Reinforcement Learning Heuristics for Aerospace Control Systems

Preston K. Robinette
Vanderbilt University
2201 West End Ave.
Nashville, TN, 37235
preston.k.robinette@vanderbilt.edu

Benjamin K. Heiner
Air Force Research Laboratory
2241 Avionics Circle
Wright-Patterson Air Force Base, OH, 45433
benjamin.heiner@us.af.mil

Umberto Ravaioli
Toyon Research Corporation
6800 Cortona Dr.
Goleta, CA, 93117
uravaioli@toyon.com

Nathaniel Hamilton
Vanderbilt University
2201 West End Ave.
Nashville, TN, 37235
nathaniel.p.hamilton@vanderbilt.edu

Taylor T. Johnson
Vanderbilt University
2201 West End Ave.
Nashville, TN, 37235
taylor.johnson@vanderbilt.edu

Kerianne L. Hobbs
Air Force Research Laboratory
2241 Avionics Circle
Wright-Patterson Air Force Base, OH, 45433
kerianne.hobbs@us.af.mil

Abstract— **Reinforcement learning (RL) is a form of machine learning (ML) where an agent is directed to a goal by learning from interactions within an environment. RL has been used to solve previously intractable problems such as Backgammon, Go, and Starcraft and offers a promising class of solutions to complex problems in various domains, including ones involving air and space. One of the major hindrances of using RL as a method is the variability with which an agent trains, which results from deep neural network architectures (e.g., number of inputs, outputs, hidden layers and neurons in each layer, activation functions, connectivity, etc.), algorithm hyperparameters (e.g., batch size, training epochs, learning rate, etc.), goal conditions, and reward functions. A slight modification to any of these factors can have a major impact on the training of the agent and the effectiveness of the learned policy. As such, hyperparameter and architecture searches are common practice in other domains involving neural networks. This work focuses on a systematic approach to RL hyperparameter and architecture optimization for aerospace control systems using direct search, zeroth-order optimization, Bayesian optimization, and asynchronous hyperband-based training methods that are demonstrated in two different safety-critical aerospace environments and tasks, shedding light on general rules of thumb for RL applied to aerospace control systems.**

TABLE OF CONTENTS

1. INTRODUCTION 1
2. AEROSPACE BENCHMARKS 2
3. OPTIMIZATION STRATEGIES 3
4. HYPERPARAMETER AND NEURAL NETWORK ARCHITECTURE OPTIMIZATION SEARCHES 5
5. RESULTS AND DISCUSSION 6
6. CONCLUSIONS AND FUTURE WORK 9
APPENDIX 10
ACKNOWLEDGMENTS 10
REFERENCES 10

978-1-6654-3760-8/22/$31.00 ©2022 IEEE
Approved for public release. Distribution is unlimited. Case Number: AFRL-2021-4042

1. INTRODUCTION

Reinforcement learning (RL) successes in high-dimensional state spaces like Go [1] and real-time strategy games like Starcraft [2] have inspired the application of RL on previously intractable problems and in various domains, including traffic control [3], robotics [4], chemistry [5], and advertising [6]. In the aerospace domain, machine learning (ML) techniques may provide new and more efficient ways to manage air and space traffic, which are projected to become increasingly populated with non-traditional, commercial products like package delivery and agricultural drones, urban air taxis, and mega-constellations of satellites.

The algorithms used in RL and Deep RL (DRL) are based on the concept of operant conditioning [7], in which reinforcement via punishment or rewards is contingent upon behavior. During training, an agent interacts with an environment by taking *actions* as shown in Figure 1. The agent receives feedback from the environment in the form of a *reward* and an *observation* state and uses this feedback to intermittently update its *policy*, which determines the best action to take given an observation. The overall goal of the agent is to maximize the total reward gained in the environment, and it does so by creating and following a policy. [8]. DRL uses the same concepts and strategies as RL but focuses on more complex problems where the solution space, or policy, is approximated by a deep neural network (DNN).

The DNN policy updates are dependent upon the RL algorithm used during training. DRL algorithms come in many different flavors, including model-free vs. model-based and on-policy vs. off-policy. Many of the distinctions among DRL algorithms are due to whether the agent has access to a model of the environment and the question of what to learn in the environment (e.g., policy, action-value function, etc.) This work uses a model-free, policy optimization method known as proximal policy optimization (PPO) [9]. The benefits of PPO compared with other DRL algorithms are that it ensures that policy updates are relatively small through a clipping parameter, has an improved sample efficiency, and has fewer hyperparameters. Although this work utilizes PPO, the optimization methods used are applicable to all RL algorithms. This work uses the PPO algorithm from RLlib

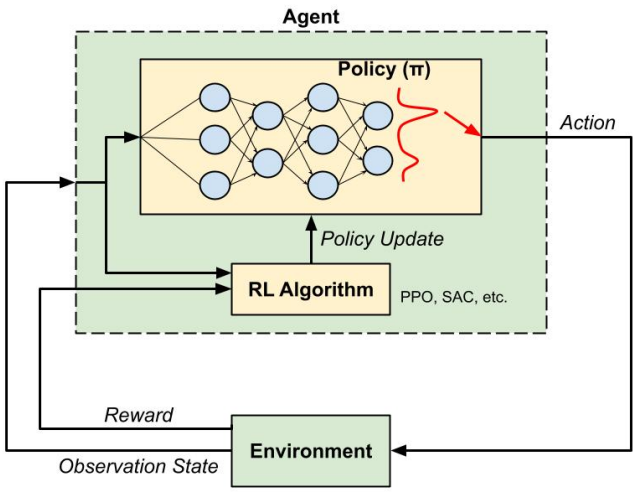


Figure 1: **RL agent and environment interaction.**

[10], which is an open-source RL library built upon Ray that facilitates and scales parallel computing across available computational resources during training.

A significant challenge in RL is optimal parameter selection of both the DNN architecture (e.g., number of inputs, outputs, hidden layers and neurons in each layer, activation functions, connectivity, etc.) and RL algorithm hyperparameters (e.g., batch size, training epochs, learning rate, etc.), which have a significant impact on the success of a learned policy. While there are many types of architectures available, this work looks at feedforward, fully connected neural network architectures. Just as with the RL algorithms, these methods can be applied to different architecture types as well. Hyperparameter optimization (HPO) and architecture optimization (AO) can require experts several months to hand tune and be financially costly [11]. However, automated HPO and AO approaches offer an alternative in an emerging field known as automated machine learning (AutoML) [11]. While AutoML involves the automation of the entire ML pipeline, including data preparation, feature engineering, model generation, and model estimation, this work focuses on the automated approaches to model generation (HPO and AO).

This work builds upon a standard set of aerospace benchmarks [12] to investigate HPO and AO methods for an aircraft formation flight problem and a spacecraft docking problem. The contributions of this work are:

- HPO search across nine learning hyperparameters using five different optimization methods for PPO applied to two aerospace benchmarks,
- AO search across number of hidden layers, number of nodes in each layer, and activation functions for fully connected, feedfoward DNN providing control signals to aerospace benchmarks,
- validation of the results over ten random seeds to reduce stochastic variability impacts, and
- a comparison of the results across the different environments.

The results of this analysis give insight into available HPO and AO methods applied to aerospace RL problems.

The remainder of this paper is structured as follows: Section 2 introduces the two aerospace benchmarks; Section 3 describes the strategies used for optimization; Section 4 outlines the HPO and AO searches used in this work; Section 5 discusses the results of the optimization methods and compares the results of each environment; and finally, section 6 outlines the conclusions and future areas of work related to this paper.

2. Aerospace Benchmarks

The HPO and AO methods discussed in the next section are implemented on two different aerospace RL benchmarks [12], described briefly in this section.

Spacecraft Docking

In the 2D Spacecraft Docking Environment, a deputy spacecraft agent is trained with RL to dock with a passively orbiting chief. The location of the deputy with respect to the chief is expressed in Hill's reference frame [13] $\mathcal{F}_{\mathrm{H}} := (\mathcal{O}_{\mathrm{H}}, \hat{\imath}_{\mathrm{H}}, \hat{\jmath}_{\mathrm{H}})$. This frame's origin $\mathcal{O}_{\mathrm{H}}$ is located at the mass center of the chief, the unit vector $\hat{\imath}_{\mathrm{H}}$ points away from the Earth along a line connecting the center of Earth to $\mathcal{O}_{\mathrm{H}}$, and the unit vector $\hat{\jmath}_{\mathrm{H}}$ is aligned with the orbital velocity vector of the chief. The position and velocity of the deputy relative to the chief are denoted with vectors, $\boldsymbol{r} = x\hat{\imath}_{\mathrm{H}} + y\hat{\jmath}_{\mathrm{H}}$ and $\boldsymbol{v} = \dot{x}\hat{\imath}_{\mathrm{H}} + \dot{y}\hat{\jmath}_{\mathrm{H}}$ respectively. The agent learns to apply thrust of the deputy in the form of $\boldsymbol{u} = F_x\hat{\imath}_{\mathrm{H}} + F_y\hat{\jmath}_{\mathrm{H}}$ to successfully dock. A first order approximation of the relative motion dynamics

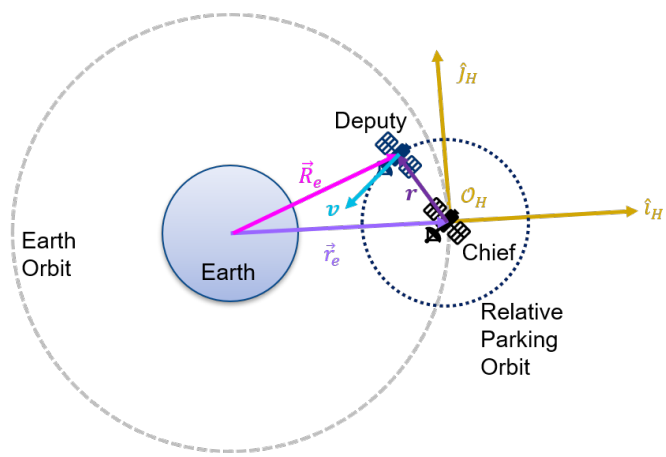


Figure 2: Hill's reference frame centered on a chief spacecraft and used to describe the relative motion of a deputy spacecraft conducting proximity operations (not to scale).

between the deputy and chief spacecraft is given by Clohessy-Wiltshire [14] equations,

$$\begin{aligned} \ddot{x} &= 2n\dot{y} + 3n^2x + \frac{F_x}{m} \\ \ddot{y} &= -2n\dot{x} + \frac{F_y}{m} \end{aligned} \tag{1}$$

where n is spacecraft mean motion and m is the mass of the deputy. The deputy is considered successfully docked when its distance to the chief is less than a desired distance ρ_d.

$$\varphi_{\text{docking}} : (\mathbf{r} \leq \rho_{\mathbf{d}}) \tag{2}$$

Aircraft Formation Flight

In the 2D Dubins Rejoin Environment, a Dubins aircraft agent, the wingman, is trained with RL to fly in formation with a lead aircraft. Figure 3 depicts the goal of the wingman.The state of the scenario $\boldsymbol{x} = [x_L, y_L, \psi_L, v_L, x_W, y_W, \psi_W, v_W]^T \in \mathcal{X} \subset \mathbb{R}^{\leftarrow}$ is the position (x, y), heading ψ, and velocity v of the lead L and wingman W aircraft. The control for the system is defined by $\boldsymbol{u} = [\dot{\psi}_W, \dot{v}_W]^T = [u_{W_1}, u_{W_2}]^T \in \mathcal{U} \subset \mathbb{R}^{\nvDash}$. A two-dimensional Dubins aircraft model is computed using Eq. (3).

$$\begin{aligned}
\dot{x}_L &= v_L \cos\psi_L \\
\dot{y}_L &= v_L \sin\psi_L \\
\dot{\psi}_L &= u_{L_1} \\
\dot{v}_L &= u_{L_2} \\
\dot{x}_W &= v_W \cos\psi_W \\
\dot{y}_W &= v_W \sin\psi_W \\
\dot{\psi}_W &= u_{W_1} \\
\dot{v}_W &= u_{W_2}
\end{aligned} \tag{3}$$

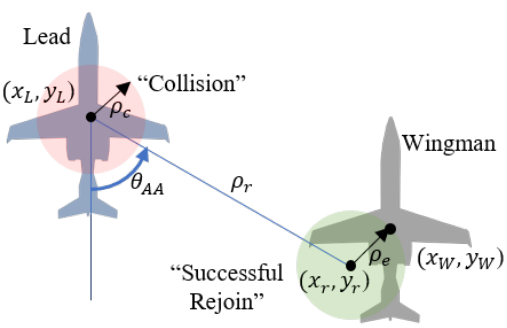


Figure 3: Depiction of general rejoin task

To be successful, the agent must move the wingman aircraft to a position defined by a relative distance ρ_r and an aspect angle θ_{AA} measured from the back of the lead aircraft (-x_L^b) to the vector pointing to the wingman's location. The aspect angle and distance correspond to a formation flight position that a human flight lead would command a wingman to enter. This relative position command can be converted to Cartesian coordinates by Eq. 4.

$$\begin{aligned}
x_r &= x_L + \rho_r \cos(\psi_L + \pi + \theta_{AA}) \\
y_r &= y_L + \rho_r \sin(\psi_L + \pi + \theta_{AA})
\end{aligned} \tag{4}$$

An agent is considered successful if the error between the wingman's position and the commanded rejoin point is less than a specified value ρ_e and if the time in the rejoin position t_{rejoin} is greater than a threshold value t_{success} (in this case 20 seconds).

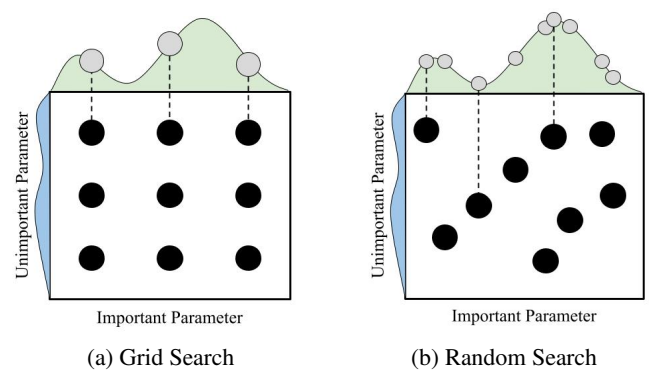
$$\begin{aligned}
\varphi_{\text{formation}} :& \left(\sqrt{(x_W - x_r)^2 + (y_W - y_r)^2} \leq \rho_e\right) \\
& \wedge \left(t_{\text{rejoin}} \geq t_{\text{success}}\right)
\end{aligned} \tag{5}$$

3. Optimization Strategies

There are a growing number of optimization methods and libraries being made available for ML and RL problems as the benefits of model tuning continue to grow. Current HPO and AO automated methods include *direct search* [15–18], *reinforcement learning* [19, 20], *genetic algorithms (GA)* [21, 22], *zeroth-order (ZO)* [23], and *bayesian optimization (BO)* [24–26]. In order to focus time and resources on approaches that are most suited to individual project needs and restrictions, it is important to know the motivations behind available methods.

Search Algorithms

Search algorithms are used to sample configurations from a user defined search space. The search space could include any number of hyperparameters, rewards, and/or model architectures. Zeroth-order (ZO) optimization methods are optimization techniques that do not use loss function gradients [27]. Direct search methods such as grid search (GS) and random search (RS) are special cases of zeroth-order methods. Grid searches are guaranteed to find an optimum in finite discrete search spaces that can be completely explored, but the global optimum in continuous search spaces could be missed entirely as shown in Figure 4a. For continuous



(a) Grid Search (b) Random Search

Figure 4: Direct Search Methods

search spaces, it is more common to use RS as shown in Figure 4b. In RS, a specified number of random combinations in the search space are evaluated as inputs to the indicated function. RS can be effective in large search spaces but wastes time by training bad trials to completion, a factor that could greatly impact RL problems. Another ZO method, univariate search, starts from an initial guess in the search space and samples values along one parameter, stopping at a local minima. At this local minima, the next parameter is sampled until it reaches another local minima. This process is repeated until an extremum is reached as shown in Figure 5. Through sampling, univariate search is able to navigate the function landscape to find optimal parameter configurations. ZO methods are not always the most efficient ways to search

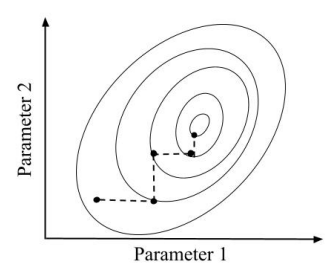


Figure 5: Univariate Search.

the state space if additional information is available; however,

Table 1: **Description and ranges of hyperparamters used in the HPO search**

Hyperparameter	RLlib Variable	Symbol	Range *Min*	Range *Max*	Description
Lambda	*lambda*	λ	0.9	1.0	Lambda reduces the variance of the advantage estimate in training.
VF Loss Coefficient	*vf_loss_coeff*	c_1	0.0001	1.0	The value function loss coefficient determines the importance of value estimation in the complete loss function used to train the agent.
Entropy Coefficient	*entropy_coeff*	a	0.00	0.01	The entropy coefficient sets the importance of exploration in the complete loss function used to train the agent.
Clip Parameter	*clip_param*	ϵ	{0.1, 0.2, 0.3}		The clip parameter is used as a constraint on policy changes. This prevents major, often times drastic, changes in policy from occurring. A clip parameter of 0.3 allows for a greater policy update.
Gamma	*gamma*	γ	0.8	0.9997	The discount factor is a constant that determines the impact of future rewards. The higher the gamma value, the more the agent cares for future rewards.
Learning Rate	*lr*	η	5e-06	0.003	The learning rate, or step size, determines how much a model is changed in the direction of the gradient of the loss function.
Epochs	*num_sgd_iter*	K	10	40	The number of times all of the minibatches are used to update the policy.
Minibatch Size	*sgd_minibatch_size*	M	128	8192	The size of the chunks collected from the train batch size.
Train Batch Size	*train_batch_size*	NT	64	160250	The number of samples, or timesteps, collected before a policy update.

Table 2: **Description and ranges of the features used in the AO search**

Feature	Range	Description
Input layer	fixed to environment	The number of neurons in the input layer is equal to the size of the observation state that is returned from the environment after an action.
Output layer	fixed to environment	The output layer size corresponds to the number of available actions of that environment, 2 in the case of both aerospace benchmarks.
Hidden Layers	{1,2,3,4,5,6}	Hidden layers refer to the number of layers between the input and output layer. For instance, if a DNN has 2 hidden layers, there are two middle layers between the input and output.
Nodes per Layer	{32, 64, 128, 256, 512}	Each hidden layer is made up of a certain number of nodes, or neurons, usually a power of 2.
Activation Function	{linear, ReLU, tanh, swish}	The activation function corresponds to the "firing" of a node. The value of a node for a fully connected, feed forward neural network is the sum of the all the nodes in the previous layer multiplied by their respective weights and added with a bias. The value of the node is then the input to an activation function which calculates if that node is "activated." The activation functions used in this work are show in Figure 6.

they are highly parallelizable.

In Bayesian optimization (BO), an optimizer periodically samples a trial configuration. This trial is trained, and the info from the training is passed back to the optimizer, allowing it to use this information to sample a new configuration. While this method intelligently samples configurations, it is only semi-parallelizable as previous trials are needed to inform the optimizer. Three different BO libraries are used in this work and described below.

1. **BayesOpt:** BayesOpt is an open-source, optimization library for nonlinear problems that uses a Gaussian Process (GP) surrogate model to maximize an indicated function [28]. BayesOpt iteratively fits a GP to a subset of configuration samples, maximizing an aquisition function to select the best hyperparameter values for the next subset of configurations. This process is repeated for the set number of samples indicated by the user.
2. **HyperOpt:** HyperOpt is an optimization open-source library based on BO that is adapted to work with Tree of Parzen Estimators (TPE), Adaptive Tree of Parzen Estimators (ATPE) and Gaussian Process (GP) surrogate models [29,30].
3. **BOHB:** BOHB is an open-source optimization library that uses BO and a scheduler known as HyperBand [31]. BOHB works much in the same way as BayesOpt, but it will automatically stop low performing configurations with rules related to HyperBand, releasing resources for new configurations.

This work applies direct search, ZO, and BO methods through the Ray Tune library[1] [32].

[1]Documentation on the Ray Tune library can be found at https://docs.ray.io/en/latest/tune/index.html

Schedulers

While search algorithms are used to sample configurations, schedulers can be used to stop, clone, or pause trials. Schedulers greatly improve the efficiency of hyperparameter and architecture searches by allocating more time and resources to promising configurations. One such scheduler is asynchronous hyperband (ASHA), which is an early stopping scheduler. ASHA uses a designated metric, such as episode reward mean, to stop trials early. If a trial is not above a threshold after a certain number of training steps, the trial is terminated or paused, and the resources are allocated to a different configuration. In most cases, ASHA can be combined with a search algorithm, but some Ray Tune optimization methods have schedulers built in, such as BOHB which uses HyperBand (HB). HB is a precursor and less aggressive version of ASHA that applies the same methodology. These schedulers are commonly used with GS, RS, and BO search algorithms.

4. Hyperparameter and Neural Network Architecture Optimization Searches

The 2D aircraft and spacecraft aerospace RL benchmarks are iteratively trained using the PPO algorithm from RLlib in the search for optimal hyperparameters and DNN architectures. An optimization method consists of a search algorithm and a scheduler. The optimization methods discussed above are applied in two phases. In the first phase, a hyperparameter search is conducted to find optimal learning hyperparameter values. The second phase builds upon these hyperparameter values to search for optimal feedforward, fully connected DNN architectures. Each of the following search methods were employed:

- Random Search + ASHA,
- ZOOpt (zeroth-order method) + ASHA ,
- BayesOptSearch(Bayesian optimization) + ASHA ,
- HyperOptSearch (Bayesian optimization) + ASHA, and
- BOHB (Bayesian optimization).

Each of these search algorithms and schedulers are implemented from the Ray Tune library [32].

Experimental Setup

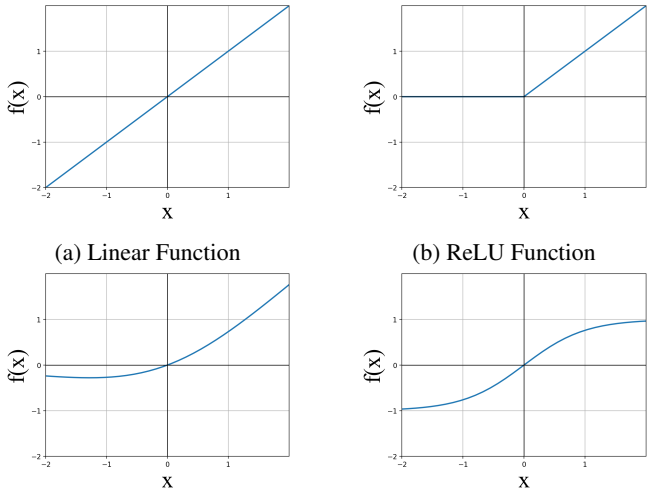
To ensure that improvements in performance were the result of optimized hyperparameters and DNN architectures rather than some other effect, three important considerations were made. First, *unsafe conditions were used as terminal conditions* - all cases terminated if the aircraft or spacecraft crashed, went out of bounds, or exceeded a time limit before successfully completing the task. Second, experiments were *repeated across ten random seeds*. Due to the stochastic nature of RL, showing the results of one trial makes it possible for experimenters to skew results by picking the best performing trial. Running experiments across at least five random seeds and averaging results is important to prove a performance trend [33]. Third, the *initial conditions for each case were equally random* so that no configuration received an advantage for solving the task given "easier" initial conditions than another.

Hyperparameter Search

A description of each of the hyperparameters optimized as well as the range of values considered in the search is listed in Table 1. Each search method is run for 500 samples on 110 CPUS with 10 workers per configuration. The number of workers corresponds to the degree of parallelism during training. By using 10 workers on 110 CPUS, 10 configurations can train at a time (each training configurations also uses 1 CPU for logging). The 10 workers for each training configuration are then used to collect samples in the environment during training in parallel [34]. Each configuration is trained on the same random seed value; however, each worker has a different seed starting sequentially from the random seed value. For instance, worker 0 would use the *random seed value*, worker 1 would then use the *random seed value* + 1, etc. At the end of the search, the top performing configuration for each search method indicated by the evaluation metrics described later is then trained across 10 random seeds, and the results from these 10 runs are then averaged for comparison.

Architecture Search

An architecture for a fully connected DNN consists of an input and output layer, hidden layers, neurons per layer, and an activation function. AO is the process of finding an optimal DNN architecture to successfully complete a task in the environment. Each of the top performing configurations found in the hyperparameter search are used as starting configurations for the AO search. The search space for the AO is shown in Table 2. RS with ASHA is used for each AO across 500 samples. Once again, the top performing configuration is trained across 10 random seeds, and the results from these 10 runs are then averaged.



(a) Linear Function (b) ReLU Function

(c) Swish Function (d) Tanh Function

Figure 6: **Activation Functions**

Evaluation Metrics

The evaluation metrics used to select for top configurations from each optimization search are listed below. These evaluation metrics are chosen to prioritize time, success, and efficiency. The metric indicator (i.e., *low*, *high*) corresponds to the expected trend of top performing configurations.

1. **Episode Length (*low*)** - The average episode length across all episodes in the final iteration of training. Episode length corresponds to the number of timesteps before an episode is terminated. An episode can be terminated because of success (e.g., the goal is completed) or failure (e.g., timeout, crash, out-of-bounds). While episode length is a good indicator of the strength of the policy found, where less time needed

corresponds to a better policy, it can also be short if the agent learns negative behavior. For instance, an agent could learn to immediately go out-of-bounds or crash in order to terminate the episode.

2. **Success Rate (*high*)** - The percentage of episodes in the final iteration of training that successfully completed the goal. Although an essential check for policy correctness, this evaluation metric does not distinguish among top performing configurations, where 100% is the default.

3. **Mean Reward (*high*)** - The mean reward across all episodes in the final iteration of training. The mean reward is a success metric that identifies configurations that are maximizing dense rewards such as the distance change reward and the time reward. Top performing configurations are usually 100% successful in the final iteration, receiving the same positive episode termination reward. Trials with high mean rewards result in policies that maximize dense rewards across all episodes, indicating a more efficient policy. However, some trials may have high rewards by exploiting behavior that was not intended by the designers and could indicate a need to further refine the reward objective function.

4. **Interaction Efficiency Rate (*low*)** - The percentage of all total interactions, or actions, needed to reach an 80% success rate during training. If a configuration has a low interaction efficiency rate, it is finding and utilizing a successful policy faster compared to other trials. In some case this early performance gain could be due to random chance, for example a lucky random seed. However, when averaged over many random seeds, efficiency can be more accurately evaluated.

By using all four metrics across 10 random seeds, the strengths and weaknesses of each metric are balanced in the AO and HPO searches.

5. Results and Discussion

This section describes the best hyperparameters found using each of the 5 combinations of HPO and AO search methods (RS + ASHA, ZOOpt + ASHA, BayesOptSearch + ASHA, HyperOptSearch + ASHA, and BOHB), as well as the performance of DNNs trained using each of the four evaluation metrics (episode length, success rate, mean reward, and interaction efficiency rate) for the Docking and Rejoin environments. A summary of the best performing hyperparameter combinations for the Docking and Rejoin environments found using 500 trials by 5 different methods (2500 total configurations) are shown in Table 5. While some of the hyperparameters in the best performing configurations are very close to the default values in RLlib (e.g. epochs, lambda, gamma), other parameters showed significant deviations. In particular, the best Docking configuration had a training batch size nearly an order of magnitude larger than the default, while the best Rejoin train batch size was double the default value. This means that many more episodes were used to inform each stochastic gradient descent update of the DNN.

Docking Benchmark

The results of the AO and HPO search for the 2D Spacecraft Docking Benchmark are shown by their performance on each of the four metrics in Figure 7. The hyperparameters and DNN architecture for the default RLlib PPO as well as the top set found using each method are shown in Table 3. The best performing configuration during the HPO indicated by the evaluation metrics (lowest episode length, highest reward, and most efficient interactions) was found using the RS algorithm with the ASHA scheduler (RS+ASHA), which had a 2x shorter episode length, 2x greater mean reward, and an 8x greater interaction efficiency compared to the default policy. The optimized architectures represented by the blue striped bars show improved performance for each of the hyperparameter configurations chosen during the HPO search, excluding the default configuration and RS+ASHA. This demonstrates that both HPO and AO searches generally improve performance.

The configuration found using the ZOOpt search algorithm with the ASHA scheduler (ZOOpt+ASHA) also performs significantly better compared to the default hyperparameter values and DNN architecture for RLlib's PPO algorithm. One key distinction between the RS+ASHA and the ZOOpt+ASHA trials is the train batch size and minibatch size. The top hyperparameters and DNN architecture RS+ASHA utilizes a train batch size of 39064 and a minibatch size of 256, whereas ZOOpt+ASHA utilizes a train batch size of 10564 and a minibatch size of 2048. RS+ASHA, therefore, is split into 152 chunks for an update, and ZOOpt+ASHA is split into 5 chunks per update. There are more updates occurring in the RS+ASHA configuration than that of the ZOOpt+ASHA resulting in a much longer training time. Although, the performance of the RS+ASHA is stronger, the ZOOpt+ASHA configuration is preferred due its shorter training time. While ZOOpt+ASHA and RS+ASHA demonstrate the best performance compared to the default configuration, all five optimization methods improved upon the default configuration performance.

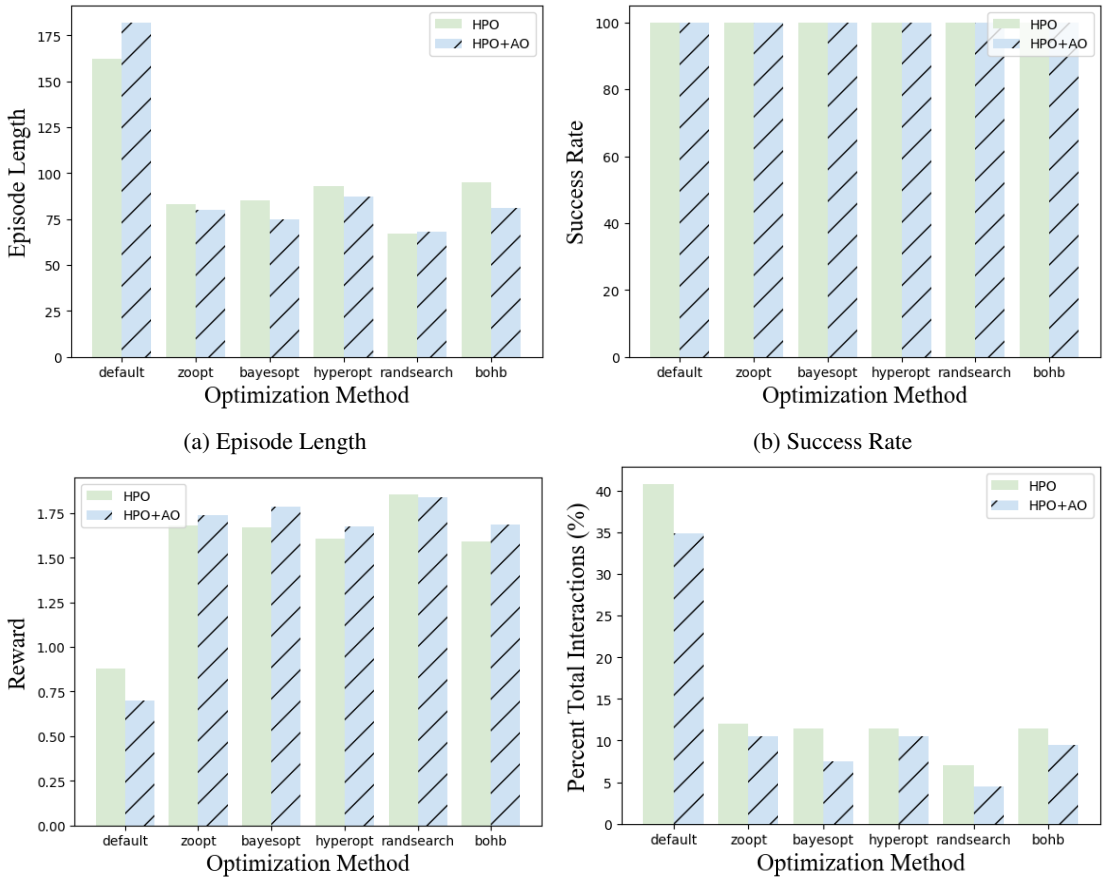
To give an idea of how the hyperparameters and DNN architectures translated to trajectories of the spacecraft under the RL agent's control, random rollouts (one simulated case of docking) for each of the selected policies are shown in Figure 8. There are some instances where the rollout of one of the optimized cases is close to the default case's performance, such as in policy rollout 2, while others like policy rollout 6 indicate how the default trajectory in magenta takes a much longer path than any of five optimized cases. All of the optimized configurations demonstrate better policies (more efficient) compared to the default configuration rollouts.

Rejoin Benchmark

The results of the AO and HPO search for the 2D Dubins Rejoin environment are shown in Figure 9. The hyperparameters and DNN architecture for the default RLlib PPO as well as the top set found using each method are shown in Table 4 The best performing configuration for the 2D Dubins Rejoin Benchmark was found using the ZOOpt+ASHA method. Although not as distinct of an improvement as that found with the 2D Spacecraft Docking Environment, the episode length is shorter, mean reward greater, and interaction efficiency smaller as well.

A random rollout for the optimized configurations is shown in Figure 10. Here the policy from the default configuration crashes. While the configurations from the HPO search are all successful, the ZOOpt+ASHA policy is more efficient. The next best policy is found with the RS+ASHA method. The results of the architecture search are indicated by the blue striped bars in Figure 9. The AO improves performance for each of the hyperparameter configurations chosen during the HPO search, except for HyperOpt with ASHA.

An important distinction between the results for each environment is that not all optimization methods performed better



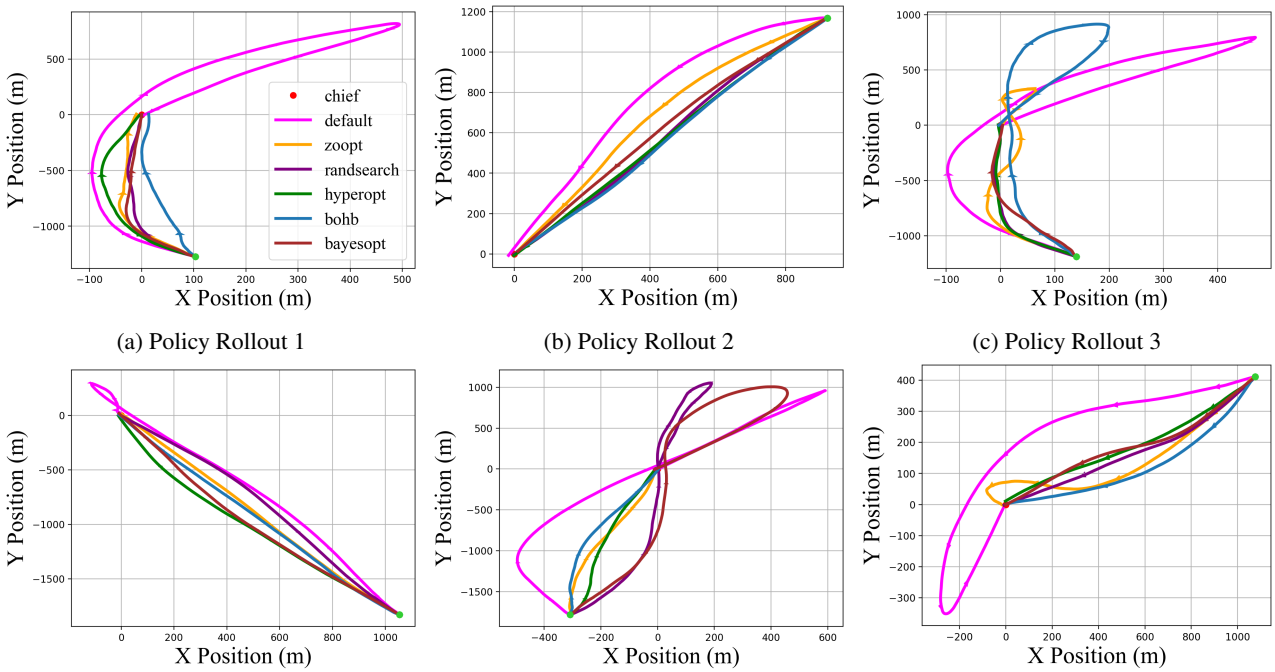
(a) Episode Length

(b) Success Rate

(c) Mean Reward

(d) Interaction Efficiency

Figure 7: **2D Spacecraft Docking Environment Results**



(a) Policy Rollout 1

(b) Policy Rollout 2

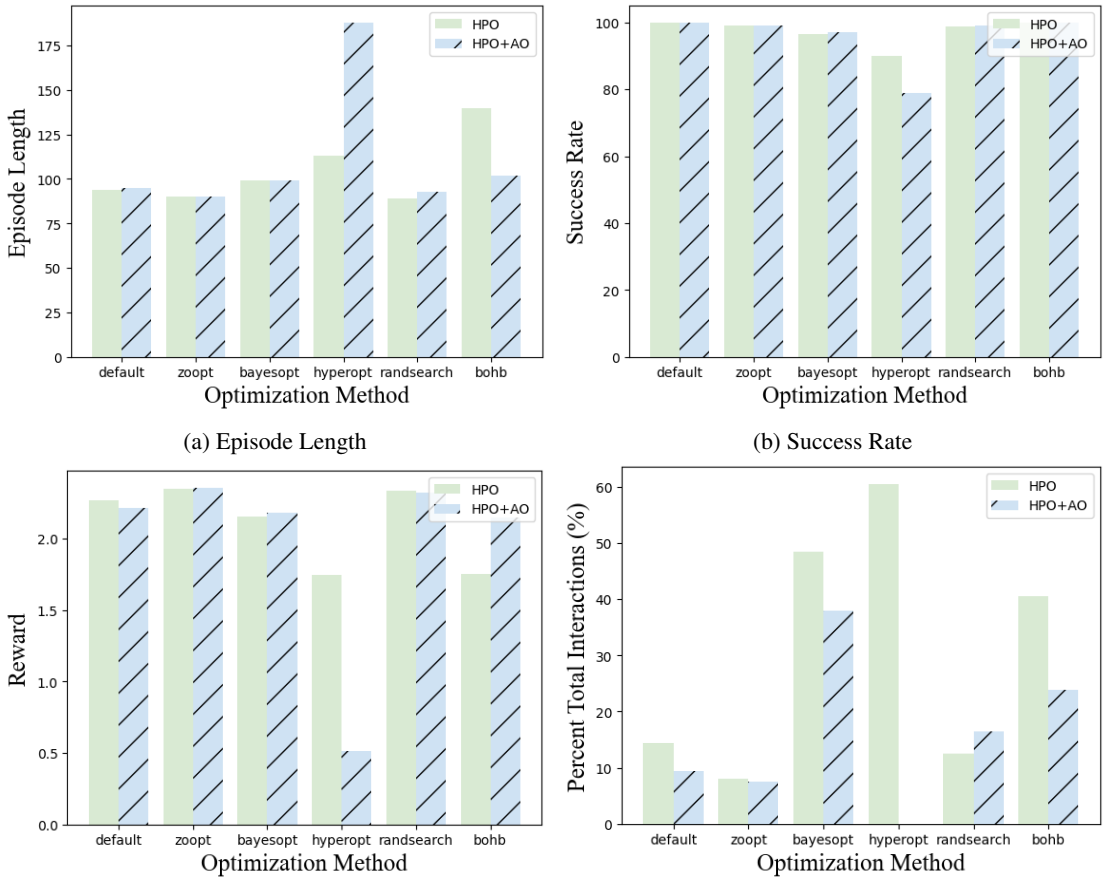
(c) Policy Rollout 3

(d) Policy Rollout 4

(e) Policy Rollout 5

(f) Policy Rollout 6

Figure 8: **Docking Policy Rollouts with HPO**



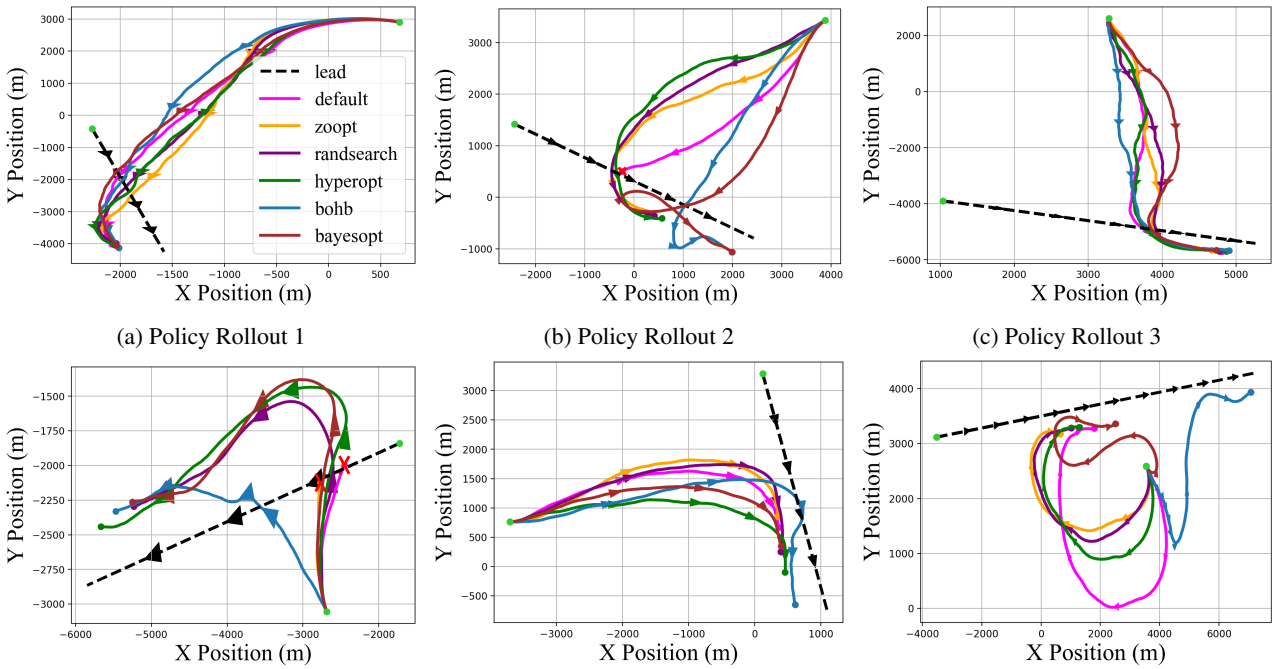
(a) Episode Length

(b) Success Rate

(c) Mean Reward

(d) Interaction Efficiency

Figure 9: **2D Dubins Rejoin Environment Results**



(a) Policy Rollout 1

(b) Policy Rollout 2

(c) Policy Rollout 3

(d) Policy Rollout 4

(e) Policy Rollout 5

(f) Policy Rollout 6

Figure 10: **Rejoin Policy Rollouts with HPO**

Table 3: **Spacecraft Docking Environment PPO Hyperparameter Values with Various Optimization Methods**

Hyperparameter	RLlib Variable	Symbol	Docking Default	Docking ZOOpt (ASHA)	Docking BayesOpt (ASHA)	Docking HyperOpt (ASHA)	Docking RS (ASHA)	Docking BOHB
Lambda	*lambda*	λ	1.0	0.904496	0.934828	0.932390	**0.94334**	0.976152
VF Loss Coefficient	*vf_loss_coeff*	c_1	1.0	0.083059	0.403379	0.237571	**0.678487**	0.564072
Entropy Coefficient	*entropy_coeff*	a	0.0	0.004315	0.0	0.005741	**0.002971**	0.002325
Clip Parameter	*clip_param*	ϵ	0.3	0.2	0.3	0.2	**0.2**	0.2
Gamma	*gamma*	γ	0.99	0.988633	0.977449	0.974967	**0.991423**	0.956411
Learning Rate	*lr*	η	0.00005	0.001344	0.00005	0.000659	**0.000082**	0.000981
Epochs	*num_sgd_iter*	K	30	34	30	25	**28**	20
Minibatch Size	*sgd_minibatch_size*	M	128	2048	226	512	**256**	1024
Train Batch Size	*train_batch_size*	NT	4000	10564	16358	10564	**39064**	19314
Hidden Layers	*fcnet_hiddens*		[512,256,256,64]	[512,256]	[512,256,32]	[512,256]	**[512,256,32,32,32]**	[512,128]
Activation Function	*fcnet_activation*		tanh	tanh	tanh	tanh	**tanh**	tanh

Table 4: **Dubins Rejoin Environment PPO Hyperparameter Values with Various Optimization Methods**

Hyperparameter	RLlib Variable	Symbol	Rejoin Default	Rejoin ZOOpt (ASHA)	Rejoin BayesOpt (ASHA)	Rejoin HyperOpt (ASHA)	Rejoin RS (ASHA)	Rejoin BOHB
Lambda	*lambda*	λ	1.0	**0.923305**	0.911153	0.940532	0.923414	0.963833
VF Loss Coefficient	*vf_loss_coeff*	c_1	1.0	**0.347996**	0.953926	0.008325	0.072642	0.373403
Entropy Coefficient	*entropy_coeff*	a	0.0	**0.002276**	0.0	0.007593	0.000163	0.004901
Clip Parameter	*clip_param*	ϵ	0.3	**0.3**	0.3	0.2	0.3	0.3
Gamma	*gamma*	γ	0.99	**0.990064**	0.878836	0.997807	0.987031	0.908378
Learning Rate	*lr*	η	0.00005	**0.000255**	0.00005	0.000845	0.000435	0.000715
Epochs	*num_sgd_iter*	K	30	**27**	30	28	26	11
Minibatch Size	*sgd_minibatch_size*	M	128	**1024**	1236	4096	256	512
Train Batch Size	*train_batch_size*	NT	4000	**8814**	7851	5064	6064	2814
Hidden Layers	*fcnet_hiddens*		[256,128,128]	**[512,512,32]**	[256,256,32,32,32,32]	[128,128,32,32]	[128,128,32,32]	[128]
Activation Function	*fcnet_activation*		tanh	**tanh**	relu	relu	swish	tanh

than the default configuration in the Rejoin environment. Each of the methods utilizing a Bayesian optimization search algorithm chose a hyperparameter configuration that performed worse than the default. This could be due to sample size, as the optimizer did not have enough samples to train on in order to find a definitive pattern in the hyperparameter search.

6. Conclusions and Future Work

The results of this work demonstrate the benefit of using HPO and AO methods for aerospace RL benchmarks, increasing performance and consistency in training and evaluation results. Although the results of this work align with Andrychowicz et als. recommendation to start with a tanh activation function [35], there is not enough information to lay claim to a consistent set of hyperparameters and architectures for aerospace reinforcement learning problems. This work also highlights the effectiveness of specific optimization methods on aerospace systems, particularly RS with ASHA and ZOOpt with ASHA. There are many interesting avenues of exploration related to this work, including methodology, environment benchmarks, and safety.

Table 5: **Comparison of the Best Hyperparameters found for the Docking and Rejoin Environments**

Hyperparameter	Default	Docking	Rejoin
Lambda	1.0	0.94334	0.923305
VF Loss Coefficient	1.0	0.678487	0.347996
Entropy Coefficient	0.0	0.002971	0.002276
Clip Parameter	0.3	0.2	0.3
Gamma	0.99	0.991423	0.990064
Learning Rate	0.00005	0.000082	0.000255
Epochs	30	28	27
Minibatch Size	128	256	1024
Train Batch Size	4000	39064	8814

Methodology

Future work could explore additional search and optimization methods. While this work applied five different search algorithms, there are other promising search algorithms and schedulers available that may lead to better results. Valuable future work includes utilizing and comparing all available method such as *AxSearch*, *NevergradSearch*, *OptunaSearch*, etc. In addition to including more optimization methods, there is also benefit in exploring the order of the searches. In neural architecture searches, a branch of AutoML, the architecture is optimized first and then the hyperparameter values. It would be interesting to see if performing the AO search before the HPO search improves overall performance. There is also room to explore the effect of performing both searches at the same time instead of sequentially. This topic is not as widely explored.

Environment Benchmarks

Future work could explore benchmarks with larger observation and action spaces. The environments used in this work have small observation and action spaces. To improve the value of the conclusions in regard to optimization methods, it would be beneficial to see how the searches respond in more complex environments. This includes making the environments harder to solve (e.g., larger range in initial position, faster speeds, smaller docking region) and using environments with larger observation and action spaces such as the 3D Spacecraft Docking Environment and the 3D Dubins Rejoin Environment.

Safety

Along the same line as environment complexity, future areas of work include using these optimization techniques in the presence of a run-time assurance (RTA) module during training. While the added RTA module does not have tunable hyperparameters, it could significantly affect the training process, possibly altering the benefit of using the corresponding optimization methods.

There are many important avenues of potential research related to this work. HPO and AO are an imperative part of the future of machine learning and RL success. Relying on fewer experts to significantly improve performance will facilitate a much larger impact on machine learning and RL technique application in the aerospace domain.

APPENDIX

The configuration of the Rejoin and Docking environments are described in Tables 6 and 7.

Table 6: **Rejoin Default Environment Configurations**

Lead	
Velocity (ft/s)	[10,100]
Velocity @t_0 (ft/s)	[40,60]
x,y Position @t_0 (ft)	[-4000,4000]
Heading @t_0 (rad)	[0,2π]
Wingman	
Velocity	[10,100]
State Reference	lead aircraft
Rel. Velocity @t_0 (ft/s)	[10,100]
Rel. Distance @t_0 (ft)	[1000,10000]
Rel. Angle @t_0 (rad)	[0,2π]
"Rudder" $\dot{\psi}$ (rad/s)	[-0.1,0.1]
$\dot{v}$ (throttle, ft/s^2)	[-10,10]
Rejoin Region	
State Reference	lead aircraft
Aspect Angle θ_{AA} (deg)	60
Relative Distance ρ_r (ft)	500
Region Radius ρ_e(ft)	150

Table 7: **Docking Default Environment Configurations**

Chief	
x_0,$\dot{x}_0$, y_0, $\dot{y}_0$, z_0, $\dot{z}_0$	0
Deputy	
Velocity	$\left[0,\ 0.2 + 2n\sqrt{x^2+y^2}\right]$
State Reference	Chief
Rel. Distance @t_0 (ft)	[100,150]
Rel. Angle @t_0 (rad)	[0,2π]
"Thrust" F_x/m (m/s^2	[-1,1]
"Thrust" F_y/m (m/s^2	[-1,1]
Docking Region	
State Reference	Chief
x offset (m)	0
y offset (m)	0
radius (m)	0.5

ACKNOWLEDGMENTS

This material is based upon work supported by the Air Force Research Laboratory Innovation Pipeline Fund and administered by the the Autonomy Technology Research (ATR) Center and Wright State University. In particular the authors would like to thank Sean Mahoney whose support contributed to the success of this project. The views expressed are those of the authors and do not reflect the official guidance or position of the United States Government, the Department of Defense or of the United States Air Force.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search,"

nature, vol. 529, no. 7587, pp. 484–489, 2016.

[2] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[3] F. Rasheed, K.-L. A. Yau, R. M. Noor, C. Wu, and Y.-C. Low, "Deep reinforcement learning for traffic signal control: A review," *IEEE Access*, 2020.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[5] Z. Zhou, S. Kearnes, L. Li, R. N. Zare, and P. Riley, "Optimization of molecules via deep reinforcement learning," *Scientific reports*, vol. 9, no. 1, pp. 1–10, 2019.

[6] J. Jin, C. Song, H. Li, K. Gai, J. Wang, and W. Zhang, "Real-time bidding with multi-agent reinforcement learning in display advertising," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 2193–2201.

[7] B. F. Skinner, "Operant behavior." *American psychologist*, vol. 18, no. 8, p. 503, 1963.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

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

[10] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3053–3062.

[11] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[12] U. Ravaioli, J. Cunningham, J. McCarroll, K. Dunlap, V. Gangal, and K. L. Hobbs, "Safe reinforcement learning benchmark environments for aerospace control systems," *IEEE Aerospace*, 2022.

[13] G. W. Hill, "Researches in the lunar theory," *American journal of Mathematics*, vol. 1, no. 1, pp. 5–26, 1878.

[14] W. Clohessy and R. Wiltshire, "Terminal guidance system for satellite rendezvous," *Journal of the Aerospace Sciences*, vol. 27, no. 9, pp. 653–658, 1960.

[15] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," *arXiv preprint arXiv:1902.08142*, 2019.

[16] A. Hundt, V. Jain, and G. D. Hager, "sharpdarts: Faster and more accurate differentiable architecture search," *arXiv preprint arXiv:1903.09900*, 2019.

[17] Y. Geifman and R. El-Yaniv, "Deep active learning with a neural architecture search," *arXiv preprint arXiv:1811.07579*, 2018.

[18] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 367–377.

[19] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[20] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[21] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.

[22] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[23] Y.-R. Liu, Y.-Q. Hu, H. Qian, Y. Yu, and C. Qian, "Zoopt/zoojl: Toolbox for derivative-free optimization," *arXiv preprint arXiv:1801.00329*, 2017.

[24] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 528–536.

[25] S. Falkner, A. Klein, and F. Hutter, "Practical hyperparameter optimization for deep learning," 2018.

[26] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Workshop on Automatic Machine Learning*. PMLR, 2016, pp. 58–65.

[27] S. Liu, P.-Y. Chen, B. Kailkhura, G. Zhang, A. O. Hero III, and P. K. Varshney, "A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications," *IEEE Signal Processing Magazine*, vol. 37, no. 5, pp. 43–54, 2020.

[28] R. Martinez-Cantin, "Bayesopt: a bayesian optimization library for nonlinear optimization, experimental design and bandits." *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3735–3739, 2014.

[29] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: a python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, no. 1, p. 014008, 2015.

[30] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[31] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1437–1446.

[32] "Ray tune: Scalable hyperparameter tuning." [Online]. Available: https://docs.ray.io/en/latest/tune/index.html

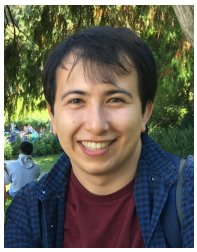
[33] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[34] "Rllib training apis." [Online]. Available: https://docs.ray.io/en/latest/rllib-training.html

[35] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski *et al.*, "What matters in on-policy reinforcement learning? a large-scale empirical study," *arXiv preprint arXiv:2006.05990*, 2020.

Preston Robinette *is a second-year Ph.D. student in Computer Science at Vanderbilt University and a National Defense Science and Engineering Graduate (NDSEG) Fellow. Her current research interests include machine learning, safety, security, and game theory. Preston received her BS in Physics from Presbyterian College where she received a NASA Undergraduate Research Award for her work in myoelectric control of prosthetics and robotics.*

Benjamin K. Heiner *is the Autonomous Air Combat Operations (AACO) AI Training Lead on the Autonomy Capability Team (ACT3) at the Air Force Research Laboratory (AFRL). There he leads researchers and developers as it pertains to the development of Deep RL framework for training and evaluating AI-based tactical autopilots for ACT3 AACO relevant scenarios. His previous experience includes work in image and natural language processing (NLP) based helicopter engine predictive maintenance, targeted 3D modeling from UAV imagery, large georeferenced mosaics from MAV Video and telemetry data, and image/signal processing for undersea systems. Benjamin has a BS in Computer Engineering Technology from Oregon Institute of Technology, an BS in Software Engineering Technology from Oregon Institute of Technology, and MS in Computer Engineering from Brigham Young University.*



Umberto Ravaioli *is an Analyst at Toyon Research Corporation and the AI Lead on the Safe Autonomy Team at Air Force Research Laboratory's Autonomy Capability Team (ACT3). There, he leads the overall effort of designing/developing the Aerospace SafeRL Framework, training RL agents, and building Runtime Assurance algorithms. At Toyon, he has contributed to DOD research efforts including Deep Learning Aerial Target Recognition with EO+LIDAR fusion, image based helicopter engine predictive maintanence, and GPS denied vision-aided navigation. Umberto received his BS in Electrical Engineering and MS in Computer and Electrical Engineering from the University of Illinois at Urbana-Champaign.*

Nathaniel Hamilton *is a Ph.D. student and National Defense Science and Engineering Graduate (NDSEG) Fellow at Vanderbilt University. His research focuses on Safe RL and how it can be used to deploy safe and robust learning-enabled controllers on real-world systems. His previous experience includes work in Transfer RL and robotics navigation and control. Nathaniel has a BS in Electrical and Computer Engineering from Lipscomb University and an MS in Electrical Engineering from Vanderbilt University.*

Taylor T. Johnson *is an Associate Professor of Computer Science at Vanderbilt University, where he also serves as Senior Research Scientist in the Institute for Software Integrated Systems. He holds a BSEE from Rice University and PhD/MSc from the University of Illinois at Urbana-Champaign, all in Electrical and Computer Engineering. His research interests are in formal verification for safe autonomy.*

Kerianne Hobbs *is the Safe Autonomy Lead on the Autonomy Capability Team (ACT3) at the Air Force Research Laboratory. There she investigates rigorous specification, analysis, and bounding techniques to enable certification of autonomous and learning controllers for aircraft and spacecraft applications. Her previous experience includes work in automatic collision avoidance and autonomy verification and validation research. Kerianne has a BS in Aerospace Engineering from Embry-Riddle Aeronautical University, an MS in Astronautical Engineering from the Air Force Institute of Technology, and a Ph.D. in Aerospace Engineering from the Georgia Institute of Technology.*