through interacting agents. This paper is motivated by the question of if it is possible to train single-agent policies in isolation and without the need for explicit cooperation or coordination still *successfully* deploy them to multi-agent scenarios. In particular we look at the multi-agent Persistent Surveillance Problem (MAPSP), which is the problem of using a number of agents to continually visit and re-visit areas of a map to maximise a metric of surveillance. We outline five distinct single-agent policies to solve the MAPSP: Reinforcement Learning (*DDPG*); Neuro-Evolution (*NEAT*); a Gradient Descent (*GD*) heuristic; a random heuristic; and a pre-defined 'ploughing pattern' (*Trail*). We will compare the performance and scalability of these single-agent policies to the Multi-Agent PSP. Importantly, in doing so we will demonstrate an emergent property which we call the *Homogeneous-Policy Convergence Cycle* (HPCC), whereby agents following homogeneous policies can get stuck together, continuously repeating the same action as other agents, significantly impacting performance. This paper will show that just a small amount of noise, at the state or action level, is sufficient to solve the problem, essentially creating artificially-heterogeneous policies for the agents.

Keywords: Multi-Agent Systems, Reinforcement Learning, Surveillance, Coverage, Emergent behaviour

1 Introduction And Background

Real-world problems such as reconnaissance and surveillance [13,14], search and rescue [9] and, drone crop-monitoring [11] rely on efficient and continuous ways of visiting areas of the world. For problems covering large areas or when higher-frequency monitoring is desirable it can be useful to deploy multiple agents, or even swarms [1], in an environment. Modelling and planning for multi-agent problems can often be difficult due to a rapidly growing decision space, made increasing complex through the interacting agents [17]. Additionally, this can result in a need for coordination and communication that may not be possible in many situations. Many Unmanned Aerial Vehicle (UAV) platforms and off-the-shelf solutions are designed in isolation and typically offer only single-agent behaviours. Unless agents have been designed for multi-agent settings or can be coordinated via some centralised control, then policy homogeneity might be unavoidable. However, as we will demonstrate in this paper this can lead to undesirable emergent properties.

The aim this paper is to explore the concept of using single-agent policies, designed and/or trained in isolation, that can be *successfully* deployed in a multi-agent scenario. In particular we focus on the multi-agent Persistent Surveillance Problem (MAPSP), as a simplified use-case pertinent to multi-agent systems research. The MAPSP is the problem of using a number of agents to continually visit and re-visit areas of a map in order to maximise a metric. We outline, in Section 3, a range of different action policies, for agents to decide the best action to take given an agent-centric local observation. Policies include 1) *Random*; 2) *Gradient Descent*; 3) *DDPG*; 4) *NEAT* and 5) Trail-Following. In particular, in Section 3.2, we will demonstrate that by deploying *homogeneous* single-agent policies in a multi-agent setting can lead to a highly undesirable emergent property that we call the 'Homogeneous Policy Convergence Cycle' (HPCC). Each of the policies will be evaluated for varying numbers of agents and the HPCC problem will be demonstrated. Finally approaches to counteract the HPCC will be discussed where we will show that by essentially making the agents less homogeneous, via the addition of noise, is sufficient to fix the problem.

2 Persistent Surveillance Problem

The Persistent Surveillance Problem (PSP) belongs to a class of problems known as *coverage problems* [4, 5]. The aim of Persistent Surveillance is to continually visit and re-visit all areas of a map in order to maximise a surveillance score that sufficiently quantifies performance. To measure this a 2-Dimensional world is divided into regions using a hexagonal-grid structure of N 'hexes', with each of these hexes having an associated surveillance score (as depicted in Figure 1). The total surveillance score, i.e. the PSP objective, is the sum of all the scores across all the hexes. In this paper we have designed a score-function that quantifies a notion of 'level of surveillance' and its subsequent exponential decay by a relationship dictated by the hex-score function:

$$\mathcal{V}(h^i_{t+1}) = \begin{cases} \mathcal{V}(h^i_t) + C, & \text{if } h^i_t \text{ occupied.} \\ \mathcal{V}(h^i_t)\lambda, & \text{otherwise.} \end{cases} \tag{1}$$

This function defines that when an agent is in a hex, h^i, it increases that hex's score, $\mathcal{V}(h^i)$ by a positive linear constant, C, each time-step it occupies the hex.

Then for any hex unoccupied the score $\mathcal{V}(h^i)$, decays exponentially by a factor $\lambda = (1/2)^{(dt/T_h)}$, which is a constant parametrised by its 'half-life' value T_h such that $\lambda \leq 1$. The score of each hex is also bounded, restricting $\mathcal{V}(h^i_t) \in [0, v_{\max}]$.

We define the set $H_t = \{h^i_t \text{ for } i = 1..N\}$ to be the set of all N hexes at time t. Thus the total *surveillance score*, at time t, is sum of all the scores of H_t, that is

$$\mathcal{V}(H_t) = \sum_{\forall h^i_t \in H} \mathcal{V}(h^i_t). \tag{2}$$

The aim of the PSP is to keep the score $\mathcal{V}(H)$ as high as possible at all times, with the max surveillance score defined as

$$\mathcal{V}^*(H) = \max_{\forall t \in T}(\mathcal{V}(H_t)). \tag{3}$$

2.1 PSP Simulation Environment

The Persistent Surveillance Problem has been implemented as an environment in-line with the OpenAI Gym [3] and follows the traditional State, Action, Reward sequence [15]. The Multi-Agent Simulator (MAS) environment keeps track of agent locations and their underlying states. The MAS can be queried for a state (observation) of any agent and in turn the agent can carry out an action in the environment and the MAS simulates the outcome, and returns a reward. In the case when there are multiple agents, all observations and actions happen simultaneously. In addition we assume that agents are unable to communicate with each other and do not attempt to plan for other agents. The environment is defined over the bounded 2-Dimensional world W. All agents are restricted to stay within W, with their motions being 'clipped' at the simulator level ensuring they can only move within the bounds.

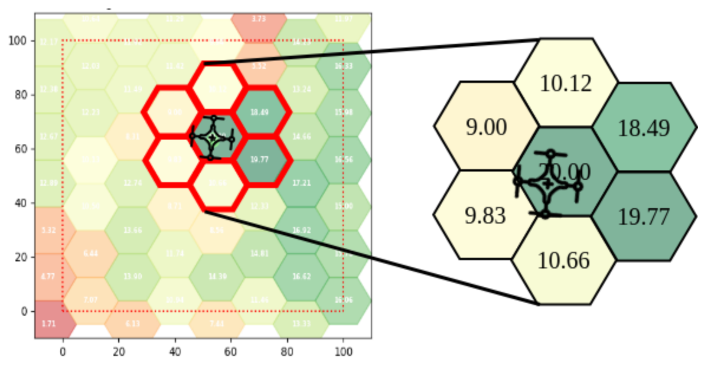


Fig. 1: Agent-centric observation, higher-values green, lower are red, $s^k_t = [20.00, 9.83, 10.66, 19.77, 18.49, 10.12, 9.00]$

The specific environmental parameter choices for the results in this paper are as follows. The world, $W = [0, 100] \times [0, 100]$ and is made up a grid of $N = 56$ hexagons each with a height of $15m$ (flat edge-to-edge). The simulation updates at a discrete time-step dt, updating all $h^i \in H$, using Equation (1) with a half-life decay $T_h = 120$, a linear increase $C = 5$ and a maximum value of $v_{\max} = 20$. To help with agent training and performance we also use a technique called frame-skip, (popularised from Deep Learning techniques designed to play Atari games [10]) which limits the rate of decision making to once every $3dt$. Thus an agent receives an observation every third time-step, acts on it and this action is held for $3dt$. Furthermore we define each agent velocity as $5m/dt$, and so every action results in the agent moving $3 \times 5 = 15$ meters across the world.

2.2 Local Observations

The PSP simulation environment keeps track of the current global state of the world via the scores of each hex, using the current values, the agent locations and Equation (1). Agents deployed in the environment will be given access to these states via *observations*, as depicted in Figure 1, which will be agent-centric, i.e. dependent on the current agent location. The policies outlined in this paper are given only these *local observations* of the state, with which to make their action decision (with the exception of *Trail* which requires a degree of global-localisation to stay on course).

We define the set of hexes directly adjacent to h^i as

$$\text{hexAdj}(h^i) = \{\forall h^j \in H | h^j \text{ adjacent } h^i\}, \tag{4}$$

which is shown in Figure 1 with h^i in the centre. Then, given an agent k, currently located in hex h^i, the agent's observation $s^k(h^i)$ (we choose to shorten the notation to simply s^k when it is appropriately clear), is defined as

$$s^k(h^i) = \{\mathcal{V}(h) \text{ for } h \in \{h^i \cup \text{hexAdj}(h^i)\}. \tag{5}$$

It is the role of each policy to use this local-state observation, s^k, to decide the best action to take in the environment.

While our agents are restricted to the bounded world W they may observe hexes outside of this, in which case those hexes are declared 'obstacle hexes'. Any obstacle hex, h^{obs}, has its observation value set to v_{obst} which is a value greater than $v_{\max}$ but with $\mathcal{V}(h^{\text{obst}}) = 0$ so as not to contribute to the score. Additionally, in the multi-agent scenarios any observation which contains another agent is also declared h^{obst} and set to v_{obst}. Note that a hex h^{obst} does not physically restrict an agent from moving into it, instead its aim is to deter agents moving there by being a high value while not contributing to the score and in turn not affecting the reward.

2.3 Action Policy

An agent k's decision making is contained entirely within its policy function π^k. The role of the policy to take an observation, s^k, and provide an action,

$a^k \leftarrow \pi(s^k)$, from the set of all possible actions $\mathcal{A}$. In this paper the action, $a^k \in [-1,1] \times [-1,1]$, is the two-dimensional trigonometric-encoding of direction in which to travel, that is, the agent heading, θ, encoded by the two values $a^k = [\sin(\theta), \cos(\theta)]$ (with $\theta = 0$ corresponding to East).

Some of the policies outlined in Section 3.1 use a discrete action policy, whereby an action is chosen from a set, $a^k \in \mathcal{A}_\mathrm{d}$, of 6 possible hexagonal directions, equivalent to the angular encoding $\theta \in \left[\frac{7\pi}{6}, \frac{9\pi}{6}, \frac{11\pi}{6}, \frac{\pi}{6}, \frac{3\pi}{6}, \frac{5\pi}{6}\right]$. These angles correspond to the angular direction from the centre point of the centre hex h^i to the centre of each of $h \in \mathrm{hexAdj}(h^i)$.

2.4 Reward Function

For policies which require 'training' (i.e. *DDPG* and *NEAT*) we must also define a reward function. At each time, t, an agent k chooses an action, a^k via its policy, π^k, the agent carries out that action in the environment and in return is given a reward, r^k. The Machine Learning (ML) agents, *DDPG* and *NEAT*, of Section 3.1 require this reward to learn, and the reward function itself is a hugely important factor within ML in shaping *how* and *what* an agent learns [8]. As the agents only have access to local observations the reward is restricted to be a function of these only. For an agent k, at time t in hex h^k_t, we define the reward to be how much the state $s^k_t(h^i_t)$ is improved as a result of taking the action, that is:

$$r_{t+1} = \mathcal{V}(s^k_{t+1}(h^i)) - \mathcal{V}(s^k_t(h^i)) \tag{6}$$

Thus the agent gets positive reward if it leaves the $s^k(h^i)$ better than when it arrived. This reward is provided to the agent by the environment as it keeps track of past hex scores. The mathematical incentive of this reward is to ensure that the linear addition of C from Equation (1) is better than the loss via the decay of λ of each of the other hexes. By having the bound of v_max it means that agents do not simply try to move between two hexes, instead they should be moving towards hexes of lower values where there is sufficient 'room' to add value.

2.5 Analytical Assessment

Given a world W, the score function of Equation (1) and a hexagonal-grid structure of the environment we are able to analyse theoretical bounds on PSP problem. For each action-step (i.e. 3 time-steps) the most value an agent can add to a hex is $3 \times C = 15$, and hexes have a maximum value $v_\mathrm{max} = 20$, this means that ideally we want to be moving towards hexes that are ideally less than $v_\mathrm{max} - 3c = 5$.

If we visit each of the $N = 56$ hexes in some sequence, $H_\mathrm{seq} = \{h^0, \ldots, h^n\}$, and spend $3dt$ at each, then each hex will have a score $a_0 = 15$ at the time it is visited and then that value will subsequently decay. The currently occupied hex h^n_t will have the highest value $\mathcal{V}(h^n_t) = a_0$, then the hex visited previously, at the current time-step, h^{n-1}_t, had the same value but has since decayed by λ

and thus $\mathcal{V}(h_t^{n-1}) = a_0\lambda$ and for $n-2$, $\mathcal{V}(h_t^{n-2}) = a_0\lambda^2$. This continues in this fashion until the hex visited n time-steps ago, which has a value $\mathcal{V}(h_t^0) = a_0\lambda^n$. The sum of these scores, and thus the current surveillance score, is in fact a geometric series:

$$a_0\lambda^0 + a_0\lambda^1 + a_0\lambda^2 + \cdots + a_0\lambda^n = \sum_{k=0}^{n-1} a_0\lambda^k = a_0\left(\frac{1-\lambda^n}{1-\lambda}\right) \tag{7}$$

Therefore, visiting each hex once, for $\lambda = (\frac{1}{2}^{\frac{3}{120}})$ the surveillance score reaches $\mathcal{V}(H_{\text{seq}}) = 542$. The value of the hex visited n time-steps ago H_{seq}, $\mathcal{V}(h_0) = a_0\lambda^n = 15 \times 0.379 = 5.684$. Therefore if we continue to visit the hexes in the order of H_{seq}, that is we now visit h_0 again, and its value becomes $5.684 + 3 \times C = 20.684$, which is above $v_{\max}$ and so is capped at 20. Repeating the same logic as before $a_0 \leftarrow 20$ and using Equation (7) the surveillance score becomes $\mathcal{V}(H_{\text{seq}}) = 723$.

In order to achieve these kinds of scores it is necessary to find an admissible sequence H_{seq} that can be visited in that order. This requires a kind of trail, that visits each hex only once and returns to the starting hex, in graph-theory this is known as a Eulerian cycle. Therefore it is clear that in order to maximise the surveillance score it is necessary to find a policy that best approximates this kind of cycle. With this in mind, in Section 3.1 a trail-agent will be outlined, which is able to follow a pre-defined cycle, and used as a benchmark to compare the other agents to.

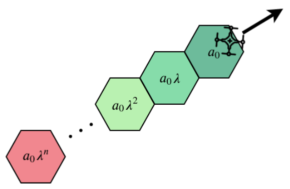


Fig. 2: Ideal hex trail resulting in geometric series

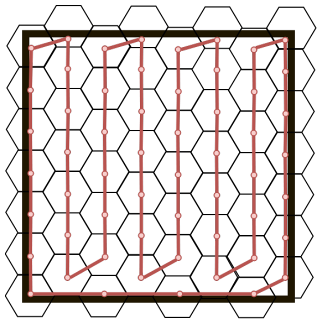


Fig. 3: Pre-defined trail for agent to follow

3 Single-Agent Policies for the PSP

We now outline five single-agent policies designed to solve the Single Agent Persistent Surveillance Problem. The objective of each of these policies will be to achieve the best $\mathcal{V}^*(H)$ during an episode, by deciding, at each step which direction to move around the environment.

3.1 Five Policies for PSP

The first two policies act as *benchmarks* to essentially bound what good and bad performance looks like. The first, *Random*, acts independently of observations and moves randomly and should result in poor performance. While the second, *Trail*, uses the idea of Section 2.5, and follows a pre-defined trail, requiring global information and should result in the best performance.

The remaining three, local policies, act on the local agent-centric observations of the world described in Section 2.2. A simple heuristic policy called *gradient descent* acts on the *model* that moving towards lower values is best. Finally, we use two different 'off-the-shelf' ML algorithms, *DDPG* and *NEAT*, and apply them to our implementation of the Persistent Surveillance Problem.

All of the methods outlined in this paper have been implemented in Python 3.5 and all the simulations are run on a Dell Precision 3520 laptop running Ubuntu 16.04, with a 2.7Ghz core i7 CPU and 16GB of RAM.

Random policy The role of the random agent is to act as a minimum benchmark and is essentially a *blind-policy.* The agent is given an observation, s^k, but is not used in its action selection. The agent simple picks, at random, a discrete direction from one of the six possible, $a^k \in \mathcal{A}_d$. Due to the randomness of this policy, it is non-deterministic.

Trail Policy Benchmark As discussed in Section 2.5 an ideal path to take is one travelling through a sequence of hexes in a Eulerian Cycle. Boustrophedon patterns, also known as ploughing patterns, are some of the best known examples of this approach and have long be used in agriculture [7] and in search and rescue missions [12].

For the purposes of providing a benchmark, a pre-defined trail-based policy is implemented as shown in Figure 3. However, we must note that a direct comparison is not entirely fair as this policy requires a degree of global-state localisation that is not afforded to the local-state policies. An agent running the trail-policy must first work out where it is on the trail in order to determine where to move next. Thus the policy here is to take the trail-point closest to the agent and then select the action in the direction of the next trail-point. An agent will continue around this cycle trying to essentially approximate the geometric sum of Equation (7).

Gradient Descent Policy A Gradient Descent (*GD*) approach is used as a simple yet effective heuristic. This acts as a very simplified *model-based* policy, where the model is 'move towards nearby hexes of lowest value'. That is, choose the hex, $h^{\min} = \text{argmin}(s_t^k)$, with the lowest observation value and move towards it. Thus the discrete action $a^k \in \mathcal{A}_d$ is the one corresponding to $h^{\min}$.

Using the example observation of Figure 1, s_t^k = [20.00, 9.83, 10.66, 19.77, 18.49, 10.12, 9.00], the GD policy would move towards the hex corresponding to minimum value $h^{\min} = 9.00$, i.e. the last value of s_t^k, North-West ($3\pi/6$). In the event of there being more than one minimum value, then one of them chosen at random (this adds a level of uncertainty to this policy).

DDPG Policy A *Tensorflow* implementation of the Deep Deterministic Policy Gradient algorithm [8] is used, which is a model-free, online, off-policy Reinforcement Learning method. It combines the use of an actor-critic architecture along with target-network updates, replay buffers and stochastic exploration. *DDPG* is designed to be used within environments with continuous actions spaces which is why it was chosen for this problem. *DDPG* is a policy gradient method and as such uses a functional approximation, in this case a Deep Neural Network, to the Q-function. RL is used to train these DNNs through standard *SARSA* structured examples [15] to allow us to compute the optimal action, a, to take for a given state s, that is the action which maximises our Q-function, $\max_a \mathcal{Q}^*(s, a)$.

This implementation uses the standard hyper-parameters of the original paper [8]. With an input layer, two hidden layers one of 400 and one of 300 units, and one output layer. A learning rate of 10^{-4} for the actor networks and 10^{-3} for the critic networks. A discount factor γ of 0.99 is used, a target-network update rate of $\eta = 0.01$. The Ornstein-Uhlenbeck process was used for training-noise [18], with noise parameters of $\theta = 0.05$ and $\sigma = 0.05$. The input to the Neural Network is the state observation of the 7 hex values normalised between 0 and 1. Our *DDPG* has two outputs, activated via hyperbolic-tangent functions, giving output values $o_1, o_2 \in [-1, 1]$ which correspond to the action a^k of Section 2.3, $\sin(\theta)$ and $\cos(\theta)$ respectively. The reasoning behind using two output values was to overcome a neuron-saturation [6] issue observed by the authors, a single angle can be easily recovered via the $\theta = \text{arctan2}(\sin(\theta), \cos(\theta))$ function.

The *DDPG* network was trained for 2000 episodes using the reward function of Equation (6) taking approximately 60 minutes, at which point training had sufficiently converged.

NEAT Policy Neuro-Evolution of Augmenting Topologies (*NEAT*) [16] is a method for evolving Neural Networks (NN) via an Evolutionary Algorithm (EA). Here a NN is subjected to an EA process in order to evolve both the structure, the weights and the activation functions of the NN with the key idea of starting by building from small NNs and evolving to add increasing complexity.

The evolutionary approach differs to the standard *SARSA* of RL by way of the Reward. Where, as is the case of *DDPG*, the reward received in RL is at each step, EAs instead use an episodic measure of success known as fitness. Therefore, any network evolved by *NEAT* is evaluated against this fitness value and a selection operator determines whether it is kept for the next generation. The fitness function used in the paper is the cumulative value over the episode of the reward function of Equation (6).

The NNs take the 7 values of s^k as input, are a single-layer deep, and have 6 output nodes with each corresponding to an action direction $a^k \in \mathcal{A}_d$. A soft-max activation function is used to select the highest-valued output neuron, the corresponding discrete action is then returned. *NEAT* is initialised with a population of fully connected single layer NNs with randomised weights and activations. The process of *NEAT* is for each generation to evolve the population of candidate NNs, test them within the PS environment, evaluate their fitness using the cumulative reward function of Equation (6) and select the best candidates to be retained for the next generation. *NEAT* continues this standard evolutionary process until a termination condition is met, in this case 2000 generations. The resulting NN is then the *NEAT* policy, taking in the observation s^k, running it through the NN, and taking the resulting action.

3.2 Multi-Agent Deployment of Single-Agent Policies

To deploy multiple agents within the MAS described in Section 2.1, we take a given policy and deploy copies of it on each agent within the environment. The MAS provides agents with observations and the agents decide action to take, these observations and actions happen *simultaneously.* The agents are unable to communicate, coordinate or plan for each other except for other agents appearing as obstacles. Outside of getting v_{obst} in an observation there is no enforcement of collision avoidance, instead agents motivation for avoiding one another is intrinsic to the reward function itself.

4 Results and Discussion

The aim of this paper is to assess how well policies, designed in isolation, are able to be deployed on multiple agents in the same environment. Firstly, we will see how well the single-agent policies perform in the single-agent environment. Then for each policy we test how well it performs when being deployed on a given number of agents. In Section 4.2 we will discuss how all agents having homogeneous policies leads to highly undesirable emergent behaviour, importantly in Section 4.3 we will demonstrate how we are able to overcome this emergent behaviour through noise.

4.1 Homogeneous-Policy Performance

The five single-agent policies outlined above are now tested in a single agent environment to assess performance. Each *trial* will be run 100 times (100 episodes), with an agent with a chosen policy deployed in the environment outlined in Section 2, of a 100m by 100m area made up of 56 hexes H. Each episode starts with the agent in a random location in W and then proceeding to run for 200 action-steps (which is 600 dt as each action is held for 3 dt).

For the real world problems needing to continuously surveil an area, there are a number of metrics, based on our surveillance score, which could measure performance. These can be continuous measures such as the average score, the maximum score achieved, time-to-reach a certain value or could be pass-fail such as never dropping below a certain minimum-value. The results presented here will be based on $\mathcal{V}^*(H_t)$ averaged over all runs. Traits of a *good* policy are high average values, which rise quickly and remain stable and ideally have lower variance across runs.

The single-agent deployment results, as shown in Figure 4a, show that the benchmark policy performs the best with almost no variance, with the 'peaks' of the trail policy, visible at step 56 and 112, depicting when the agent completes each lap of the trail. As expected the random policy performs the worst with the highest levels of variance. The remaining three policies (*GD*, *DDPG*, *NEAT*) all perform somewhat similarly, with *DDPG* initially performing well but later being outperformed by *GD*. All agents experience the diminishing rate of reward towards the later parts of the episode due to most of the hexes having high scores, at which point the policies simply maintain them.

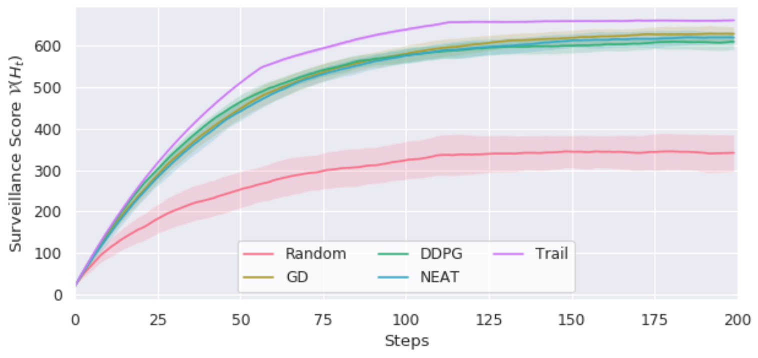
However, the results of Figure 4 and Figure 5 show that when multiple-agents are used, we quickly observe that the three policies *GD*, *DDPG* and *NEAT* all exhibit the same dramatic drop-off in performance. This is due to an effect we will discuss next in Section 4.2 which we call Homogeneous-Policy Convergence Cycle (HPCC). This effect appears to worsen with increasing numbers of agents, exhibiting this performance-drop more quickly. What is equally alarming is that this effect is so bad that for 10 agents it appears that an entirely random policy can outperform them on average.

Notably even the analytical best policy, that is The Trail Policy, also fails to easily transfer to the multi-agent scenario albeit for a different reason to the others. Here it is due to the fact the agents are placed in the world at random and are therefore not necessarily evenly spaced across the trail. This means that you could end up with agents grouped behind one another, and as discussed in Section 2.5 agents ideally want to be going to the least visited hexes next, and not to one just visited by another agent. This could be fixed in a number of ways but would require some additional coordination or planning, such as being forced to slow down to space out, or this could utilise the observations more and moving to parts of the trail where the next hex in the trail has a lower score, however these are left for future work.

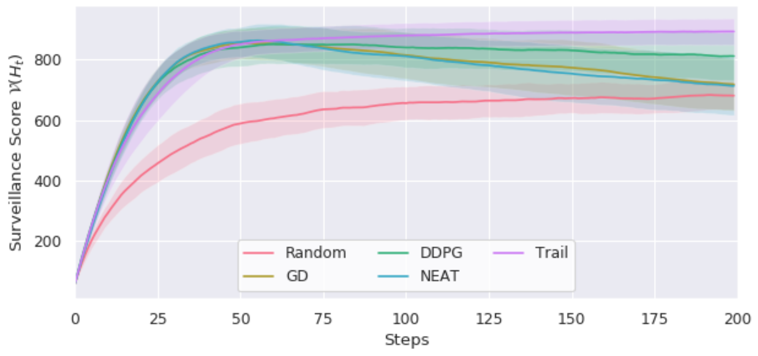
4.2 Homogeneous-Policy Convergence Cycle

Figure 4 and Figure 5 show a large reduction in performance by simply deploying a homogeneous policy to multiple-agents within the same environment. The cause of this is due to an emergent property that we call *Homogeneous-Policy Convergence Cycle* (HPCC). This property is cyclical in nature and can occur when two or more agents occupy the same hex and essentially get 'stuck' together. The process is depicted in Figure 6a and happens, at some time t, as follows

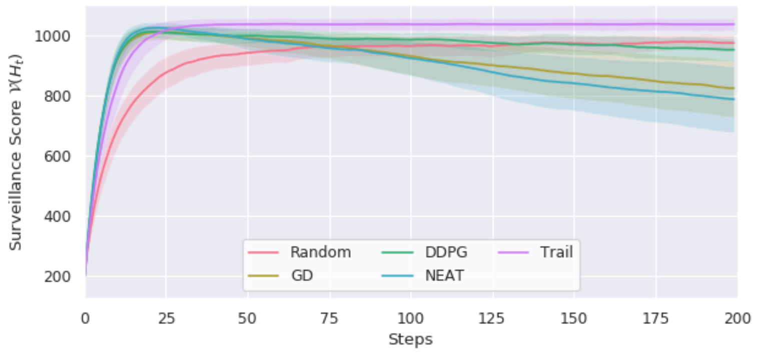
1) Agents move to the same hex h_t;
2) Agents get an identical local state observation s_t;
3) Identical, deterministic policies π, return identical action choices a_t
4) Agents in the same hex, h_t, perform identical actions, and move to the same hex, h_{t+1}, as the other agents - thus returning to step 1)



(a) 1 Agent Mean Score $\mu \pm \sigma$

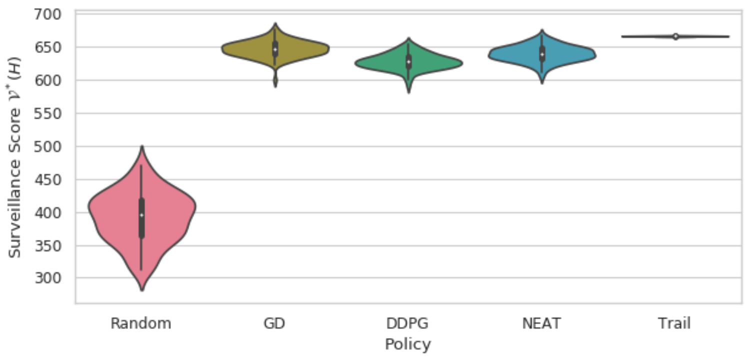


(b) 3 Agents Mean Score $\mu \pm \sigma$

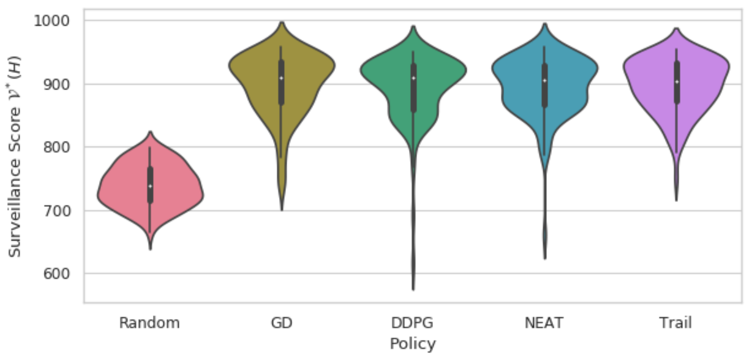


(c) 10 Agents Mean Score $\mu \pm \sigma$

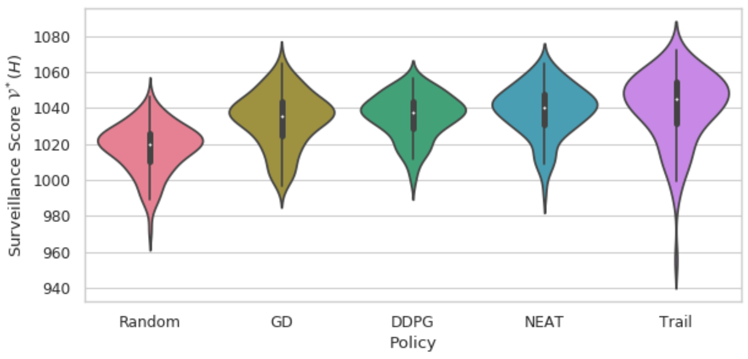
Fig. 4: Multi-Agent PSP performance over 200 action steps



(a) 1 Agent

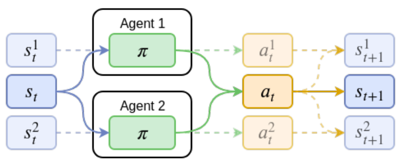


(b) 3 Agents

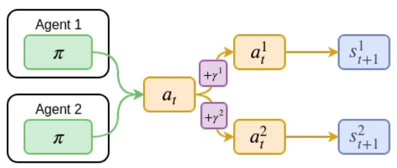


(c) 10 Agents

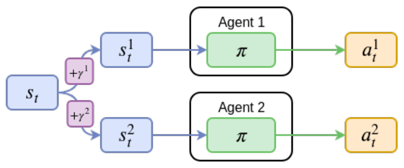
Fig. 5: $v_{\max}$ value distribution for PSP performance for differing number of Agents



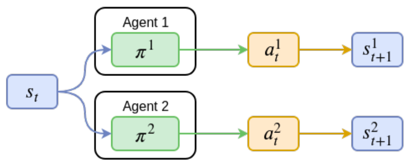
(a) HPCC outline



(b) Adding Action Noise



(c) Adding State Noise



(d) Heterogenous Policies

Fig. 6: Homogeneous Policy Convergence Cycle and potential fixes

The HPCC problem is essentially a product of homogeneity and determinism, something that appears in game-theory, the El Farol Bar problem [2] for example. The solid-line path of Figure 6a represents this convergence cycle, if we break away from any of the 4-steps above and transition onto one of the dotted paths we can break this cycle, this can be achieved by doing one of the following:

1) Co-operate to avoid moving into the same hex $\rightarrow h_t^1 \neq h_t^2$
2) Have differing states $\rightarrow s_t^1 \neq s_t^2$ and therefore $a_t^1 \neq a_t^2$
3) Have differing action choices $\rightarrow a_t^1 \neq a_t^2$ and therefore $s_{t+1}^1 \neq s_{t+1}^2$
4) Have differing policies so identical states may result in differing actions

To achieve 1) we would require the addition of coordination which is not the aim of this paper. We could also achieve 4) as we have outlined 5 different PS single-agent policies which are heterogeneous from one another. So for scenarios with 5 agents or fewer we could feasibly deploy a team of agents each with a heterogeneous-policy as depicted in Figure 6d, however for an increasing number of agents designing new, distinct, policies is more difficult and is unrealistic for large numbers of agents. Instead we focus on 2) and 3) and look at a straightforward method for overcoming the HPCC problem via the addition of noise.

4.3 Artificial Heterogeneity

We will now demonstrate two similar ways, as depicted in Figures 6b and 6c, to break HPCC through artificial-heterogeneity. By simply adding noise to either an agent's action or to an agent's state observation, we are able to essentially turn homogeneous policies into heterogeneous ones.

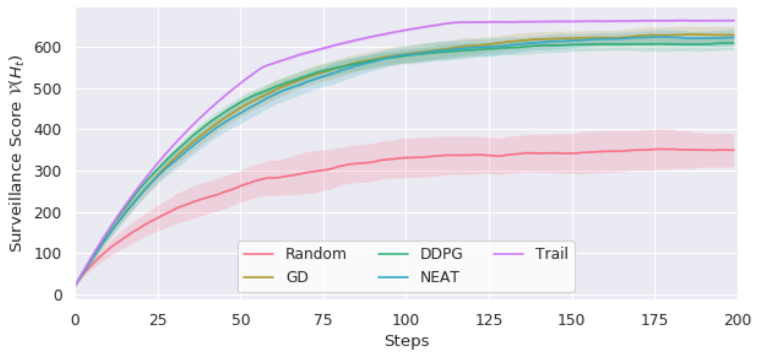
Action Noise The optimal action chosen by our policy $a_t \leftarrow \pi^*(s_t)$ will be the same for an identical state as it is deterministic. To achieve a non-deterministic action, it suffices to take the action a_t of the policy π and add a small amount of noise, γ^i, for each agent i, and for each dimension of the action space. This idea is to increase the likelihood of producing different action choices $a_t^1 \neq a_t^2$, as in Figure 6b. The noise value is drawn from a uniform distribution $\gamma \sim \mathcal{U}(-0.1, 0.1)$ for each action dimension and added to the output of the policy. Note that the addition of noise results in an action which at an individual policy level is suboptimal $a_t + \gamma \notin \pi^*(a|s_t)$.

Action-noise is sufficient in breaking HPCC, as shown in Figure 7, the previously exhibited long-term performance drop-off is entirely removed. This allows us to effectively deploy single-agent heterogeneous policies on multiple agents with only a minor adjustment.

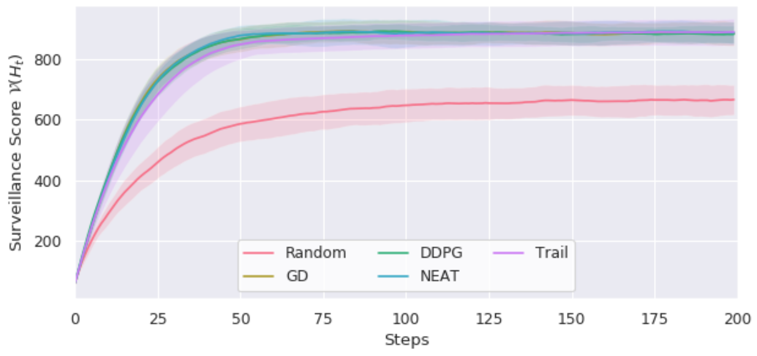
State Noise Instead of directly altering a homogeneous policy's action choice we can instead perturb our state observations, s_t, by again adding small amounts of noise. This is depicted in Figure 6c, where for each agent i, we add the noise γ^i (the same size as s) to the identical state s_t, so that $s_t^i = s_t + \gamma^i$. The aim is to increase the chances that $s_t^i \neq s_t$. Again, noise γ^i, is taken from a uniform distribution $\gamma \sim \mathcal{U}(-1.0, 1.0)$ for each agent i, with dimensions to match the state observation (this distribution is proportionally similar to the action noise distribution). The aim is that we are able to ensure that even if agents occupy the same hex, the observations they receive differ slightly and thus for a sufficient level of state noise will result in distinct action choices $a_t^1 \neq a_t^2$.

The results of Figure 8 clearly show just how effective even a small amount of noise added to either the action or state is, fixing HPCC. It appears that for higher agent counts, state noise results in higher max surveillance scores along with lower variance. As expected adding noise to the Trail policy has little effect, as it does not suffer from HPCC and it does not act on the state information.

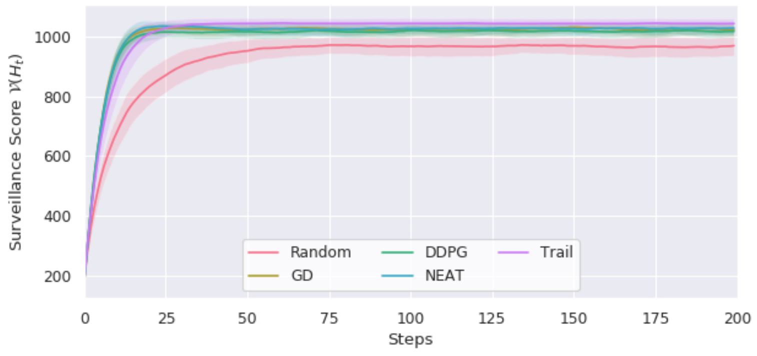
The choice of where the noise is added has some subtle differences. By adding noise to the state observation you are relying on $s + \gamma^1$ and $s + \gamma^2$ *at some point* being sufficiently different to result in the policy π outputting two different actions, and the agents then moving out of the same hex. Whereas with action noise, assuming $\gamma^1 \neq \gamma^2$, you are *forcing* $a + \gamma^1 \neq a + \gamma^2$, so agents are therefore *always* taking different actions. This is likely the cause of the difference in variance of the two approaches. Additionally, due to the discrete action selection of *GD* and *NEAT*, perturbation in the action space, as we add a continuous amount of noise, allows a little more continuity in the agents movements. With state noise only, discrete actions remain discrete, but if the same state plus noise for two agents can now produce two different actions choice, those two actions will differ by at least the discrete resolution instead of just the smaller action noise.



(a) 1 Agent Mean Score $\mu \pm \sigma$

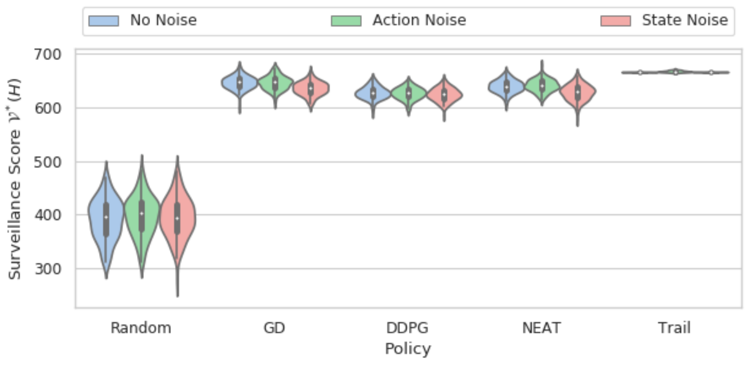


(b) 3 Agents Mean Score $\mu \pm \sigma$

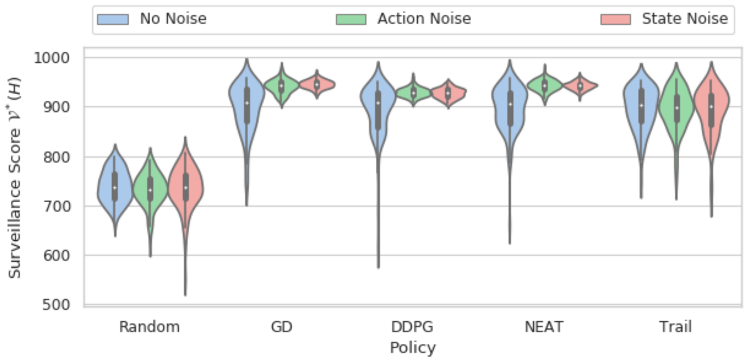


(c) 10 Agents Mean Score $\mu \pm \sigma$

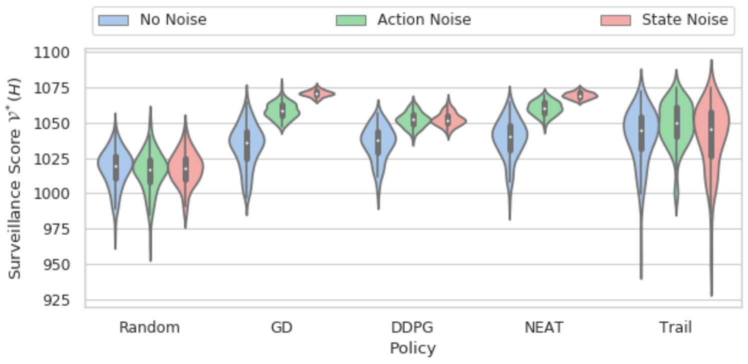
Fig. 7: Multi-Agent PSP performance over 200 action steps with action-noise



(a) 1 Agent



(b) 3 Agents



(c) 10 Agents

Fig. 8: Multi-Agent PSP $\mathcal{V}^*(H)$: Comparison of the effect of adding noise

5 Conclusion

This work was motivated by the idea of being able to design single-agent policies in isolation, and without the need for explicit cooperation or coordination still *successfully* deploy them to multi-agent scenarios. We used the multi-agent Persistent Surveillance Problem as a simple scenario to test our question. We outlined and demonstrated the results of five distinct single-agent policies designed to solve the single agent PSP: *Random*; *Gradient Descent*; *DDPG*; *NEAT*; and *Trail*. By deploying these single agent homogeneous policies to multiple agents we quickly observe a negative emergent property that we called the *Homogeneous-Policy Convergence Cycle* (HPCC). A property almost entirely the result of homogeneity and determinism. Whilst we demonstrated the existence of HPCC in MAPSP, one can imagine that this or a similar *class* of emergent properties could occur in other scenarios. Environments with similar action-state transitional properties of those depicted in Figure 6a could be subject to similar undesirable effects. Importantly however, we showed that we are able to remove this property entirely, through the simple addition of noise. By adding a small amount of noise to each agent's action choice or state observation we were able to essentially create artificial heterogeneity from entirely homogeneous policies.

This shows that some degree of noise can be a desirable property within a system. This may appear somewhat reassuring as in many real world scenarios, the introduction of state and action noise will often arise inadvertently, through imperfections in aspects such as sensing, communication or computation. However, this also hints at the potential for a whole class of emergent properties such as HPCC which may exist in many complex systems but remain unnoticed.

Many future directions are of interest, including exploring the impact of *parameter space choices*, such as different reward and score functions along with different environments. However, of greatest interest is in understanding the impact of *system level choices*, such as the inclusion of different aspects of coordination and communciation, asking not just where, but also to what degree it is necessary.

Acknowledgments

This work was funded and delivered in partnership between the Thales Group and the University of Bristol, and with the support of the UK Engineering and Physical Sciences Research Council Grant Award EP/R004757/1 entitled 'Thales-Bristol Partnership in Hybrid Autonomous Systems Engineering (T-B PHASE)'.

References

1. Albani, D., Manoni, T., Nardi, D., Trianni, V.: Dynamic UAV swarm deployment for non-uniform coverage: Robotics track. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS pp. 523–531 (2018)

2. Arthur, W.B.: Complexity and the Economy. Science **284**(5411), 107–109 (apr 1999). https://doi.org/10.1126/science.284.5411.107
3. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym pp. 1–4 (2016), http://arxiv.org/abs/1606.01540
4. Butterworth, J., Tuyls, K., Broecker, B., Paoletti, P.: Evolving coverage behaviours for MAVs using NEAT. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS **3**, 1886–1888 (2018)
5. Cabreira, T., Brisolara, L., Ferreira Jr., P.R.: Survey on Coverage Path Planning with Unmanned Aerial Vehicles, vol. 3 (2019). https://doi.org/10.3390/drones3010004
6. Hochreiter, S.: The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **06**(02), 107–116 (apr 1998). https://doi.org/10.1142/S0218488598000094
7. LaValle, S.M.: Planning Algorithms. Methods (2006). https://doi.org/10.1017/CBO9780511546877
8. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (sep 2015), http://arxiv.org/abs/1509.02971
9. Liu, Y., Nejat, G.: Robotic urban search and rescue: A survey from the control perspective. Journal of Intelligent and Robotic Systems: Theory and Applications **72**(2), 147–165 (2013). https://doi.org/10.1007/s10846-013-9822-x
10. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (feb 2015). https://doi.org/10.1038/nature14236
11. Mogili, U.R., Deepak, B.B.: Review on Application of Drone Systems in Precision Agriculture. Procedia Computer Science **133**, 502–509 (2018). https://doi.org/10.1016/j.procs.2018.07.063
12. Nigam, N.: The Multiple Unmanned Air Vehicle Persistent Surveillance Problem: A Review. Machines **2**(1), 13–72 (2014). https://doi.org/10.3390/machines2010013
13. Nigam, N., Kroo, I.: Persistent surveillance using multiple unmanned air vehicles. IEEE Aerospace Conference Proceedings pp. 1–15 (2008). https://doi.org/10.1109/AERO.2008.4526242
14. Peters, J.R., Wang, S.J., Surana, A., Bullo, F.: Asynchronous and Dynamic Coverage Control Scheme for Persistent Surveillance Missions pp. 1–12 (2016), http://arxiv.org/abs/1609.05264
15. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems, vol. 37. University of Cambridge, Department of Engineering Cambridge, England (1994)
16. Stanley, K.O., Miikkulainen, R.: Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation **10**(2), 99–127 (jun 2002). https://doi.org/10.1162/106365602320169811
17. Stone, P., Veloso, M.: Multiagent systems: a survey from a machine learning perspective. Autonomous Robots **8**(3), 345–383 (2000). https://doi.org/10.1023/A:1008942012299
18. Uhlenbeck, G.E., Ornstein, L.S.: On the Theory of the Brownian Motion. Physical Review **36**(5), 823–841 (sep 1930). https://doi.org/10.1103/PhysRev.36.823