Final Project Report: A Comparative Analysis of Reinforcement Learning Methods for Stratospheric Balloon Station Keeping

Sean P. Corum* [1] Jared A. Watrous* [2]

Abstract

Station seeking by stratospheric balloons is an important problem amenable to reinforcement learning. In this Final Project Report, we define station keeping as a reinforcement learning problem and report on seven agents (three control agents implemented by others and four reinforcement agents trained by us). We evaluated the agents within the Balloon Learning Environment and found relatively good performance by two positive controls (StationSeeker and "Perciatelli"), intermediate performance by two of our agents (PPO and DQN), and poor performance by our other two agents (discrete SAC and QR-DQN) as well as the random walk negative control. Future work includes (1) determining the problems with the discrete SAC and QR-DQN agents and engineering them to learn, (2) systematically tuning hyperparameters for better agent performance overall, and (3) determining the reason for differences observed between our results and those reported in the literature.

1. Introduction

Lighter-than-air stratospheric balloons are used in scientific research, in the military arena, and in civil applications such as predicting the paths of forest fires and establishing communications networks after disasters (Osprey, 2020). To operate effectively, these balloons need to steer to a location and then maintain position. This is called *station keeping*. However, stratospheric balloons have no powered locomotion, so the means of steering is to collect or dispense with ballast (e.g., air) in a fixed volume chamber, causing a change in elevation to one with favorable winds that then guide the balloon towards its station. This steering needs to take into account predicted winds as a function of elevation, air temperature and pressure, internal pressure, time of day, and power reserves. Traditionally, this has been done with a combination of heuristic-based computer programs and ground-based human pilots. However, in recent years reinforcement learning (RL) has been used to improve station keeping as compared to traditional methods (Bellemare et al., 2020; Yang et al., 2020; Gannetti et al., 2023; Jeger et al., 2023; Saunders et al., 2023; 2024). In this study, we aimed to the compare the published RL methods used for station keeping within a consistent simulated environment in order to see which one might be best for real-world implementation.

*Equal contribution [1]Aerostar International LLC, Sioux Falls, SD, USA [2]Computer Science Department, Stanford University, Palo Alto, California, USA. Correspondence to: Sean Corum <spcorum@stanford.edu>, Jared Watrous <jwatrous2002@stanford.edu>.

2. Related Work

Surveying the literature on the RL methods used for station keeping, slightly older papers (published 1-4 years ago) used variants of the Deep Q-Network (DQN) algorithm while slightly newer papers (published 0-2 years ago) used the Soft Actor Critic (SAC) algorithm. Specifically, Bellemare et al. used Quantile Regression DQN (QR-DQN) (Bellemare et al., 2020), Xu et al. used dueling double DQN (DD-DQN) (Xu et al., 2022), and Gannetti et al. used vanilla DQN (Gannetti et al., 2023); Jeger et al. and Saunders et al. used the Soft Actor Critic (SAC) algorithm (Jeger et al., 2023; Saunders et al., 2023; 2024). Curiously, we found no papers using Proximal Policy Optimization (PPO) (Schulman et al., 2017) or policy gradient algorithms for the station keeping problem.

3. Our Approach

3.1. Reinforcement Learning Problem Definition

For our purposes, we define the stratospheric balloon station keeping RL problem as a finite-horizon, discounted MDP, $< \mathcal{S}, \mathcal{A}, R, T, \gamma >$. Following Bellemare et al. (Bellemare et al., 2020), we define the state space $\mathcal{S}$ as consisting of a mix of 373 discrete and continuous variables related to the internal state of the balloon (altitude, power reserves, various pressures, last action, etc.) along with a cellularized column of wind attributes (magnitude, relative bearing, and uncertainty). For specific details see Bellemare et. al. We define the *discrete* action space to consist of three actions: ascend, descend, and stay level. The distance based, power penalized reward reward function is

$$r(s, a) = r(\Delta, \omega) = f_\omega r_{\text{dist}}(\Delta), \tag{1}$$

where

$$r(\Delta) = \begin{cases} 1.0, & \text{if } \Delta < \rho \\ c_{\text{cliff}} 2^{-(\Delta-\rho)/\tau} & \text{otherwise} \end{cases} \tag{2}$$

and

$$f_\omega = \begin{cases} 0.95 - 0.3\omega, & \text{if } \omega > 0 \\ 1.0, & \text{otherwise} \end{cases}. \tag{3}$$

Here, ρ is the distance in km from a vertical line extending from the surface of the Earth defining the center of a station and $\omega \in [0, 1]$ is a normalized measure of power consumption. Continuing to follow Bellemare et al. (Bellemare et al., 2020), we use $c_{\text{cliff}} = 0.4$ and $\tau = 100$ km as those authors found through experimentation that those values give the best performance for station keeping with a 50 km radius. We do not define the transition probabilities T explicitly as they are only considered implicitly in the algorithms we used in this study.

3.2. Simulation Environment

For a simulation environment, we used Google's Balloon Learning Environment developed by the Google Loon team (Greaves et al., 2021), which is designed for the state space, action space, and reward function as we have defined them above. Each training or evaluation episode was a simulated instance of station keeping consisting of 960 three minute steps (48 hours total) whereby a stratospheric balloon attempted to steer in an uncertain wind field using simulated physics. Each episode was initialized with 95% battery charge, a distance away from station sampled from a beta distribution, angle relative to the station sampled uniformly between 0 and 2π, internal pressure sampled uniformly from valid pressures (Bellemare et al., 2020), zero super pressure, latitude sample uniformly between -10 and 10 degrees, longitude sampled uniformly between -175 and 175 degrees, starting time sampled to the second uniformly between 00:00:00 on Jan. 1, 2011 and 00:00:00 on Dec. 31, 2014. In addition, the simulated atmosphere was initialized and winds were simulated with the default parameters of the Balloon Learning Environment. The Balloon Learning Environment's wind simulations are based on the `ble_wind_field` data set (Hersbach, 2017).

3.3. Algorithms and Training

To perform a comparative analysis of the RL techniques used for stratospheric balloon station keeping, we used or attempted to use the following algorithms to solve our RL problem: PPO, DQN, QR-DQN, and SAC (discrete version).

We started with PPO, an on-policy gradient method with a clipped surrogate objective function that ensures updated policies do not change too much compared to the policy in the previous update step. For algorithm details and to see the form of the objective function, see (Schulman et al., 2017). In our implementation, we adapted the PPO algorithm from CS234 Homework 2. We trained a PPO agent on 3500 episodes, and we found that we were able to learn (see Fig. 1). For hyperparameter details, see Appendix A.1.

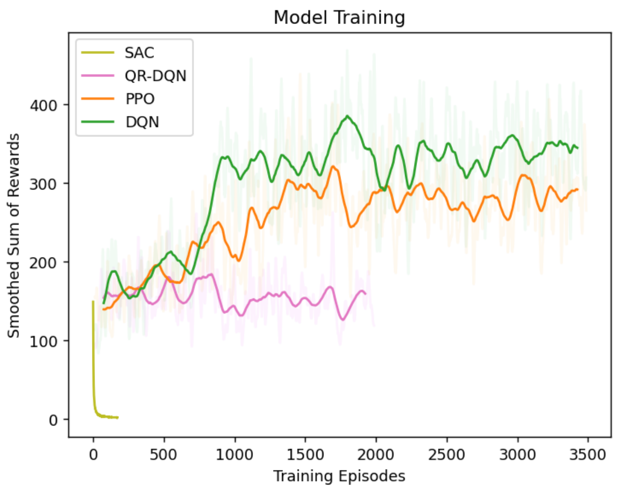

Figure 1. Smoothed learning curves.

Next, we tried applying DQN and QR-DQN agents supplied by the Balloon Learning Environment (Greaves et al., 2021; Castro et al., 2018; Bellemare et al., 2020). DQN is an off-policy, neural network-parameterized Q-learning method characterized by (1) the use of two Q-networks (an online learning network and a target network), and (2) experience replay. For algorithm details, see (Mnih et al., 2013). QR-DQN extends DQN by learning a value distribution instead of a just a value function (i.e., the mean). This was shown to have better empirical results than DQN on Atari games, and we surmised it may show similar gains in the Balloon Learning Environment (Dabney et al., 2018; Bellemare et al., 2020). Unfortunately, we could neither get the DQN nor the QR-DQN agents to learn, despite trying many combinations of hyperparameters. We do not report the DQN results here. See Fig. 1 to see the QR-DQN learning curve over the course of 2,000 training episodes. We noted as we wrote this report that we may have made a bad choice of the random walk exploration probability (0.8; see Section A.3), leading to too much exploration and too little exploitation. In future work, we would try smaller choices of this parameter, in the 0.01 to 0.1 range.

Not wanting to give up on DQN, we next drew inspiration from a PyTorch DQN tutorial and associated Git repository (Paszke & Towers, 2022; 2024). We adapted the code from this tutorial to implement our own DQN agent, and we were successfully able to get this agent to learn over the course of 3,500 episodes of training (see Fig. 1). See Section A.2 for hyperparameter choices.

Finally, we wanted to explore SAC as it was used by two different groups in the literature (Jeger et al., 2023; Saunders et al., 2023; 2024). SAC is an off-policy algorithm that attempts to maximize expected reward while also maximizing entropy in a stochastic actor-critic framework (Haarnoja et al., 2018). First, we needed to adapt SAC to the discrete action setting since it was originally designed for continuous action spaces. We did this by following the guidance in (Christodoulou, 2019; Zhou et al., 2022b) and adapting the code from the associated Git repository (Zhou et al., 2022a). Unfortunately, the SAC learning results were abysmal (see Fig. 1). We were not able to achieve a reasonable learning curve, even one that looked as if it was functional but just wasn't learning. We attributed the issue to a problem with our code that we could not locate within the time allotted for this project.

4. Experimental Results: Evaluation

To evaluate the relative quality of our models, we use three control agents: (1) an agent that performs a random walk by aiming for a random pressure target (uniformly sampled among valid pressures), (2) StationSeeker, a heuristics-based steering program from the Balloon Learning Environment with strong empirical performance (Bellemare et al., 2020), and "Perciatelli," a frozen QR-DQN model developed by Bellemare et al. and cited by those authors as the model with best performance they achieved and close to the best performance possible.

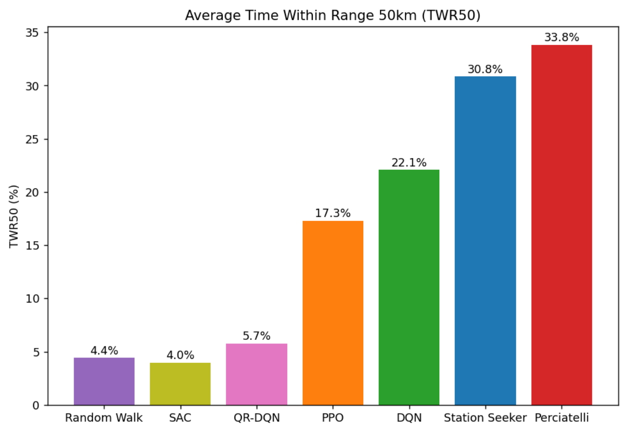

Figure 2. Average TWR50 per evaluation episode.

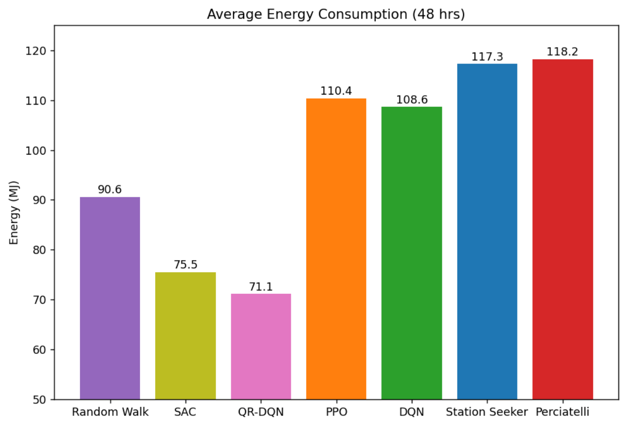

Figure 3. Average energy consumed per evaluation episode.

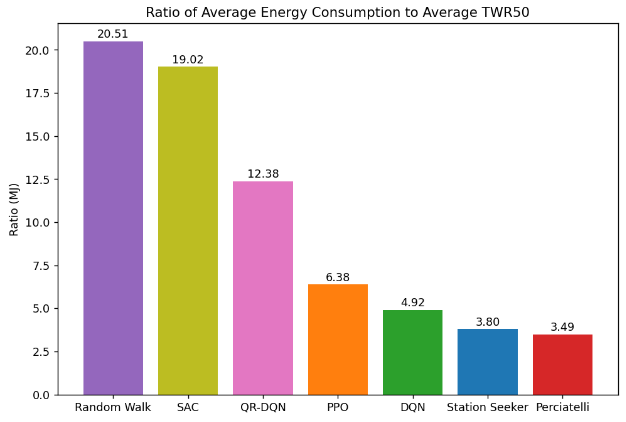

Figure 4. Ratio of average TWR50 to average energy consumed.

We evaluated the three control agents along with the PPO, DQN, QR-DQN, and discrete SAC agents of over the course of 1,000 episodes. For each episode, we calculated (1) the proportion of time spent within a 50 km radius of the station (the TWR50 metric, an industry standard), (2) the total energy consumed by the balloon per episode, and (3) the distance from the balloon to the station at each time increment. We also recorded the 3D trajectories. For each agent, we plot (1) the average TWR50 (Fig. 2), (2) the average energy consumed (Fig. 3), and (3) the ratio of the average TWR50 to the average energy consumed. We also plot a histogram of the distance to station (Fig. 5 shows the near station behavior while Fig. 6 shows the behavior farther away from station). We also display a sample of the paths taken by the balloon while station keeping under the guidance of each type of agent projected in two dimensions (Fig. 7). Fig. 8 shows example paths of the PPO, DQN, and Perciatelli agents' paths in three dimensions. Finally, we produced heat maps depicted the spatial distribution of each

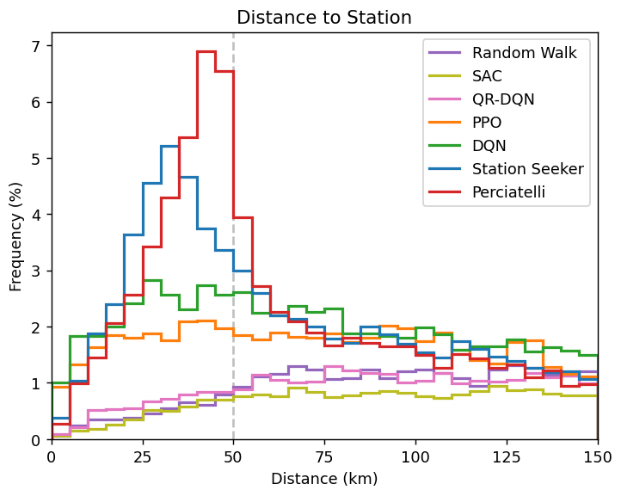

Figure 5. Histogram of distance to station. The data being binned are distances measured at each time step for all time steps in all evaluation episodes. This histogram limits the horizontal axis to 150 km to show the behavior of the agents inside and nearby station.

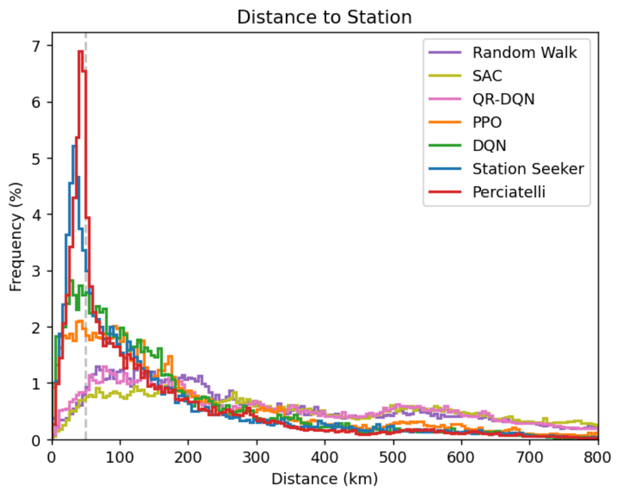

Figure 6. Histogram of distance to station. The data being binned are distances measured at each time step for all time steps in all evaluation episodes. This histogram extends the horizontal axis to 800 km so the tail behavior of each agent can be observed.

agent in and around station (Fig. 9).

From these figures, we make the following observations:

- The positive StationSeeker and Perciatelli controls outperform the Random Walk negative control (higher TWR50 values and lower energy consumption / TWR50 ratios. PPO and DQN perform better than chance, but not as good as StationSeeker and Perciatelli. As expected by its terrible training curve, discrete SAC performs at random walk levels or perhaps slightly worse. See Figs. 2 and 4. Neither of the two random walk and discrete SAC agents are spending energy keeping the balloon near the station.

- From Fig. 2, we note that the average TWR50 values for StationSeeker (30.8%) and Perciatelli (33.8%) are lower than the values for the same cited in (Bellemare et al., 2020) (40.5% and 55.1%, respectively). We are not sure of the reason for the discrepancy. We double-checked that we are using the same environment and reward function parameters as Bellemare et al., and we could not locate a reason for the difference in our results from those reported in the literature.

- From Fig. 5, StationSeeker, PPO, and DQN all keep the balloon relatively closer to station than Perciatelli, whereas Perciatelli allows the balloon to spend more time closer to the 50 km boundary. We interpret this behavior as the agent finding a way to optimize power resources will still keeping within 50 km of station (Bellemare et al., 2020).

- In Fig. 7, StationSeeker and Perciatelli qualitatively look as if they are station keeping well, whereas Random Walk and discrete SAC look like they are wandering aimlessly. As expected, PPO and DQN fall somewhere in between.

5. PPO Discussion

In this work, we trained a PPO agent to keep station, something that had not been done in the literature. In considering why PPO and other policy gradient methods have not been chosen for the station keeping problem, we hypothesize that the reason may have to do with data efficiency. DQN, QR-DQN, and SAC all use experience replay and hence continually learn from prior experiences, whereas PPO learns from newly collected trajectories in an on-policy fashion. This difference is especially significant when the simulated environment is computationally expensive, such as the Balloon Learning Environment (Bellemare et al., 2020; Greaves et al., 2021). Experience replay may also contribute to the underlying quality of agents that are trained with it, as those agents may more easily recover from poor early decisions or difficult initial states. Agents trained with experience replay learn from individual actions, allowing them to learn recovery strategies at a fine granularity and relatively early on. In contrast, PPO trains on returns from entire rollouts, requiring it to explore one or more full recovery trajectories by chance before it can successfully exploit them.

6. Conclusion

In this work, we attempted to perform a comparative analysis of reinforcement learning algorithms used in the literature for the problem of stratospheric balloon station keeping. We were partially successful. Within the Balloon Learning Environment framework, we successfully implemented PPO and DQN agents. While those agents clearly learned, they were not competitive with the state-of-the-art agent reported in Bellemare et al. (Bellemare et al., 2020) (i.e., Perciatelli). On the other hand, our attempts to train and/or implement QR-DQN and discrete SAC were not successful. To truly make our research a comparative analysis of RL methods for station keeping, as is our aim, our future work includes (1) determining the problems with the discrete SAC and QR-DQN agents and engineering them to learn, (2) systematically tuning hyperparameters for better agent performance overall, (3) determining the reason(s) for differences observed between our results and those reported by Bellemare et al., and (4) implementing a dueling DQN agent (since dueling DQN was in the literature, but we did not implement it here).

We would like to mention that we were compute resource limited in performing this research, with training and evaluation runs taking 24-48 hours. In the future, if we are able to access faster compute resources and/or have more time to train and evaluate many, many more models, we expect to make better progress towards our stated goal of completing a true comparative analysis of state-of-the-art RL methods for stratospheric balloon station keeping.

7. Location of Source Code and Model Checkpoints

The source code and model checkpoints for this project can be found on Github at the following hyperlink: `https://github.com/spcorum/student_balloon/`.

8. Statement of Author Contributions

Sean Corum and Jared Watrous overall contributed equally on this project. Sean was mainly responsible for coming up with the original idea, performing research and finding relevant papers and Git repositories, and writing the project proposal, the milestone report, the poster, and the final report. Jared was responsible for coding, training models, performing evaluations, and generating results plots. Jared also helped write the project proposal and researched some Git repositories. Both Sean and Jared collaborated equally on the direction of research of the project as it was being completed.

A. Agent Hyperparameters

Most hyperparameters were adjusted by hand until a combination that seemed to learn as well as we could find compared to other combinations. Network sizes of seven or eight fully connected hidden layers of 600 units wide and discount factor $= 0.993$ were chosen in accordance with best parameters found by Bellemare et al. (Bellemare et al., 2020).

A.1. PPO

For PPO, we used a two neural networks, a policy network and a baseline network, both parameterized by a neural network of seven hidden layers with each layer 600 units wide (each hidden layer consisted of a fully connected linear layer with ReLU rectification), Adam optimization with PyTorch default parameters, 100 Adam optimization steps per network update, a learning rate of 1×10^{-5}, 5 episodes between network updates, a discount factor of 0.993, and a value of ϵ_{clip} of 0.15. Advantages were normalized and calculated with a baseline.

A.2. DQN

For DQN, we used two Q-networks (one online and one target) parameterized by neural networks of eight hidden layers with each layer 600 units wide (each hidden layer consisted of a fully connected linear layer with ReLU rectification), a learning rate of 2×10^{-6}, Adam optimization with PyTorch default parameters, 100 Adam optimization steps per network update, a batch size of 32, a discount factor of 0.993, a replay buffer size of 500,000, a soft update factor for the target network (τ) of 0.05, and a greedy ϵ parameter decayed from 0.9 to 0.05 over 10000 steps (extrapolated).

A.3. QR-DQN

For QR-DQN, we used two Q-networks (one online and one target) parameterized by neural networks of eight hidden with each layer 600 units wide (each hidden layer consisted of a fully connected linear layer with ReLU rectification), a learning rate of 2×10^{-6}, Adam optimization with PyTorch default parameters, 100 Adam optimization steps per network update, a batch size of 32, a discount factor of 0.993, a replay buffer size of 2,000,000, and a random walk exploration probability of 0.8.

A.4. Discrete SAC

For Discrete SAC, we used one actor and two critic networks each parameterized as neural networks of eight hidden with each layer 600 units wide (each hidden layer consisted of a fully connected linear layer with ReLU rectification), a learning rate of 2×10^{-6}, Adam optimization with PyTorch default parameters, 100 Adam optimization steps per network update, a batch size of 64, a discount factor of 0.993, a replay buffer size of 500,000, a soft update factor for the target network (τ) of 0.005, and an entropy weight parameter α of 0.05. The authors in (Zhou et al., 2022b) mention "avg_q", "clip_q", and "entropy_penalty" parameters, which we all set to zero.

References

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110.

Christodoulou, P. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Dabney, W., Rowland, M., Bellemare, M., and Munos, R. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Gannetti, M., Gemignani, M., and Marcuccio, S. Navigation of sounding balloons with deep reinforcement learning. In *2023 IEEE 10th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pp. 591–596. IEEE, 2023.

Greaves, J., Candido, S., Dumoulin, V., Goroshin, R., Ponda, S. S., Bellemare, M. G., and Castro, P. S. Balloon learning environment, 12 2021. URL https://github.com/google/balloon-learning-environment.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Hersbach, H. Complete era5: Fifth generation of ecmwf atmospheric reanalyses of the global climate. copernicus climate change service (c3s) data store (cds), 2017. URL https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-complete?tab=overviewt(accessed01-04-2021).

Jeger, S. L., Lawrance, N., Achermann, F., Pang, O., Kovac, M., and Siegwart, R. Y. Reinforcement learning for outdoor balloon navigation: A successful controller for an autonomous balloon. *IEEE Robotics & Automation Magazine*, 2023.

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

Osprey, S. M. Autonomous balloons take flight with artificial intelligence. 2020.

Paszke, A. and Towers, M. Reinforcement learning (dqn) tutorial, 2022. URL https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.

Paszke, A. and Towers, M. Pytorch tutorials: Reinforcement learning (dqn), 2024. URL https://github.com/pytorch/tutorials/blob/main/intermediate_source/reinforcement_q_learning.py.

Saunders, J., Prenevost, L., Şimşek, Ö., Hunter, A., and Li, W. Resource-constrained station-keeping for helium balloons using reinforcement learning. *arXiv preprint arXiv:2303.01173*, 2023.

Saunders, J., Saeedi, S., Hartshorne, A., Xu, B., Şimşek, Ö., Hunter, A., and Li, W. Identifying optimal launch sites of high-altitude latex-balloons using bayesian optimisation for the task of station-keeping. *arXiv preprint arXiv:2403.10784*, 2024.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Xu, Z., Liu, Y., Du, H., and Lv, M. Station-keeping for high-altitude balloon with reinforcement learning. *Advances in Space Research*, 70(3):733–751, 2022.

Yang, X., Yang, X., and Deng, X. Horizontal trajectory control of stratospheric airships in wind field using q-learning algorithm. *Aerospace Science and Technology*, 106:106100, 2020.

Zhou, H., Lin, Z., Li, J., Fu, Q., Yang, W., and Ye, D. Revisiting-discrete-sac, 2022a. URL https://github.com/coldsummerday/Revisiting-Discrete-SAC.

Zhou, H., Lin, Z., Li, J., Fu, Q., Yang, W., and Ye, D. Revisiting discrete soft actor-critic. *arXiv preprint arXiv:2209.10081*, 2022b.

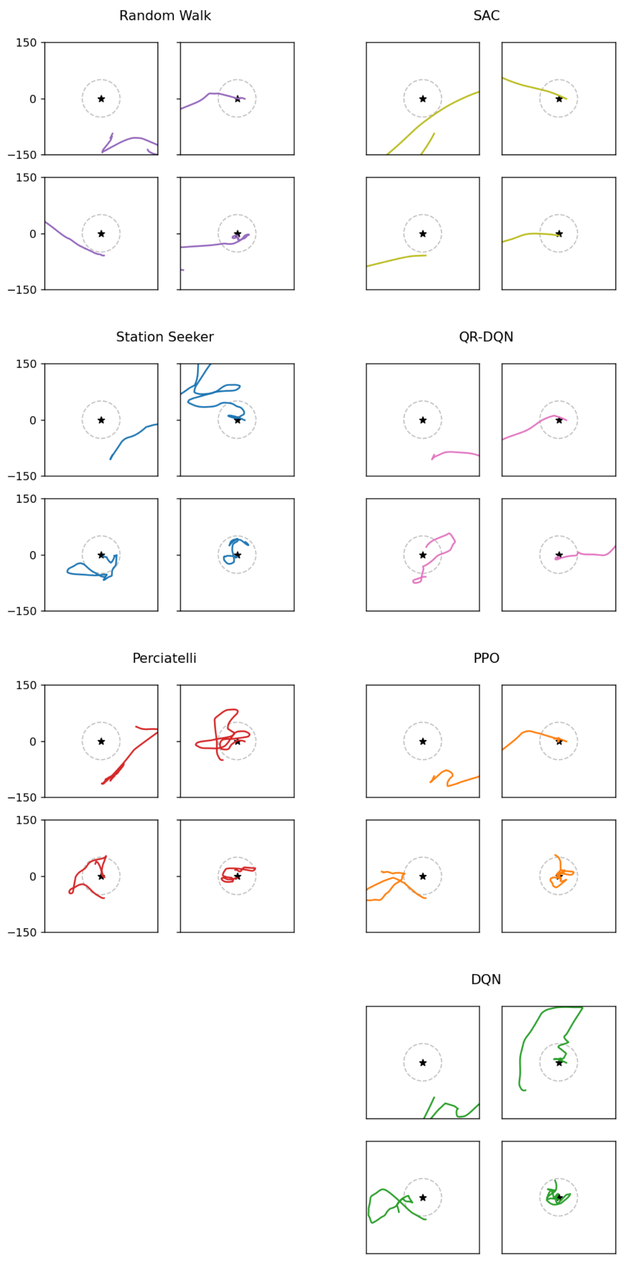

Figure 7. Sampling of two-dimensional paths taken by balloons for four sample episodes for each agent. The space is a Cartesian projection onto longitude (y-axis) and latitude (x-axis). The origin is at the respective station and increments on both axes are in km.

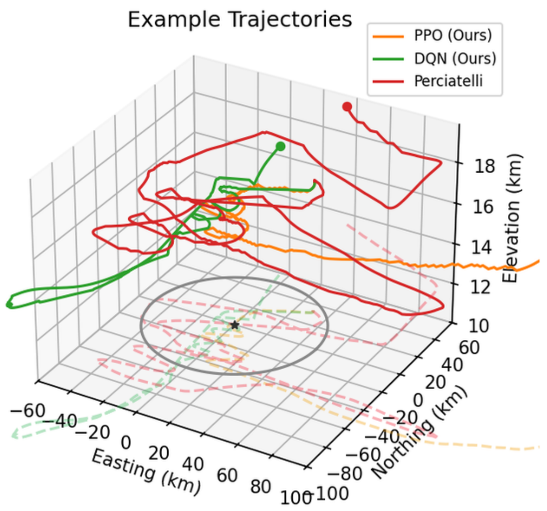

Figure 8. Example three-dimensional paths taken by PPO, DQN, and Perciatelli agents.

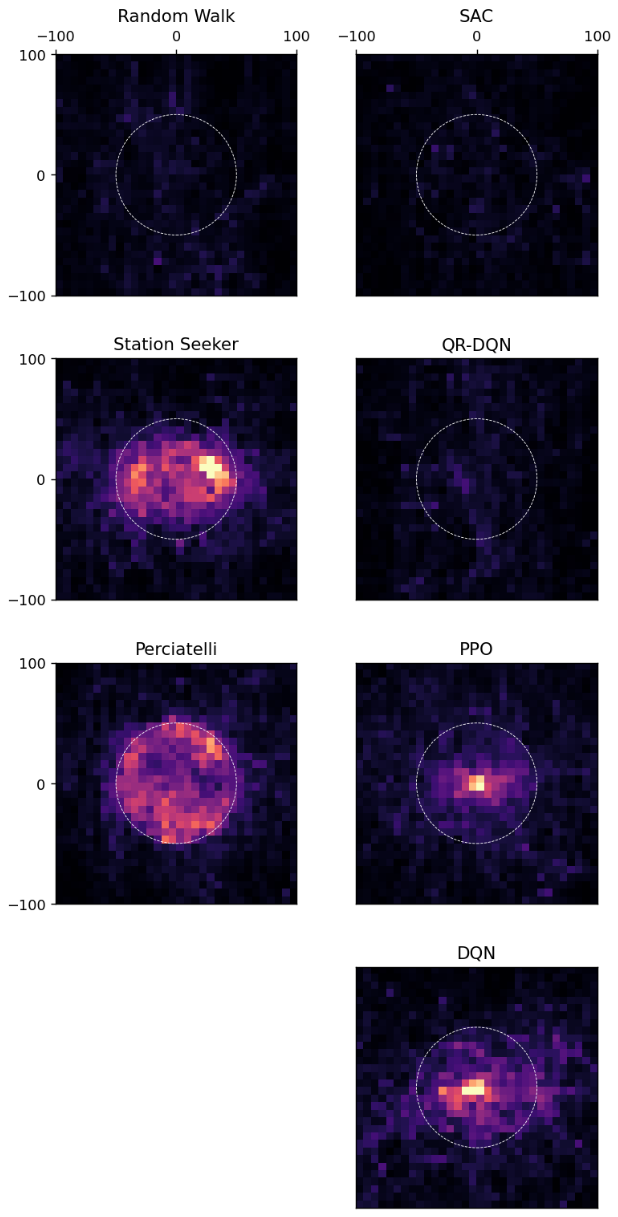

Figure 9. Heatmaps depicting the spatial distribution of each agent in and around station. The color axis varies between 0 and 1,000, and each color unit represents a count of one increment of one episode an agent spends in a cell. The origin is at the center of the station and increments on both axes are in km.