Deep Reinforcement Learning Applications to Space Situational Awareness Scenarios

Benedict Oakes[*], **Jason F. Ralph**[†]
[*]*Centre for Doctoral Training in Distributed Algorithms*
[*†]*Department of Electrical Engineering and Electronics, University of Liverpool, UK*
Jordi Barr[‡]
[‡]*Defence Science and Technology Laboratory, UK*

ABSTRACT

In recent years, with the success of large constellation networks like Starlink and OneWeb, the number of satellites launched into low Earth orbit (LEO) has increased dramatically. With limited ground-based resources, maintaining effective space situational awareness (SSA) becomes an increasing problem - with the task of assigning sensors to observe targets being potentially combinatorially complex. In recent years, reinforcement learning (RL) has been successfully applied to a range of problems, including assignment problems such as SSA. In this paper, we present a set of RL environments for tracking satellites in different SSA scenarios. We outline methods for tracking a set of known targets, and for searching for targets in the field of regard. RL is well suited for any orbital paradigm, and in this paper we focus specifically on Earth-based sensors that can make measurements of LEO satellites. These environments are evaluated using different RL algorithms that produce sensor tasking policies, and their performance is measured.

1. INTRODUCTION

Keywords Deep Reinforcement Learning, Machine Learning for SDA Applications, Space Situational Awareness

Satellites have become an essential element of society in the 21st century, with a wide array of uses, from communication and environmental monitoring to GPS and remote sensing. As such, it is of paramount importance that the space domain is well understood, so that we can maintain a functioning system for the custody of and cataloguing of orbiting bodies. However, it is harder than ever to properly monitor the space domain due to the significant number of new satellite launches. With limited sensing availability including telescopes and radar, there is a mismatch of sensor resources to targets as the number of targets exceeds the number of sensors. This is difficult in all the orbital regimes, but we focus here on the LEO regime where targets pass quickly over an individual sensor location.

In recent years, an increasing amount of research has been conducted into the applicability of reinforcement learning to a range of domains - including problems involving resource allocation or assignment - after successes in applying RL algorithms to different games such as StarCraft and Go [1], [2]. The impressiveness of these results relate to the complex action spaces that the RL algorithms had to learn policies for. RL is apt for modelling complex, high-dimensional problems that may take classic rule-based algorithms a long time to solve. Another advantage of reinforcement learning is that once a policy has been learned, it is readily and quickly applicable to other scenarios [3], where a rule-based algorithm would be forced to recalculate the policy.

RL has been successful in a wide range of problems and there has been some research conducted into the applicability of RL to the sensor scheduling problem in SSA, but this area remains in its infancy. Others have considered formulations of sensor scheduling for SSA, with some solutions for ground-based [4, 5, 6, 7, 8, 9, 10] and space-based sensors [11]. Although RL is widely applicable to the SSA problem in many forms, there have been relatively few in-depth studies testing the suitability of RL to solving the types of assignment problems found in sensor scheduling for SSA, including its applicability to the LEO regime.

In this paper, we demonstrate the use of 2 RL environments. In both, we consider a controllable Earth based sensor, that is free to point in different directions within its field of regard (FoR). We simulate satellites in LEO orbits that pass

[*]E-mail: sgboakes@liverpool.ac.uk
[†]E-mail: jfralph@liverpool.ac.uk
[‡]E-mail: jmbarr@dstl.gov.uk

through the FoR. The sensor can make measurements of satellites if they appear in the sensor's field of view (FoV). The inferred state of each target is estimated using an unscented Kalman filter (UKF). In the first environment, the RL agent (the ground based sensor) tracks a known number of targets, and must select which targets are most appropriate to look at. In the second environment, the agent is not aware of where the targets are, and must move in the field of regard to locate potential targets. To generate sensor scheduling policies for these scenarios, we apply RL algorithms including Double-Deep Q-Network (DDQN) [12], Soft Actor-Critic (SAC) [13] and Proximal Policy Optimisation (PPO) [14] to the environments to generate policies.

To evaluate the performance of RL algorithms in our environments, we compare the policies produced to a greedy policy, where at each time-step of the scenario, each possible action is considered and the best myopic (one-step ahead) action is chosen. We also inspect the results from the UKF and observe the change in target uncertainty throughout the observation period.

This paper demonstrates the use of RL in the SSA domain. With some time-constrained offline learning, the agents learn policies that are capable of either choosing targets or locating them. These policies are general and work well on targets with different orbits, with the time to deploy a learned policy being very short. In future work, RL can be adapted to consider more advanced scenarios with multiple sensors, or where one is interested in unexpected target manoeuvres or collision events.

2. REINFORCEMENT LEARNING

In reinforcement learning, an intelligent agent makes decisions to maximise a cumulative reward. The reward the agent receives is impacted by the decisions it makes, and the agent learns which decisions are good or bad via an observation of the environment. RL environments can take the form of a partially-observed Markov decision process (POMDP) [15]. A Markov decision process is a control process in which a set of states S, can be reached by taking actions A, with a probability of taking actions P, and returning a reward R. This forms a 4-tuple (S,A,P,R). In RL applications, the full state of the MDP is not always known and the agent is only aware of certain influences of its decision making; hence the MDP is partially observed [16].

In the case of SSA, we model the reinforcement learning *environment* where the agent making the decisions represents the controllable telescope or radar making angle and range measurements of satellites that are orbiting the Earth. At each step of the environment, we can control the telescope to point in certain directions and it is rewarded by how well the observation made improves the understanding of the satellites by reducing position and velocity uncertainty, for example.

2.1 Algorithms

We employ a range of algorithms for training agents in our environments.

There are several different RL algorithms that are able to learn from a wide range of inputs. Currently well used single agent algorithms include DDQN, SAC, and PPO [17].

DDQN is based on Q-learning. Q-learning is a simple value iteration update on a Markov decision process. Q-values, or quality-values, refer to the expected reward gained by taking a certain action in a given state. Q-learning attempts to find Q-values for a range of states and actions, and to then exploit the Q-values by selecting the action that returns the highest reward at any state in a greedy policy. Q-learning is different from a Q-value iteration algorithm, as the transition probabilities and rewards are initially unknown [18].

Q-learning is defined by:

$$Q^{new}(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \cdot (r_t + \gamma \cdot \max Q(s_{t+1},a) - Q(s_t,a_t)) \tag{1}$$

where Q is the expected reward and is a function of action a and state s at time t. α is the learning rate, and $0 < \gamma < 1$ is the discount factor. α is a tuning parameter that determines how quickly the algorithm learns new information. γ is a parameter required for convergence of the algorithm, and determines how much weight is given to information in the future. If γ is close to 1, the future is valued almost as much as the present. If γ is close to 0, the immediate information is much more highly valued [19].

In Actor-Critic methods, such as SAC, there are 2 networks that are trained simultaneously: the actor and critic networks [13]. The actor refers to the policy, that asks the estimated value function, or critic, about the next possible

state values, which the critic improves during learning. SAC employs the use of entropy regularisation. In RL, entropy is the amount of information contained in a stochastic variable.

If we have a random variable $X = \{x_0, x_1, \ldots, x_n\}$ with probability distributions $p(x)$, then the entropy H of X is given by

$$H(X) = \mathbb{E}[I(X)] = -\sum_{x \in \mathbb{X}} p(x) \log p(x) \tag{2}$$

To find optimal policies, there must be some stochasticity to visit enough states and actions in the environment. Entropy can help determine how well distributed a policy is.

If we want to ensure that we have a robust policy, we can force the RL agent to visit every state by using a maximum entropy objective function. This way, instead of learning a policy to maximise the reward, we can learn a policy that handles a range of different behaviours in the environment.

The maximum entropy objective function is given by

$$\pi^* = \arg\max_{\pi} \sum_{s \in S, a \in A} \mathbb{E}r + \alpha H(\pi(a|s)) \tag{3}$$

where not only are we maximising the expected reward r, but we also promote increasing entropy by exploring the state and action spaces more fully. By incentivising exploration, this method can also train faster than other policies.

In proximal policy optimisation, multiple agents learn simultaneously and each can send and receive weights from a master network. We make use of the PPO-clip algorithm; when we make large policy updates, the objective function is clipped to prevent the policy changing too much between updates. This increases training stability [14].

In this paper, we make use of the DQN, SAC, and PPO algorithms on our simulated environments to generate separate policies. We compare the effectiveness of these policies on each environment.

3. ENVIRONMENTS

Our environments are all centred around controlling a ground-based sensor. In each environment, the RL agent controls the pointing of the sensor.

3.1 Sensor

The ground based sensor provides measurements in the form $y = [\theta, \phi, r]$. This models either a radar that can detect azimuth, elevation, and range, or an optical telescope that measures azimuth and elevation, supplemented with a laser-ranging system, for example. The sensor's dwell centre, or boresight, is moved around its field of regard by the RL agent. The sensor is restricted from looking too close to the horizon. The sensor has a square field of view around the boresight, and any target that appears within the FoV is detected.

Fig. 1 shows the sensor's field of regard with targets passing overhead. Each coloured line is a satellite passing overhead, and the sensor FoV is shown with a black box. The thick black line specifies a minimum elevation horizon that the sensor can not point below. As we are considering LEO satellites, the paths move quickly over the sensor position, so there is only a limited amount of time for the sensor to be able to detect targets before they disappear.

3.2 Satellite Data, Propagation, and Filtering

In each environment, we make use of real satellite data in the form of Two-Line Element (TLE) sets [20]. The TLE data format describes the orbit of a satellite at a particular point - or epoch - in time. The TLE format is specifically designed to work with the simplified general perturbations (SGP4) propagator [21], which includes secondary propagation effects like atmospheric drag and solar radiation pressure.

In our scenario, we consider only the TLEs of LEO satellites with an altitude of less than 2000km. To select appropriate TLEs for a scenario, we select a sensor location, and then calculate what the measurements for each TLE would be from that location at a given time. We keep all satellites that would be visible from the sensor location for at least one time step at that time.

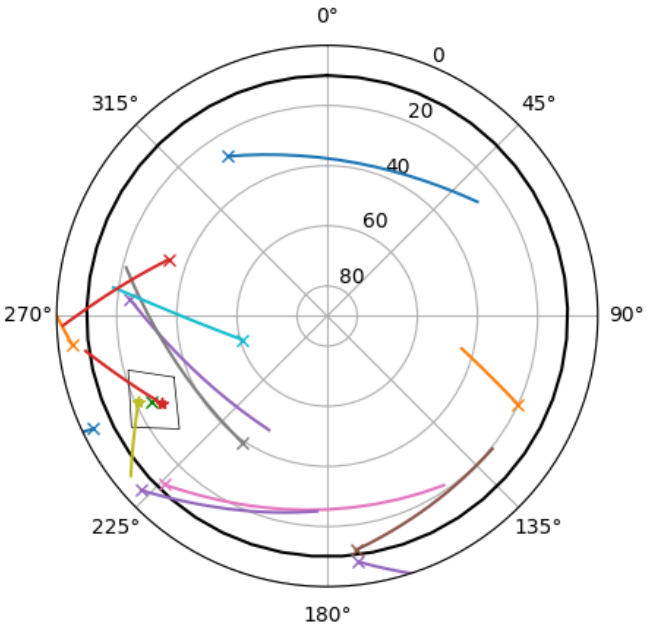


Fig. 1: Satellite paths crossing an Earth based sensor's field of regard. The middle of the graph is the sensor's zenith, and the horizon is the edge. The sensor field of view is shown by the box, with a minimum elevation indicated with a dark line.

We use the TLEs and SGP4 propagator in conjunction with an unscented Kalman filter (UKF) to help handle the non-linear dynamics of orbiting bodies. We make use of the Stone Soup framework [22], a Python package for state estimation and target tracking. We store the state vector of the target in the filter as a Cartesian state in the geocentric Cartesian reference system (GCRS) frame.

3.3 RL Environment

In each environment, we specify a number of parameters that determine key features.

Parameter	Value	Description
N_{S_T}	5, 10, 20, 40	number of satellites in simulation
L_{sim}	30	number of discrete time steps
L_{step}	60	length in seconds between each time step
sensor	-	-
position	$28°, -16°$, 2400m	sensor location in latitude, longitude, altitude
field of view	$15°$	length of side of FoV
horizon	$10°$	minimum elevation
slew rate	$2°$	sensor slew rate
start time	date	start time of simulation in UTC
TLE	tle	which TLEs are input

Table 1: Environment parameters

We run the above setup on the different algorithms discussed in Section 2, and we vary the number of satellites to see the effect on the performance of the agents. In the environment, we must specify an action space and an observation

space that the RL agents use to train their policies. At a high level, there are 2 environments that we consider. Below, we define the environments.

3.3.1 Environment A

Environment A is a discrete action space environment, where the action space is set to the catalogue of available satellites.

Action Space: In environment A, the action space is determined by the number of satellites that are simulated. At each environment time-step, the agent chooses which satellite to point the sensor at.

Observation Space: The observation space is the $\mathrm{Tr}(p)$ of each satellite where p is the covariance matrix, with a Boolean V for whether or not it is visible at the current time-step.

$$O = [\underset{x\in X}{p(x)}, \underset{x\in X}{V_x}] \tag{4}$$

Reward Function: The reward function at each time-step is defined as the difference between the prior and posterior uncertainties:

$$R = \Delta P = P_{k-1} - P_k = \sum_{x\in\mathbb{X}} \mathrm{Tr}(p(x)_{k-1}) - \sum_{x\in\mathbb{X}} \mathrm{Tr}(p(x)_k) \tag{5}$$

Here, the agent receives a higher reward for minimising the overall uncertainty of all the targets. We subtract the posterior uncertainty from the prior uncertainty because if the total uncertainty is lower at the current time-step than at the previous time-step, we want to have a positive reward. This should encourage the agent to make measurements of particularly uncertain targets.

3.3.2 Environment B

Environment B is a continuous action space environment, where the action space is set to changing the sensor dwell centre in the field of regard.

Action Space: In environment B, the action space is set as a movement of the sensor position in elevation and bearing. At each time-step, the agent determines the amount by which to change the sensor's azimuth and bearing.

Observation Space: Observation is a list containing the current sensor position, followed by the contributions of each element in the reward function. s_θ and s_ϕ are the sensor's azimuth and elevation boresights respectively, N_S, is the number of detected targets at each step, $N_{S,same}$ is the number of targets detected that were also detected in the previous time step, $N_{S,new}$ is the number of targets detected at the current time step that weren't detected at the previous time step, and N_T is the total number of satellites detected so far in the episode.

$$O = [s_\theta, s_\phi, N_S, N_{S,same}, N_{S,new}, N_T] \tag{6}$$

Reward Function: The reward function at each time-step is defined as the number of targets detected, with a punishment for observing previously detected targets, and a reward for observing new targets:

$$R = S_k + S_k \veebar S_{k-1} - S_k \wedge S_{k-1} + S_T \tag{7}$$

Here, we get a reward for detecting more targets S_k, a bonus reward for seeing new targets $S_k \veebar S_{k-1}$, a punishment for seeing the same targets $S_k \wedge S_{k-1}$, and an additional bonus for the total number of targets seen so far in the episode S_T. As the observation space also contains the sensor azimuth and elevation, the agent should learn that staying at similar azimuths/elevations as previous time-steps is associated with seeing the same targets, which will reduce the reward.

4. RESULTS

To train the algorithms, we use TensorFlow Agents (TFA) [23]. We use the TFA framework to build the custom environments and train them with the included agents. The training was completed on Science and Technology Facilities Council (STFC) Hartree Centre's Scafell Pike high-performance computer (HPC) [24]. We make use of the HPC due to long training times and the need for parallel computation.

For training the agents, we train with the hyper-parameters shown in Table 2 (where α is the learning rate and γ is the discount factor). We run separate simulations for a different number of satellites that cross the field of regard to see how the RL agent handles more targets. While we set a maximum limit on the number of iterations, we also stop training after 16 hours to ensure that training times are not unreasonable.

To compare the results with a myopic policy, we wrap each RL agent with a greedy policy, that maximises the rewards returned at each time-step, instead of the cumulative reward. This allows us to determine if the reward gained from a non-myopic - or long-term - policy is better than a greedy policy, and thus whether RL is well suited to this problem.

Parameter	DDQN	SAC	PPO
Number of Iterations	2×10^5	1×10^6	5×10^5
α[1]	1×10^{-3}	1×10^{-3}	1×10^{-3}
γ	1.0	0.99	0.99
Neural Network Layers[2]	100, 50	256, 256	200, 100
Optimiser		Adam	
Loss Function	Element-wise Squared	Element-wise Squared	Entropy Regularisation

[1] Includes critic and actor learning rates for SAC
[2] Number of nodes in fully connected layers

Table 2: Training Hyper-parameters

4.1 Environment A

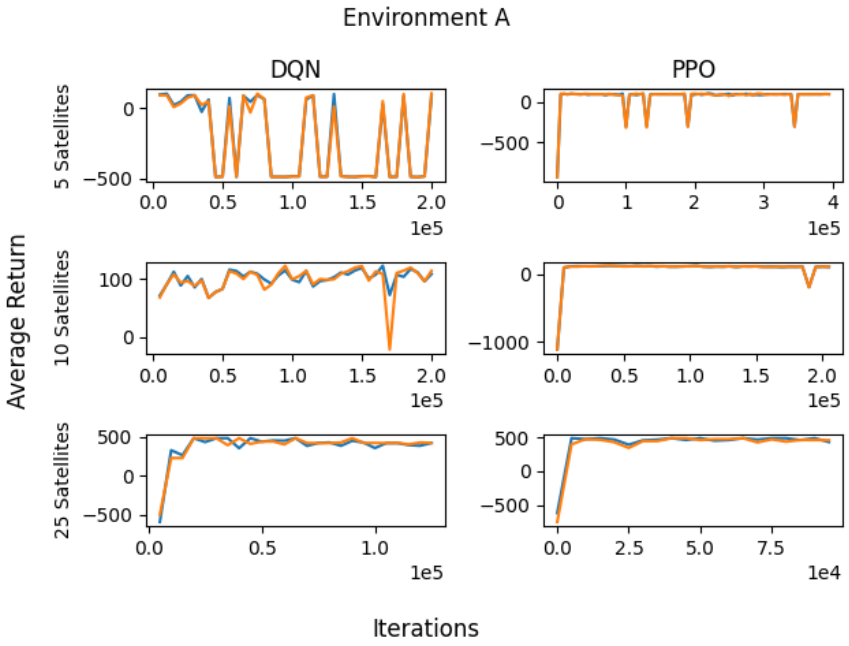


Fig. 2: Average reward from 10 episodes returned by Environment A using DDQN and PPO. The blue line shows the non-myopic policy, and the orange shows the greedy policy.

In Fig. 2, we see the reward gained during training for environment A. As the RL agent trains, we see that the agents mostly trend towards an optimum policy, where the average returned reward during an episode approaches a maximum value. The DDQN has a much less stable learning curve than is seen from PPO, but both attain an optimal policy. The PPO algorithm reaches this optimum quite quickly. There is not much divergence between the non-myopic and greedy policy's rewards, which means there may only be limited benefit of using non-myopic policies.

In Figs. 3 and 4, we use the policy with the UKF to see how the DDQN and PPO agents reduces uncertainty in environment A. We compare this with the greedy policies in Figs. 3b, 3d, 3f, 4b, 4d, and 4f. We see that the policy performs slightly better than the greedy policy. For some targets in the scenario, the UKF does a very poor job of filtering, and if no measurements are made, some target's uncertainty get too high and we consider the target to be lost. However, for nearly all targets in the 5 and 10 satellite simulation, and over half of the targets in the 25 satellite simulation, we see that uncertainty is kept low and we maintain a good catalogue of the targets. While both the DDQN and PPO agents end up with a low final uncertainty, we see that the PPO agent is better at maintaining lower uncertainty at the start of the episode, while the DDQN agent allows the uncertainty to balloon before being brought under control.

To summarise the results, we see the final total uncertainties in Table 3 for different numbers of satellites and different policies. The policies were ran on 10 seeded episodes and an average final total uncertainty was found. For the simulations with 10 and 25 satellites, the lowest uncertainty comes from the DDQN and PPO policies respectively, whereas in the 5 satellites simulation, the PPO greedy policy is the best. This follows in that for more complicated scenarios, the RL policies will have more ample opportunity to demonstrate non-myopic behaviour.

N_S	DDQN	DDQN Greedy	PPO	PPO Greedy
5	3.850×10^4	5.235×10^4	2.412×10^4	$\mathbf{2.275 \times 10^4}$
10	$\mathbf{9.312 \times 10^6}$	1.032×10^7	1.834×10^7	2.620×10^7
25	2.893×10^8	6.552×10^8	$\mathbf{2.478 \times 10^8}$	4.348×10^8

Table 3: Final total trace uncertainties for Environment B with different policies and number of satellites on a seeded environment

4.2 Environment B

In environment B, we can see that the RL agent trains in Fig. 5. As with environment A, we see that the policies mostly tend to an optimum policy. We see that the PPO policy in the 5 and 10 satellite simulations initially achieves a higher reward than is obtained at the end, but this early gain is likely attributed to an over-fitted scenario. Again, there is only a very small difference between the non-myopic and greedy policies.

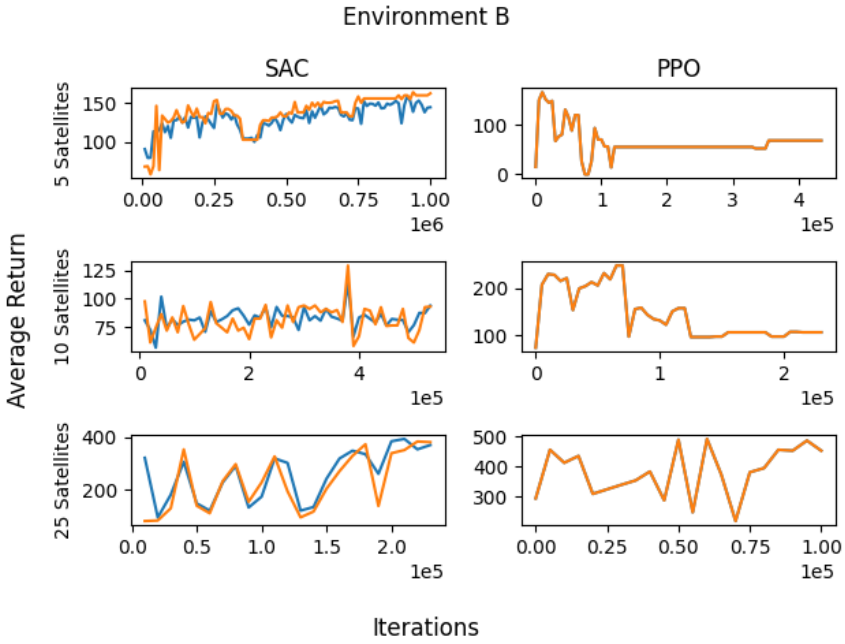


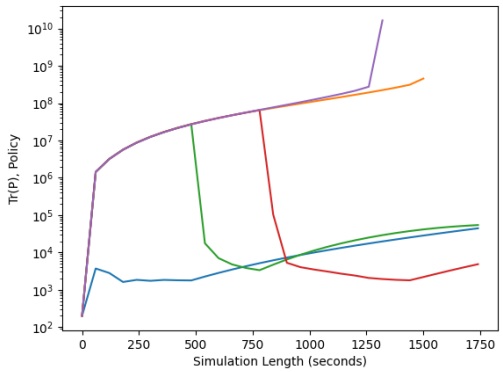
Fig. 5: Average reward from 10 episodes returned by Environment B using SAC and PPO. The blue line shows the non-myopic policy, and the orange shows the greedy policy.

In Figs. 6 and 7, we can again see the uncertainty decreases. In this environment, we see the covariances increase after the object has been detected. This is not indicative of poor performance however, as the sensor should be moving away from already detected targets in an attempt to see new ones.

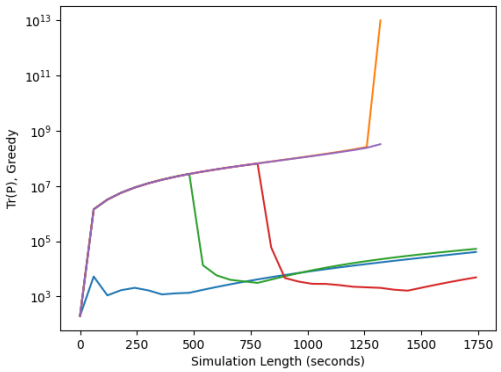
To summarise the results, we see the final total uncertainties in Table 4 for different numbers of satellites and different policies. The policies were ran on 10 seeded episodes, and in each episode, the total trace uncertainty at the end of the episode for all satellites was summed. These total trace uncertainties were then averaged over the 10 runs. We see that in all environments, the greedy policies returned the lowest covariance. However, in environment B we are focused not on reducing uncertainty, but discovering targets in the field of regard. We also investigated the total number of satellites seen at the end of an episode using the policy. We see that generally, the greedy policies were just as capable at finding targets as the non-myopic policies. We see no real correlation for the final uncertainty between myopic and non-myopic policies, but both were capable of making enough observations to reduce uncertainty. Generally, the PPO algorithm was much better at discovering targets than the SAC algorithm.

N_S	SAC		SAC Greedy		PPO		PPO Greedy	
	P	N_T	P	N_T	P	N_T	P	N_T
5	3.864×10^6	**5**	4.236×10^6	**5**	2.272×10^6	**5**	$\mathbf{2.062 \times 10^6}$	**5**
10	2.790×10^7	7	$\mathbf{5.488 \times 10^6}$	5	2.140×10^8	**10**	4.272×10^8	**10**
25	1.061×10^9	17	$\mathbf{4.947 \times 10^8}$	16	1.022×10^9	**21**	3.176×10^9	**21**

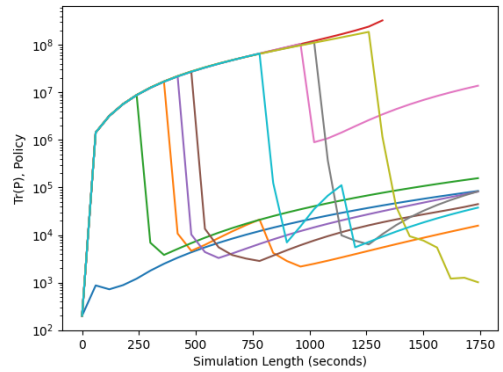
Table 4: Final total trace uncertainties P for Environment B with different policies and number of satellites on a seeded environment, with the total number of satellites observed N_T.



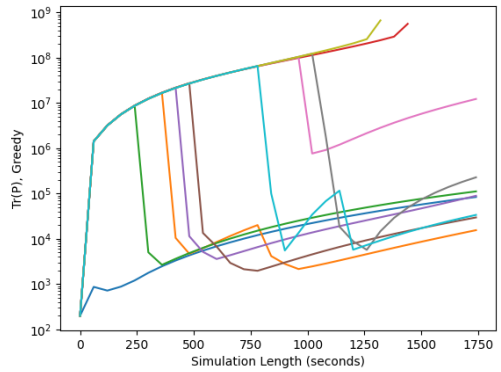
(a) DDQN policy, 5 satellites



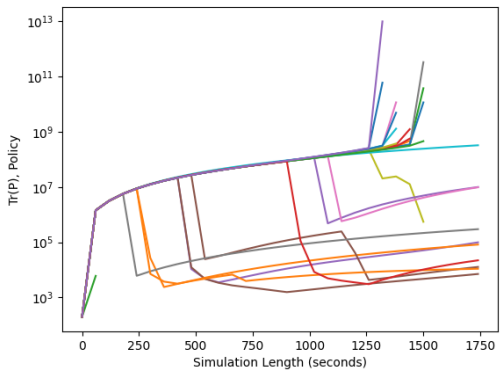
(b) DDQN greedy policy, 5 satellites



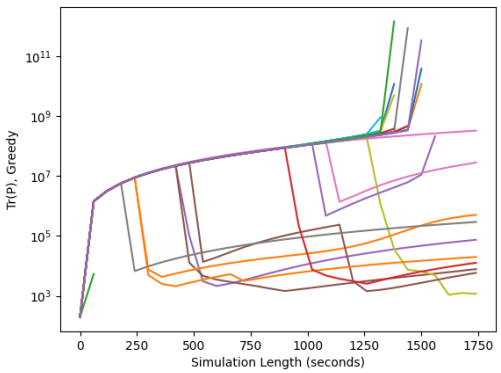
(c) DDQN policy, 10 satellites



(d) DDQN greedy policy, 10 satellites

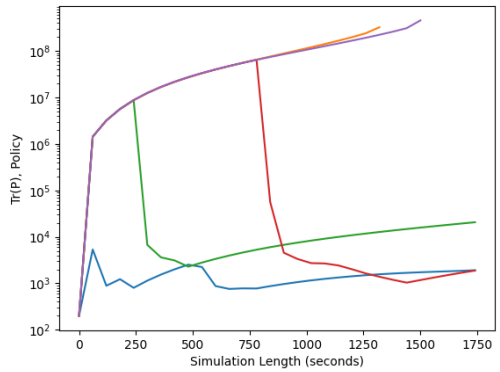


(e) DDQN policy, 25 satellites

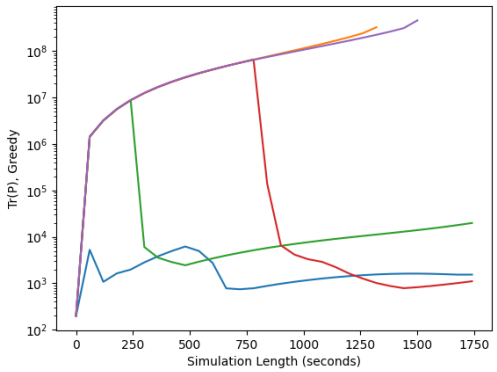


(f) DDQN greedy policy, 25 satellites

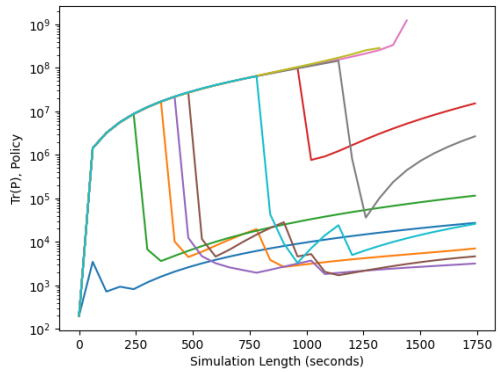
Fig. 3: Trace satellite covariances for environment A using the DDQN policies. The trace of each satellite's uncertainty is plotted during an episode using the policy. Each line represents 1 satellite.



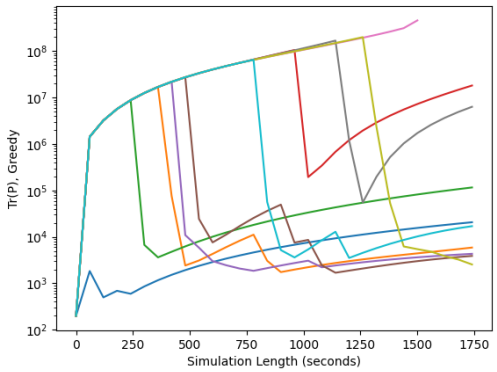
(a) PPO policy, 5 satellites



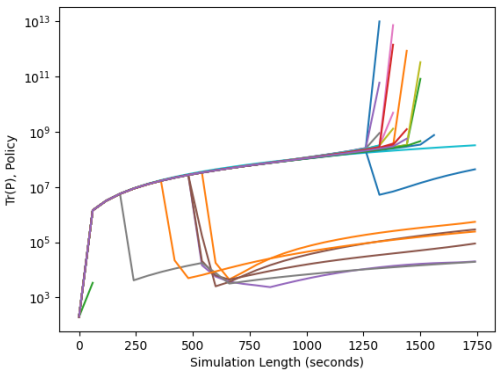
(b) PPO greedy policy, 5 satellites



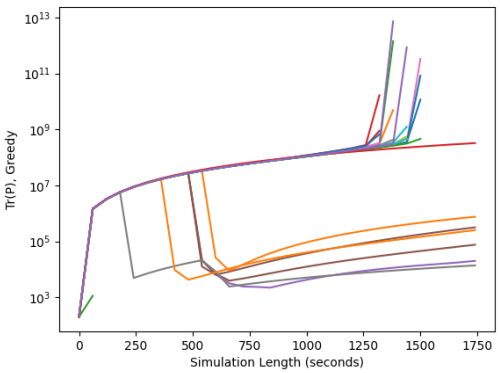
(c) PPO policy, 10 satellites



(d) PPO greedy policy, 10 satellites

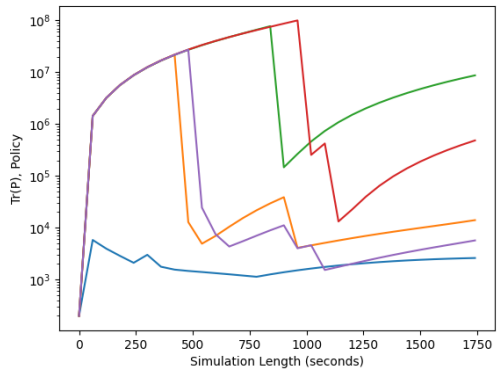


(e) PPO policy, 25 satellites

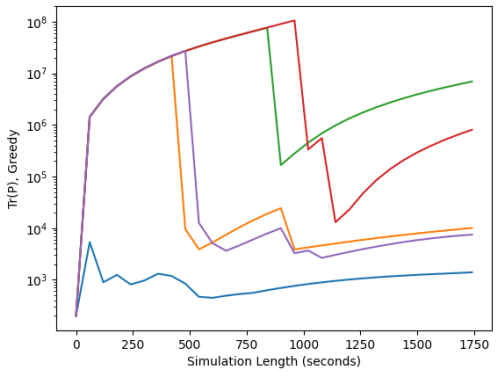


(f) PPO greedy policy, 25 satellites

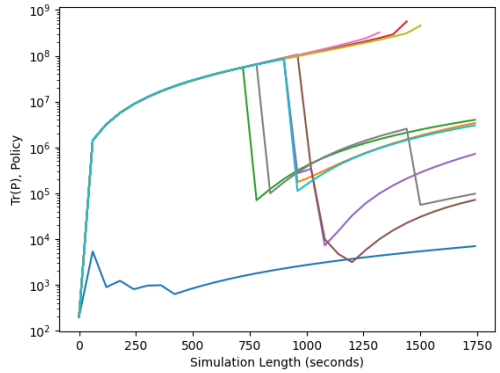
Fig. 4: Trace satellite covariances for environment A using the PPO policies. The trace of each satellite's uncertainty is plotted during an episode using the policy. Each line represents 1 satellite.



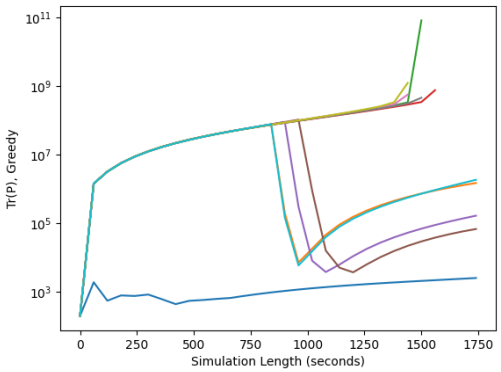
(a) SAC policy, 5 satellites



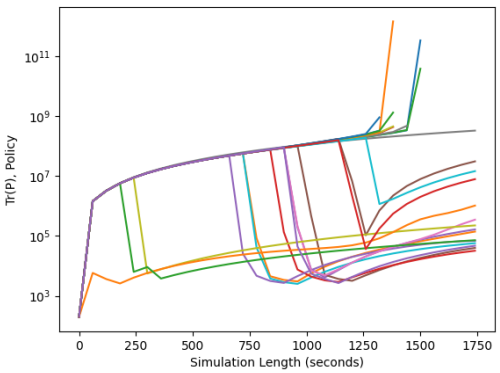
(b) SAC greedy policy, 5 satellites



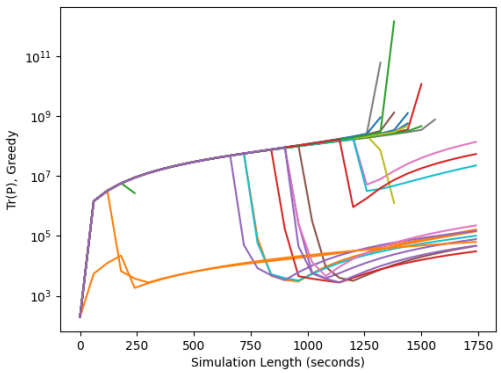
(c) SAC policy, 10 satellites



(d) SAC greedy policy, 10 satellites

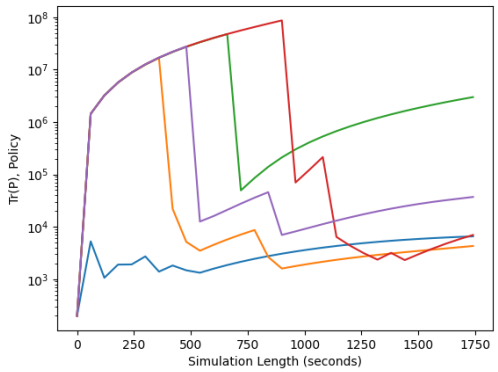


(e) SAC policy, 25 satellites

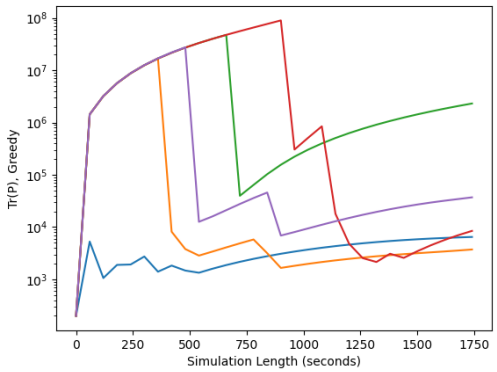


(f) SAC greedy policy, 25 satellites

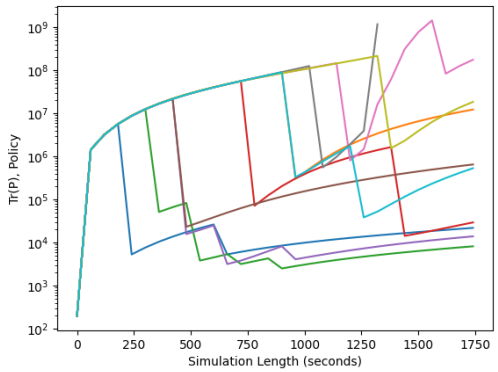
Fig. 6: Trace satellite covariances for environment B using the SAC policies. The trace of each satellite's uncertainty is plotted during an episode using the policy. Each line represents 1 satellite.



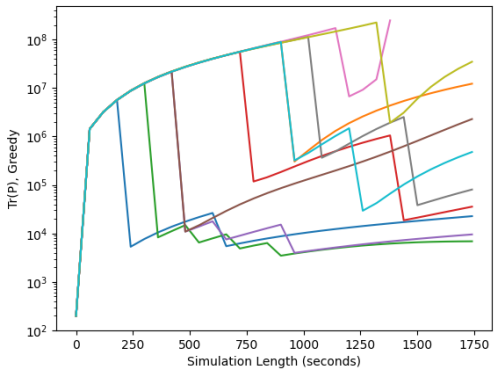
(a) PPO policy, 5 satellites



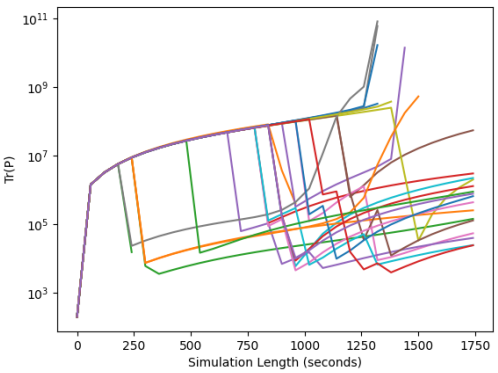
(b) PPO greedy policy, 5 satellites



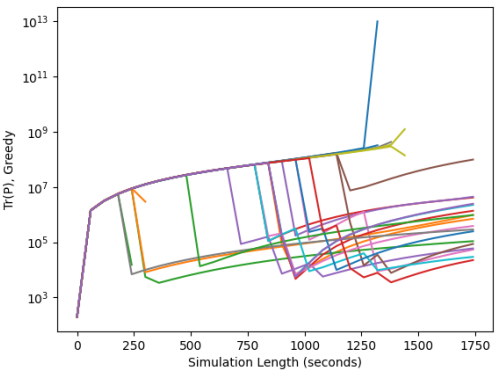
(c) PPO policy, 10 satellites



(d) PPO greedy policy, 10 satellites



(e) PPO policy, 25 satellites



(f) PPO greedy policy, 25 satellites

Fig. 7: Trace satellite covariances for environment B using the PPO policies. The trace of each satellite's uncertainty is plotted during an episode using the policy. Each line represents 1 satellite.

5. CONCLUSIONS

In this paper, we presented the use of reinforcement learning to task ground-based sensors for tracking satellites in low-Earth orbits. We used real satellite data in the form of two-line element sets with a simulated controllable sensor, to generate measurements that were used with an unscented Kalman filter to maintain an estimate of the multi-target state which is used to determine a sensor assignment policy. We present two reinforcement learning environments, one where the goal is to reduce total uncertainty of satellites in a catalogue, and one where the goal is to discover targets in the field of regard. These scenarios were trained using different reinforcement learning algorithms, and the results were compared with a greedy policy. In the first environment, we saw that uncertainty was reduced for targets that were tracked, but not all targets were well localised. In the second environment, nearly all of the targets were detected using the reinforcement learning algorithm. In general, there is no significant difference between the non-myopic and greedy policies in terms of uncertainty reduction in the first environment, or number of targets detected in the field of regard in the second environment. Further testing is required to investigate if RL is capable of generating better long-term sensor scheduling policies.

In future work, we aim to consider the effect of applying multi-agent reinforcement learning (MARL) to model multiple sensors in space situational awareness scenarios for satellites in LEO. It is likely that for the LEO regime, MARL will be particularly useful given the quick crossing times of satellites over sensors; if one sensor is incapable of seeing a target due to it's quick pass, another sensor may be able to detect it. We also would like to consider in detail the effect of RL on non-myopic decision making to improve tracks from collisions or manoeuvring targets.

REFERENCES

[1] Oriol Vinyals et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[2] David Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

[4] Daniel Jang, Peng Mun Siew, David Gondelach, and Richard Linares. Space situational awareness tasking for narrow field of view sensors: A deep reinforcement learning approach. *71st International Astronautical Congress. International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law*, 2020.

[5] Peng Mun Siew, Daniel Jang, Thomas G Roberts, Richard Linares, and Justin Fletcher. Cislunar space situational awareness sensor tasking using deep reinforcement learning agents. *23rd Advanced Maui Optical and Space Surveillance Technologies*, 2022.

[6] Peng Mun Siew, Tory Smith, Ravi Ponmalai, and Richard Linares. Scalable multi-agent sensor tasking using deep reinforcement learning. *24th Advanced Maui Optical and Space Surveillance Technologies*, 2023.

[7] R. Linares and R. Furfaro. Dynamic sensor tasking for space situational awareness via reinforcement learning. In *Advanced Maui Optical and Space Surveillance Technologies Conference*, page 36, September 2016.

[8] R. Linares and R. Furfaro. An Autonomous Sensor Tasking Approach for Large Scale Space Object Cataloging. *Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, page 55, January 2017.

[9] Ashton Harvey, Kathryn Laskey, and Kou-Chu Chang. Machine learning applications for sensor tasking with non-linear filtering. *Sensors*, 2021.

[10] Ashton E. Harvey and Kathryn B. Laskey. Online learning techniques for space situational awareness (poster). *2019 22th International Conference on Information Fusion (FUSION)*, pages 1–7, July 2019.

[11] Peng Mun Siew, Daniel Jang, Thomas G. Roberts, and Richard Linares. Space-based sensor tasking using deep reinforcement learning. *The Journal of the Astronautical Sciences*, 69(6):1855–1892, 2022.

[12] Arthur Guez Hado van Hasselt. Deep reinforcement learning with double q-learning.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms.

[15] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforce-

ment learning for POMDPs. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2117–2126. PMLR, 10–15 Jul 2018.

[16] Kevin P Murphy. A survey of pomdp solution techniques. *environment*, 2(10), 2000.

[17] Fadi AlMahamid and Katarina Grolinger. Reinforcement learning algorithms: An overview and classification. In *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–7, 2021.

[18] A Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2019. ISBN: 9781492032649.

[19] G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 1994.

[20] David Vallado and Paul Cefola. Two-line element sets - practice and use. *Proceedings of the International Astronautical Congress, IAC*, 7:5812–5825, 01 2012.

[21] David Vallado and Paul Crawford. Sgp4 orbit determination. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, page 6770, 2008.

[22] Steven Hiscocks, Jordi Barr, Nicola Perree, James Wright, Henry Pritchett, Oliver Rosoman, Michael Harris, Roisín Gorman, Sam Pike, Peter Carniglia, Lyudmil Vladimirov, and Benedict Oakes. Stone soup: No longer just an appetiser. In *2023 26th International Conference on Information Fusion (FUSION)*, pages 1–8, 2023.

[23] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. `https://github.com/tensorflow/agents`, 2018. [Online; accessed 25-June-2019].

[24] STFC Hartree. High performance computing - hartree centre. `https://www.hartree.stfc.ac.uk/technologies/high-performance-computing/`.