AAS 23-026

PLANNING AUTONOMOUS SPACECRAFT RENDEZVOUS AND DOCKING TRAJECTORIES VIA REINFORCEMENT LEARNING*

Vincent Chen,† Sean A. Phillips,‡ and David A. Copp†

We present a Proximal Policy Optimization (PPO) Reinforcement Learning algorithm for three-dimensional autonomous spacecraft trajectory planning. Specifically, we consider a chaser spacecraft performing a rendezvous and docking mission with a target spacecraft on a circular orbit. This reinforcement learning approach utilizes an actor and critic method to plan safe trajectories for the chaser spacecraft given constraints on its motion, including maximum thrust and line-of-sight constraints. We consider a fully actuated chaser spacecraft capable of applying continuous thrust in all three dimensions. Given this action space, we train a PPO model to perform rendezvous and docking maneuvers using spacecraft relative motion dynamics. We describe the training procedure and environment in detail and present results of numerous simulations showing that the trained model produces successful rendezvous trajectories that satisfy line-of-sight constraints, even when starting far from the target with significant random variations in initial positions and velocities. Finally, we present statistics on performance, including the terminal state reached, mission time, fuel consumption, and computation time.

INTRODUCTION

As the number of applications for autonomous spacecraft increases, so does the need for path planning algorithms that are safe and can adapt to complex and dynamic environments. One class of such applications is spacecraft Autonomous Rendezvous, Proximity Operations, and Docking (ARPOD) missions, where a chaser spacecraft is tasked with rendezvous and docking with a passive (non-actuated) target spacecraft on orbit. The rendezvous and docking operation mainly consists of three phases: an approach, an alignment, and a securing of the attachment. The different phases of an ARPOD mission are shown in Figure 1. These missions are challenging due to a number of factors including uncertain environments, actuation constraints, and limited sensing capability. Therefore, planning safe trajectories for a chaser spacecraft that are also time and fuel efficient is an important and challenging problem.

There have been many recent advances in trajectory optimization for space vehicle control,[1] all with varying sensing and actuation constraints. Perhaps one of the first trajectory optimization algorithms for satellite reconfiguration maneuvers with position and attitude constraints was proposed in (Ref. 2). Feedback control methods, such as Model Predictive Control (MPC) have proved successful in ARPOD missions, and there is significant literature on the topic.[3–6] Although MPC formulations have shown robust results in each individual phase of an ARPOD mission, discontinuities between phases can affect performance and feasibility of latter phases. This can be particularly

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED. PUBLIC AFFAIRS APPROVAL #AFRL-2023-1822.

†University of California, Irvine. Irvine, CA 92697, USA.

‡Space Vehicles Directorate, Air Force Research Laboratory, Kirtland AFB, NM 87117, USA. 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.

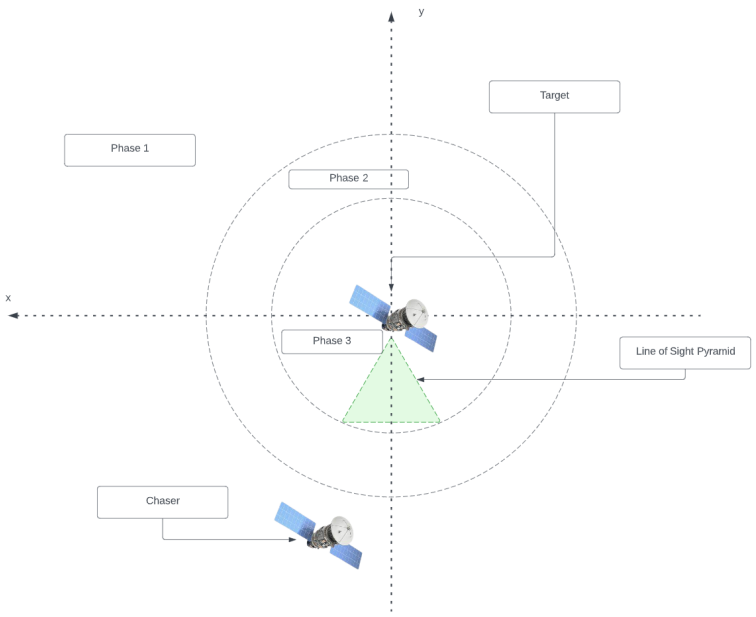


Figure 1: Depiction of ARPOD mission phases. Phase 1 of the mission is the initial phase of the rendezvous maneuver when only relative angle measurements are available. Phase 2 involves the rendezvous maneuver when both relative angle and range measurements are available. Phase 3 is the docking phase.

problematic between phases 2 and 3, where the chaser transitions from approach velocities to a more constrained docking maneuver. Additionally, with discrete phases, motion near the edge of a phase transition may result in undesired cyclic policy switching between phase definitions. Often the mission dynamics and constraints are simplified to enable MPC approaches that are discrete time, linear, and convex to improve feasibility and computational performance for real-time application. Therefore, there is room to improve existing algorithms for autonomous rendezvous and docking that result in higher fidelity simulations and formal safety and performance guarantees.

Learning algorithms may be advantageous for addressing these challenges and can perform well in unknown and complex environments. There are several examples of existing work that apply reinforcement learning techniques to spacecraft guidance, navigation, and control problems. For example, learning algorithms have been applied to individual phases of the docking missions without obstacles.[7] A policy for 3-DoF proximity operations using a deep policy gradient method was recently developed and experimentally validated,[8] and the effect of run time assurance on reinforcement learning training performance for satellite docking problems has been studied.[9] Surprisingly, not much work has been done on generating full mission trajectories by leveraging strengths of learning algorithms to consider all constraints and phase transitions simultaneously in higher fidelity models.

Proximal Policy Optimization (PPO),[10] which is an online policy gradient method for reinforcement learning, may be capable of planning for all mission constraints starting far from the target, thereby eliminating the need for a phased approach and its associated policy switching at phase transitions. PPO has previously been applied to spacecraft relative motion and docking problems,[11–13] and PPO has been shown to be effective for problems with large and complex observation and ac-

tion spaces outside of spacecraft relative motion, too, including an impressive demonstration from OpenAI that uses PPO to play the competitive multiplayer game Dota 2.[14] In this case, PPO handles a nonlinear continuous observation domain, large action domain, long term dependencies, and variable episode time. Therefore, PPO is capable of producing policies in computationally challenging environments without compromising run-time.

In this work, we design a PPO algorithm for a spacecraft rendezvous and docking problem and analyze its performance to plan efficient and constrained trajectories between phases 2 and 3 of an ARPOD mission. We follow the formulation of a spacecraft benchmark problem[15] and specifically consider a chaser spacecraft attempting to rendezvous and dock with a non-actuated target spacecraft. We start the mission close enough to the target such that both angle and range measurements are available but sufficiently far away so that the chaser is not yet considered to be in the docking phase. Moreover, we assume that the full state can be estimated from these measurements, and we use full state information to design the learning algorithms that calculate trajectories (including position, velocity, and actuation) to complete the rendezvous and docking mission. To make the model aware of mission constraints, such as maximum thrust and staying within a line-of-sight (LoS) region, we formulate the model feedback through a predefined reward function.

To avoid problems associated with transitioning between mission phases, we take advantage of a learning-based approach to plan longer trajectories than span both phases 2 and 3 of an ARPOD mission. We modify the nonlinear LoS cone from the benchmark problem[15] by extending from 100 meters to 800 meters from the docking port with an opening angle of 60 degrees so that the trained model will learn to start alignment with the docking port much sooner, resulting in smoother trajectories and higher likelihood of successful docking without the challenges of ensuring a feasible trajectory upon entering the docking phase.

Our approach is largely motivated by (Ref. 12), where a PPO algorithm for a 3-DoF chaser spacecraft performing rendezvous is proposed. We extend this work to consider phases 2 and 3 of the full mission starting from farther away from the target, a reward formulation that considers fuel consumption for more fuel-efficient trajectories, and an LoS constraint. Other differences include continuous rather than impulsive thrust, a chaser spacecraft with larger mass and different thrust constraints, and differences in PPO implementation details. We evaluate the resulting trained model by performing 100 Monte Carlo Simulations with significant random variations in initial positions and velocities and record mission completion time, fuel consumed, terminal states, and computation time per step in the model.

PROBLEM FORMULATION

We consider a passive (non-actuated) target spacecraft that is orbiting in a circular Keplerian orbit. Then the equations for relative motion between an active chaser spacecraft and the passive target are[16]

$$\begin{aligned} \ddot{x} - 2n\dot{y} - n^2(R+x) + \mu\frac{R+x}{((R+x)^2+y^2+z^2)^{\frac{3}{2}}} &= u_x \\ \ddot{y} + 2n\dot{x} - n^2 y + \mu\frac{y}{((R+x)^2+y^2+z^2)^{\frac{3}{2}}} &= u_y \\ \ddot{z} + \mu\frac{z}{((R+x)^2+y^2+z^2)^{\frac{3}{2}}} &= u_z. \end{aligned} \tag{1}$$

These nonlinear translational spacecraft relative motion dynamics describe the chaser spacecraft's position and velocity in a local-vertical/local-horizontal (LVLH) frame with reference fixed on the center of gravity of the target spacecraft. The states $\mathbf{x} := [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}]^\top \in \mathbb{R}^6$ include the spacecraft's position and velocity. The state x denotes the position in the radial direction (from earth), also known as the cross-track direction, y denotes the position in the in-track direction, and z denotes the position in the cross-track direction that completes the right-hand coordinate system with x and y. We consider six continuously variable thrusters that provide thrust in the positive and negative direction of each dimension and define the thrust actuation control inputs as $\mathbf{u} := [u_x\ u_y\ u_z]^\top \in \mathbb{R}^3$. R denotes the orbit radius of the target spacecraft, $n = \sqrt{\mu/R^3}$ is the angular speed, or "mean motion," of the target through its orbit, and μ is the Earth's gravitational constant.

When the chaser is close to the target, the nonlinear spacecraft relative motion dynamics (1) can be linearized around the target's position, which gives the following Hill-Clohessy-Wiltshire (HCW) equations[17, 18]

$$\ddot{x} - 3n^2x - 2n\dot{y} = u_x, \qquad \ddot{y} + 2n\dot{x} = u_y, \qquad \ddot{z} + n^2z = u_z. \tag{2}$$

We assume that the chaser spacecraft has six thrusters capable of continuously variable thrust, so the three control inputs are continuously variable up to a maximum thrust of $\bar{u}$. Thus, u_x, u_y, and u_z can all take values within the set $[-\bar{u}\ \ \bar{u}]$. We assume that the full state $\mathbf{x}$ is available for training and testing. In the examples to follow, we use the linear dynamics (2) to design the learning algorithms and simulate those same dynamics when testing the learned model.

Therefore, for the rendezvous and docking mission, the problem is to find control inputs $\mathbf{u}$ that drive the state of the chaser spacecraft $\mathbf{x}$ to zero (i.e., the origin of the LVLH reference frame centered at the target's position) given full state feedback. In the following, we design and implement a PPO algorithm to solve this problem.

PROXIMAL POLICY OPTIMIZATION

PPO is a policy gradient method of reinforcement learning that uses a neural network π_θ, to parameterize an agent's policy with weighting parameters θ. To do so, the network iteratively optimizes weights through gradient descent based on a reward function evaluated during repeated simulations of an environment. This reward is maximized to achieve the ultimate goal of minimizing an objective loss function $L(\theta)$. The resulting policy π_θ is a neural network that maps input variables to mean μ and log standard deviations σ of each action to form independent normal distributions representing the action. These distributions are sampled to determine each corresponding control action (i.e., thrust).

Training using PPO consists of two parts: a roll-out phase and an update phase. In the roll-out phase, the agent uses a randomly initialized π_θ to navigate the space using equations (2). Every step the agent returns a control input to the environment from which futures states (observations) are calculated using the dynamics (2) and fed back to the agent. The agent uses the data it collects every T intervals to optimize its parameters further. These data include the actions from the agent (control inputs), the agent's perceived log probability of taking that action, the reward feedback, and the new state resulting from the action. These metrics are used to compute the optimal update to the network parameters θ.

Actor Critic Method

Our PPO formulation consists of two distinct neural networks that do not share parameters: the actor and the critic. The actor network learns to parameterize the control policy within its network architecture. Specifically, the actor follows a policy $\pi_{\theta_p}(s_t)$ that takes the current state observation s_t as input and returns a mean value for each element in the action space. Additionally, the actor stores parameters independent of the feed-forward network, which predict the log standard deviations of each action component. With both the standard deviation and mean of an action component, PPO models the continuous action space as a set of independent Gaussian distributions that better characterize non-discrete decisions. The approximate standard deviation is derived by exponentiation of the log standard deviations. Each distribution is sampled to obtain an action a_t (i.e., control input), at each time step t. Compared to most other machine learning algorithms, PPO's unique Gaussian action representation gives an idea of how confident the model is in its actuation at a given state. As the standard deviation converges to a minima, we see less exploration and more exploitation of learned experiences. This intuition of the network's confidence is then applied to calculate the advantage, a scalar quantifying the quality between two decisions.

The other network (the critic network) also receives the same input, however the network learns to output a single scalar value $V_{\theta_c}(s_t)$ that predicts the expected cumulative reward starting from s_t for the episode under policy $\pi_{\theta_p}(s_t)$.

At every step we sample and record the observation s_t, the action taken a_t, the reward r_t, the value prediction $V_{\theta_c}(s_t)$, and the log probability $\log(a_t|s_t)$ in order to carry out policy updates at the end of a roll-out (i.e., a collection of step samples). The policy $\pi_{\theta_p}(s_t)$ makes T iterations to accumulate T step samples of s_t, a_t, r_t, $V_{\theta_c}(s_t)$, and $\log(a_t|s_t)$. The sampled data are used to calculate the advantage A_{GAE}, the policy ratio $\boldsymbol{\rho}$, and the entropy H over the set of possible actions.

We apply value bootstrapping to episodes that end before the time limit of the environment. To achieve this we denote ***Termination*** as a boolean variable where ***Termination*** $= 1$ when an episode ends as a result of reaching the time limit and ***Termination*** $= 0$ otherwise. Then, the critic network evaluates a policy using the Generalized Advantage Estimate A_{GAE} given as

$$A_{\text{GAE}} = \sum_{k=t}^{T-1} (\gamma\lambda \times \textbf{\textit{Termination}})^{k-t} \delta_k,$$

where

$$\delta_t = r_t + \gamma V_{\theta_c}(s_{t+1}) \times (1 - \textbf{\textit{Termination}}) - V_{\theta_c}(s_t).$$

Here, δ_t is the "TD-error", which represents the temporal difference between the true reward r_t and the critic network's estimate of r_t. Since $V_{\theta_c}(s_t)$ is the predicted sum of future rewards starting from t for the episode, the critic network's estimated reward at step t is calculated as the difference $V_{\theta_c}(s_{t+1}) - V_{\theta_c}(s_t)$. To include consistent discounting, the full TD-error is $\gamma V_{\theta_c}(s_{t+1}) - V_{\theta_c}(s_t)$, where γ is the discount factor. Another Variable $\lambda \in [0\ 1]$ balances bias and variance in advantage estimates. This is a variation of the typical advantage formulation seen in older actor critic based models such as TRPO.[19] Given the Generalized Advantage Estimate, the returns are calculated at each step t as

$$G_t = A_{\text{GAE}} + V_{\theta_c}(s_t). \tag{3}$$

Roll-out samples paired with respective advantage values at index t are then shuffled and split into subsets of the roll-out. The subsets, commonly referred to as mini-batches or batches, will determine values for the policy ratio $\boldsymbol{\rho}$, and action entropy H. The values are used in the actor loss objective to apply updates to parameters θ_p and θ_c iteratively per batch for all batches.

PPO evaluates the policy deviations between new and old parameters with the policy ratio

$$\boldsymbol{\rho}(\pi_{\theta_p^{\text{new}}}, \pi_{\theta_p^{\text{old}}}) = \sum_{t=0}^{T} \frac{\pi_{\theta_p^{\text{new}}}(a_\text{t}|s_\text{t})}{\pi_{\theta_p}(a_\text{t}|s_\text{t})}.$$

Lastly PPO includes the action entropy $H(\pi_{\theta_p}(a_\text{t}|s_\text{t}))$ into the loss formulation to encourage exploration. The entropy per batch is formulated as

$$H(\pi_{\theta_p}) = \sum_{t=0}^{T} 0.5 + 0.5\log(2\pi) + \log(\sigma(\pi_{\theta_p}(a_\text{t}|s_\text{t}))).$$

The parameters θ_p of the policy network are adjusted for policy performance by optimizing the loss

$$L = \mathrm{E}[A_{\text{GAE}} \times \boldsymbol{\rho}(\pi_{\theta_p^{\text{new}}}, \pi_{\theta_p^{\text{old}}})].$$

The original gradient loss is seen in many policy gradient methods, in particular TRPO. This vanilla loss often results in overconfident updates, resulting in training instability. In PPO this loss is clipped to prevent this instability in training as

$$L_c = \mathrm{E}[\min\left(L, \text{clip}(L, 1-\epsilon, 1+\epsilon) + (c_e H(\pi_{\theta_p}))\right)].$$

The parameters θ_c of the value function network are adjusted for improved value function estimation by optimizing the loss

$$L_V = c_v \sum_{t}^{T} (V_{\theta_c}(s_\text{t}) - G_\text{t})^2.$$

The old parameters are cached and both new and old sets of parameters are used for update iterations in the next batch.

This entire method is outlined in Algorithm 1.

Environment and Rewards

In the environment, the agent's next observation is calculated using the dynamics (2) and the current control action a_t and current observation s_t. Every new observation is used to give reward feedback according to a reward function. The agent learns to map state observations to control inputs that transition to states that maximize the reward. The force F_t that each thruster applies is bounded within $\left[-\bar{F},\ \bar{F}\right]$, and the mass of the chaser m is assumed to stay constant. Thus, the control input $\mathbf{u}$ is constrained such that $u_t = F_t/m \in \left[-\bar{F}/m,\ \bar{F}/m\right]$.

Because we want the chaser spacecraft to stay within an LoS region, we build that constraint into the reward function. We denote $\boldsymbol{\rho}_t$ as the position vector of the chaser relative to the target at time

Algorithm 1: Rollout Update in PPO with GAE and Early Truncation

Input : Policy parameters θ_p, Value function parameters θ_c
for *each training iteration* **do**
  Collect state-action pairs (s_t, a_t) for T time steps;
  Calculate rewards r_t and values $V_{\theta_c}(s_t)$;
  Calculate advantages A_{GAE};
  Normalize advantages A_{GAE};
  Calculate returns as formulated in 2;
  Shuffle state-action pairs, rewards, values, and advantages;
  Divide shuffled data into mini-batches;
  for *each mini-batch* **do**
    Update policy network π_{θ_p};
    if $\theta_p^{new} = None$ **then**
      $\theta_p \rightarrow \theta_p^{\text{new}}$
    end
    for *each state-action pair* (s_t, a_t) **do**
      Calculate old policy probability $\pi_{\theta_p}(a_t|s_t)$;
      Calculate new policy probability $\pi_{\theta_p^{\text{new}}}(a_t|s_t)$;
      Calculate surrogate loss $L_c(\theta_p)$;
      Back Prop and Update new $\rightarrow \theta_p^{\text{new}}$;
    end
  end
  Update value function network V_{θ_c};
end

t, $\boldsymbol{\beta}$ as a vector that points 800 meters out from the target's docking port in the y direction, and ϕ denotes the angle of the LoS cone. Then the LoS region is given as

$$LoS = \left[\frac{\boldsymbol{p}_t \cdot \boldsymbol{\beta}}{\|\boldsymbol{p}_t\| \|\boldsymbol{\beta}\|} \geq \cos\left(\frac{\phi}{2}\right) \right] \wedge [y \geq y_d],$$

where y_d is the y-position of the docking port. This LoS region is visualized in Figure 2.

Given the target/docking state $\mathbf{x}_d$, we define $\boldsymbol{d}_t = \mathbf{x}_d - \mathbf{x}_t$ as the element wise difference between the docking state and the current state. The reward generally captures the loss and is a function of the chaser spacecraft's state relative to the target/docking state and the LoS region. Thus, we define the reward as

$$r_t = \begin{cases} \mathbf{d}_t^\top Q_x \mathbf{d}_t + \mathbf{u}_t^\top Q_u \mathbf{u}_t - (\rho_1 - \|\mathbf{d}_t\|_2)^2 & \text{if not LoS and } \|\boldsymbol{d}_t\|_2 > \rho_1 \\ \mathbf{d}_t^\top Q_x \mathbf{d}_t + \mathbf{u}_t^\top Q_u \mathbf{u}_t - (\rho_2 - \|\mathbf{d}_t\|_2)^2 & \text{if not LoS and } \|\boldsymbol{d}_t\|_2 < \rho_2 \\ \mathbf{d}_t^\top Q_x \mathbf{d}_t + \mathbf{u}_t^\top Q_u \mathbf{u}_t + (\rho_2 - \|\mathbf{d}_t\|_2)^2 & \text{if LoS and } \|\boldsymbol{d}_t\|_2 < \rho_2 \\ \mathbf{d}_t^\top Q_x \mathbf{d}_t + \mathbf{u}_t^\top Q_u \mathbf{u}_t + (\rho_2 - \|\mathbf{d}_t\|_2)^2 + (20\rho_3 - \|\mathbf{d}\|_2)^2 & \text{if LoS and } \|\boldsymbol{d}_t\|_2 < \rho_3 \ , \end{cases}$$

where Q_x and Q_u are diagonal weighting matrices and ρ_1, ρ_2, and ρ_3 are scalars that define the conditions for changing the weights of the reward function.

The environment truncates terminal or win conditions during training so as to avoid the agent developing a policy that tries to terminate the mission early to avoid negative penalty.

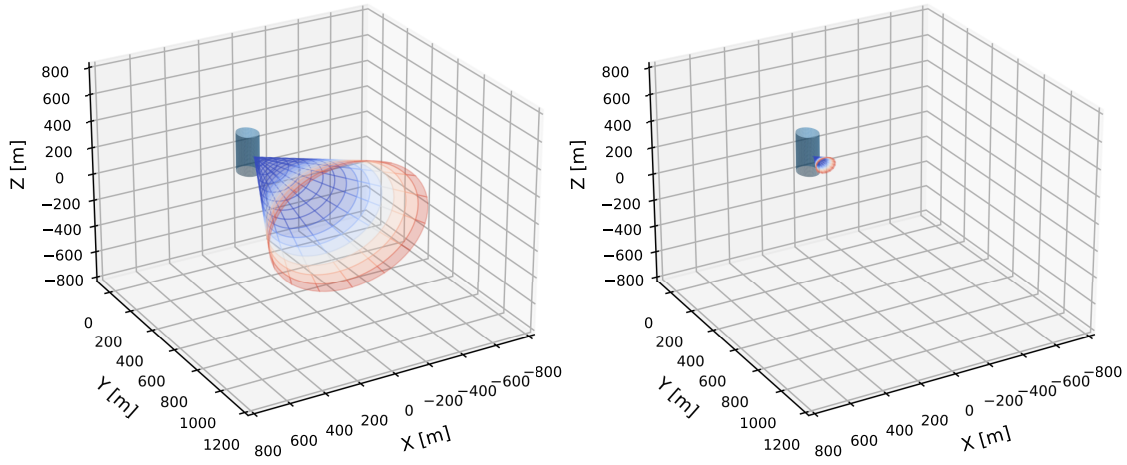


Figure 2: Visualization of target spacecraft and LoS region. Left: Proposed LoS that extends 800 meters in the y direction. Right: Standard benchmark LoS that extends 100 meters in the y direction.

TRAINING PROCEDURE

Algorithm 1 describes how training is executed. We heavily modified the PyTorch PPO implementation from Stable-Baselines-3 for our environment. We ran multiple environment instances with the vectored environment wrapper from Stable-Baselines-3. To handle the difference between termination and episode truncation, we applied an additional time limit wrapper over our environment to modify the done signal as truncated if the environment had not reached its step limit. The machine used to train the model had 16GB of system memory, AMD Ryzen 9 3900XT 12-Core Processor CPU build on x86 architecture, and a single 24GB RTX 3090-TI NVIDIA GPU with CUDA 12 and VULKAN 1.2 compute capabilities. All environment dynamics were calculated using the CPU on system memory, and all other calculations were done with the GPU on GPU memory.

Parallel Environments and Time Limit Handling

Since our observation and action spaces are both continuous, a single agent in this environment will never be able to gain a representative sample size at each policy update. To improve on this, we use multiple agents that navigate through the environment in parallel on the policy to collect 12 sets of T roll-outs. The discount and advantage calculations remain environment specific, however afterward the roll-outs are all merged into a single roll-out. The collection of all samples in the merged roll-out is treated as a single roll-out, which is shuffled and split into subsets of B samples, or batches, to then compute policy updates.

In the initial stages of training, a smaller maximum environment time T_e is used. The maximum environment time T_e is incrementally increased as the variance in initial states is increased. After, the environment does not reset; instead, it continues from where it ended and the process is repeated until $T = 500$ steps are taken in the environment. Then the returns and other updates are calculated. In summary, we have $N_e = 30$ environments that are each sampled for $T = 500$ steps, so $T_r = N_e \times T = 15,000$ total steps (corresponding to more than 4 hours of mission time) per roll-out.

We also make a distinction between terminal and truncated episode reset conditions. In the environment, we truncated the docked state and the target collision condition. The differences in the policy update is detailed in Algorithm 2.

Algorithm 2: Truncation vs Termination Bootstrap in PPO

for *each state* s_t **do**
  if *episode is truncated* **then**
    Calculate returns G_t as in (3);
  else
    Calculate returns as $G_t = \delta_t + \hat{V}(s_t)$;
  end
  Backpropagate and update parameters $\rightarrow \theta_c$;
end

Network Architecture

The actor and critic networks are completely separate and do not share parameters. Both networks are orthogonally initialized layers with a default standard deviation of $\sigma = \sqrt{2}$ for each layer. Between each layer we use GeLU activation functions.[20] The actor network has four layers with widths of 130, 44, 30, and 3, respectively. The orthogonal layer that outputs the action means is initialized with a standard deviation of $\sigma = 0.01$. The critic network has the same number of layers as the actor network but has a much wider profile. The width of each layer is 145, 25, 5, and 1, respectively, where the last layer is the single scalar value estimate. This is described in Table 1.

Table 1: Actor Critic Model Architectures

	Actor Architecture		**Critic Architecture**	
Layer	**Neurons (Width)**	**# of Parameters**	**Neurons (Width)**	**# of Parameters**
Input	(130)	910	(145)	1015
Hidden Layer 1	(44) Activation: GeLU	5764	(25) Activation: GeLU	3650
Hidden Layer 2	(30) Activation: GeLU	1350	(5) Activation: GeLU	130
Output Layer	(3)	93	(1)	6
Total Trainable Params	-	12918	-	12918

Normalization

The shape of the observation, scale of observation values, episode duration, and scale of rewards all affect the network output and loss. Input normalization is a common pre-processing step in other machine learning applications that can improve training efficiency.[21] In our environment, the possibly large differences in scaling of rewards and observations at different stages of the mission made normalization important.

We define o_t as the current observation, and $\mathcal{O}$ as the set of all observations seen in training. The mean μ and standard deviation σ are running calculations based on the training duration. The

normalized observation $\hat{o}_t$ is calculated as

$$\hat{o}_t = \frac{o_t - \mu(\mathcal{O})}{\sigma^2(\mathcal{O})} \in [-\bar{o}, \bar{o}]$$

and is clipped such that $\bar{o}$ denotes the maximum value observation o_t can take.

The advantage A_{GAE} is normalized per batch, such that the normalized advantage $\hat{A}_{\text{B}}$ is given as

$$\hat{A}_{\text{B}} = \frac{A_{\text{GAE}} - \mu(\mathcal{A})}{\sigma(\mathcal{A})},$$

where $\mathcal{A}$ is the set of all advantages from previous sampling windows in the batch. We calculate the normalized returns over every sampling window as

$$\hat{G}_T = \frac{\sum_{t=0}^{T} G_t - \mu(\mathcal{G})}{\sigma^2(\mathcal{G})},$$

where $\mathcal{G}$ is the set of all returns from previous sampling windows.

SIMULATION RESULTS

In this section we present results from training a PPO algorithm for the spacecraft autonomous rendezvous and docking problem, as described above*. The hyper parameter values chosen are given in Table 2, system parameter values are given in Table 3, and results from 100 different runs using the trained model with randomly varying initial positions and velocities are shown in Figures 3– 5 and Tables 4–5.

Table 2: Hyper parameters.

Parameter	Description	Value
T	Sampling Window	500
N_e	Parallel Environments	30
T_r	Time Steps per Roll-out	$N_e \times T = 15000$
B	Batch Size	1000
T_e	Max Environment Time	1500
η_a	Actor Learning rate	0.0003
η_c	Critic Learning rate	0.0005
γ	Discount factor	0.998
λ	GAE lambda	0.98
N_R	Recycles per Roll-out	20
ϵ	Clip range	0.20
c_e	Entropy coefficient	0.0002
c_v	Value coefficient	0.5
∇	Max Normal Gradient	0.002

*Code is available as a pip package here: https://github.com/Vince-C156/PLANNING-AUTONOMOUS-SPACECRAFT-RENDEZVOUS-AND-DOCKING-TRAJECTORIES-VIA-REINFORCEMENT-LEARNING

Table 3: System, environment, and reward parameters.

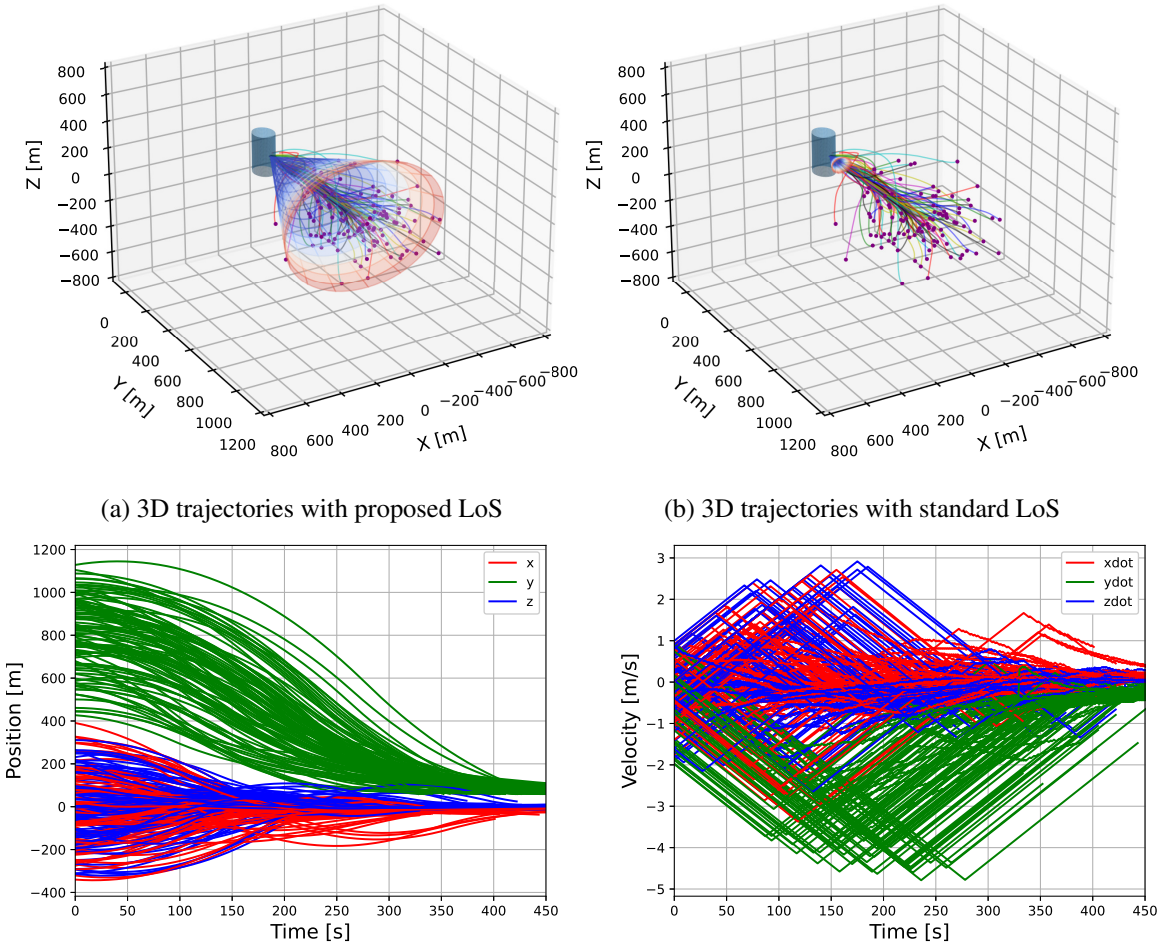
Parameter	Description	Value	Units
$\bar{F}$	Max thrust force	10	N
m	Mass of chaser spacecraft	500	kg
$\bar{u}$	Max control input, i.e., $\bar{F}/m$	0.02	m/s^2
T	Environment Time Limit	1500	s
μ	Earth's gravitational constant	3.986×10^{14}	m^3/s^2
r	Earth's radius	6,178,317	m
R	Geostationary orbit semi-major axis	$35.786 \times 10^6 + r$	m
h	Sampling time step	1	s
ϕ	Angle of cone defining LoS region	60	deg
ρ_1	Reward function condition 1	800	m
ρ_2	Reward function condition 2	350	m
ρ_3	Reward function condition 3	5	m
$\bar{o}$	Maximum normalized observation value	100	-
Q_x	Reward weighting matrix	diag$\{.35\ .15\ .35\ 0.015\ 0.35\ 0.015\}$	-
Q_u	Reward weighting matrix	diag$\{10.5\ 10.5\ 10.5\}$	-

For these results, we progressively trained an agent with initial positions uniformly sampled from $x \in [-100,\ 100]$ m, $y \in [400,\ 600]$ m, and $z \in [-5,\ 5]$ m. We took the weights from each set of training and incrementally increased the agent's initial positions to be uniformly sampled from $x \in [-400,\ 400]$ m, $y \in [400,\ 1200]$ m, and $z \in [-400,\ 400]$ m. For all training rounds the initial velocities $(\dot{x}\ \dot{y}\ \dot{z})$ were each uniformly sampled from $[-2,\ 1]$ m/s. The docking point is at the position $(x_d,\ y_d,\ z_d) = (0,\ 60,\ 0)$ m, which is the target's location with an extension in the y direction. Then the target/docking state is given as $\mathbf{x}_d = [0\ 60\ 0\ 0\ 0\ 0]^\top$.

Figure 3 shows 100 trajectories produced by the trained model with random initial positions that begin up to 1200 meters from the target. The majority of these trajectories result in the chaser spacecraft successfully achieving rendezvous with the target while staying within the LoS region. Once within a few meters of the docking port, additional sensing may be available, and a local controller could be used to successfully complete the docking maneuver. Figure 4a shows the distribution of initial positions varying from $x \in [-400, 400]$, $y \in [400, 1200]$ and $z \in [-400, 400]$. Figure 4b shows that the terminal positions are within a few meters of the docking port.

Figure 5 shows a single trajectory produced by the trained model. Figures 5a and 5b show that the trajectory stays within the proposed extended LoS and the standard benchmark LoS, respectively. It is apparent that the trained model considered the LoS region earlier than if the standard LoS were used, which enabled a smooth trajectory as the chaser approached the target. Figure 5c further shows this, as the x and z positions begin to converge when the agent is approximately 400 meters from the target in the y direction. Figure 5d shows that the trained model is careful to both stay within the LoS region and to not collide with the target since $\dot{y}$ reaches zero around 310 seconds and then is increased slightly only once $\dot{x}$ and $\dot{z}$ (and the corresponding positions) converge to zero.

Statistics for the terminal states are given in Table 4 and generally show that the trained model achieves rendezvous by producing trajectories with states that terminate close to the docking state. These results show a bias in the terminal x positions in the negative direction, with a mean of -6.48 meters and large standard deviation of 13.5 meters even though the median is 0.601 meters.



(a) 3D trajectories with proposed LoS

(b) 3D trajectories with standard LoS

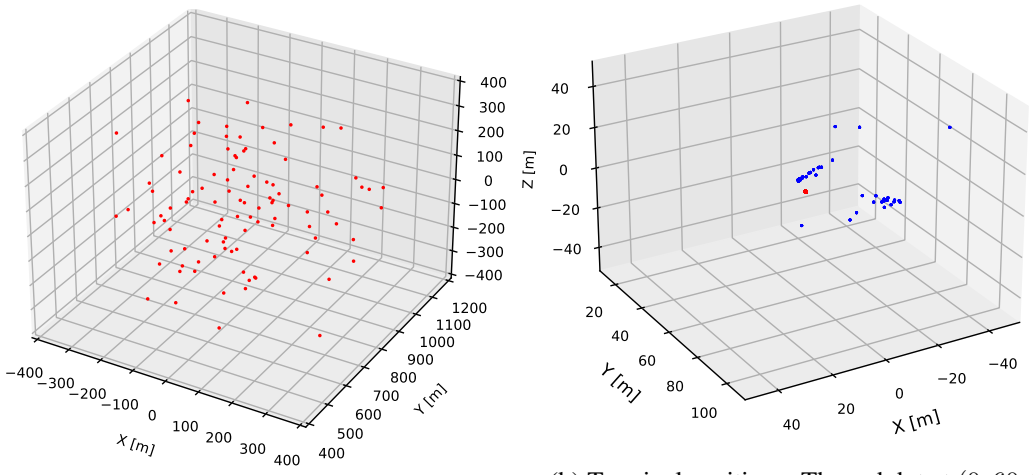
(c) Position states for 100 trajectories

(d) Velocity states for 100 trajectories

Figure 3: 100 rendezvous maneuver trajectories produced by the trained model with random initial positions and velocities

This may be due to a large fraction of the 3D trajectories shown in Figure 3a, with initial positions shown in Figure 4a, starting both close to the target and with significant initial negative x position. Figure 4b also shows this with two main clusters for the terminal positions: one with $-x$ deviation and one without. This may imply that the cause of the negative x bias is due to a particular set of conditions rather than being intrinsic to the policy. Table 4 also shows that the terminal $\dot{x}$ and $\dot{z}$ averages of 0.0264 and 0.0142 m/s, respectively, are considerably lower than the average terminal $\dot{y}$ at -0.39 m/s. This shows the model's tendency to prioritize aligning the chaser spacecraft within the center of the LoS region before finally approaching and docking with the target in the y direction, thereby planning both phases 2 and 3 of the mission as a single trajectory.

Table 5 shows episode and environment statistics, including episode length (i.e., mission time), total fuel consumed, and computation time per step. Of the 100 mission times, 80 are between 386.9 and 576.0 seconds. These mission times are possible due to significant fuel use with continuous



(a) Initial positions

(b) Terminal positions. The red dot at $(0, 60, 0)$ meters is the docking port.

Figure 4: Initial and terminal states for the 100 rendezvous maneuver trajectories shown in Figure 3

Table 4: Terminal state statistics for the 100 trajectories shown in Figure 3

Variable	Mean	Standard Deviation	Median	Units
x	-6.48	13.5	0.601	m
y	63.9	3.54	66.21	m
z	1.62	8.93	4.56	m
$\dot{x}$	0.0264	0.184	0.0372	m/s
$\dot{y}$	-0.390	0.473	-0.188	m/s
$\dot{z}$	0.0142	0.140	0.0254	m/s

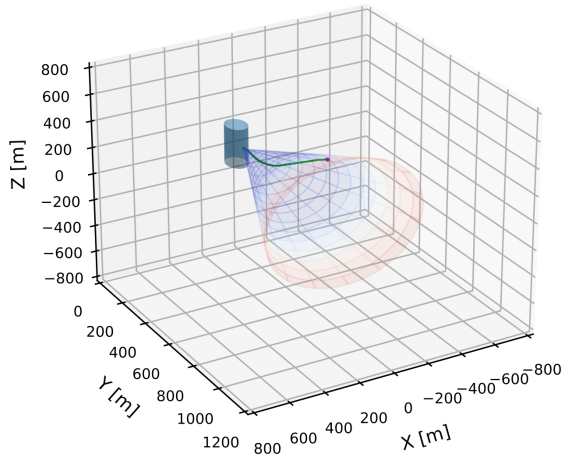
thrust. The computation time is constant over all time steps.

Table 5: Episode/mission and environment statistics for the 100 trajectories shown in Figure 3

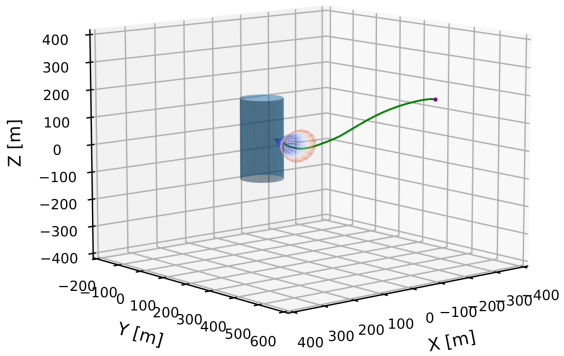
Data	Mean	Standard Deviation	Median	Units
Episode Length (mission time)	481.4	73.78	505.5	s
Episode Fuel	16.64	2.56	17.48	m/s^2
Agent Step Run-Time (CPU)	373.5	0	373.5	ms

DISCUSSION ON IMPLEMENTATION

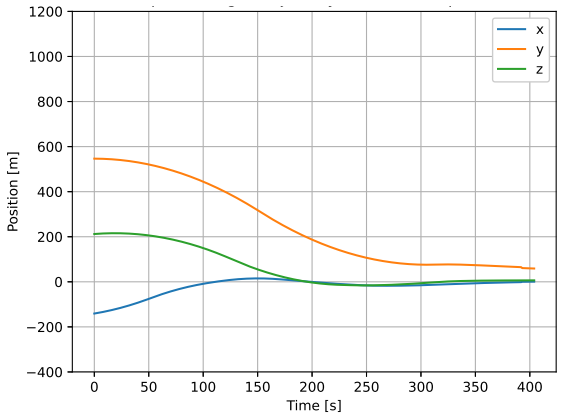
Normalization and network parameter initialization were essential to the training process. Without imitation learning, we were able to achieve our results by iteratively training the policy. Attempts to train the policy for long episodes and significant distances from the target yielded no success. We instead trained an initial policy on closer and shorter missions. The initial parameters converged to reasonable behavior, and we took the same weights from the policy and value networks and retrained



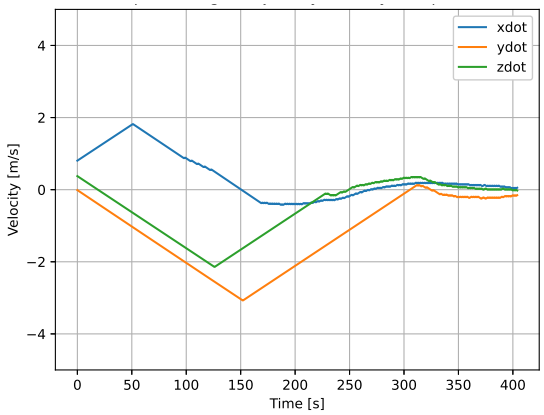
(a) Single trajectory with proposed LoS



(b) Single trajectory with standard LoS



(c) Position states for single trajectory



(d) Velocity states for single trajectory

Figure 5: Single rendezvous maneuver trajectory produced by the trained model

them on initial conditions with larger variance. We were also able to modify the reward function and parameters between training sections as long as the changes were small in magnitude. Major modifications to the reward or discount would make it difficult for the value function to converge.

Throughout the training process, we found that several parameters and implementation details were especially important. Initial results showed inconsistencies in the PPO loss and ability for the reward to converge. We originally chose the PPO algorithm to minimize the training inconsistencies commonly seen with other policy gradient methods, however, tuning the PPO algorithm was challenging.

- Tuning the sampling window T along with the number of parallel environments N_e was important. Due to the discounted reward formulation, the number of time steps per roll-out T_r cannot be too large because reward feedback near the end of the roll-out would not be accounted for.

- Through experimentation, we found that the discount factor γ had to be set relatively high (e.g., $\gamma = 0.998$) to weigh rewards throughout the entire sampling window.
- Tuning the Generalized State Advantage A_{GAE} (i.e., tuning the value of λ) was important to decrease overall variance in the feedback. First, we applied a high discount factor γ without using a Generalized State Advantage. This resulted in instabilities in training, as too high of a discount factor may cause the agent to over-fit to state trajectories that are unlikely to occur.
- Heavy normalization was important. Since our mission space is continuous in the observation and action spaces, heavy normalization is needed to handle the magnitude of the inputs before the forward pass of the neural network.
- Lastly, use of N_e parallel unique environments for each roll-out update was important. In order to achieve a representative sample size of the mission before updating the policy, multiple parallel agents were needed to sample the space.

CONCLUSION AND FUTURE WORK

We developed a Proximal Policy Optimization (PPO) method of Reinforcement Learning to compute safe and efficient trajectories for a 3-DoF chaser spacecraft performing an autonomous rendezvous and docking mission with an un-actuated target spacecraft on orbit. The resulting trained model produces successful rendezvous trajectories starting from up to 1200 meters away from the target while keeping the chaser spacecraft within an extended line-of-sight (LoS) region relative to the target. This extended LoS region proved to be an effective formulation to nearly guarantee that the chaser spacecraft satisfies the nonlinear LoS constraint as it approaches the target spacecraft by incentivizing early alignment of the chaser with the target in the x and z directions. Additionally, the trained model demonstrates robustness by generating successful rendezvous trajectories given large variations in initial positions (up to a variation of 800 meters in each direction) and velocities (up to a variation of 3 meters per second in each direction). The results from 100 trajectories produced by the trained model show convergence of the chaser's position and velocity to close to the target's state. Moreover, the average mission time, fuel consumed, and computation time were reported. Importantly, these results show the viability of using a reinforcement learning-based control policy for this problem with nonlinear constraints and discontinuities in phase transitions while maintaining reasonable computation time.

There are numerous directions for future work. Phase 1 of an Autonomous Rendezvous, Proximity Operations, and Docking (ARPOD) mission could be included, where the full state is only partially observable. Sensor and actuation errors can be incorporated to test the method's ability to develop models that are robust to these types of errors. Modifying PPO with Long Short Term Memory (LSTM) networks has been demonstrated to converge on optimal Partially Observable Markov Decision Policies, and future work in this direction would be interesting. Additionally, imitation learning can be applied on real world data from past flights or simulated controller performance to initialize the policy before applying training. This would alleviate problems commonly associated with model free reinforcement learning such as premature convergence and unstable training by initializing the policy closer to the global minimum. Another opportunity for future work is to re-design the reward formulation. Our rewards were dense with feedback at every step. A sparser reward formulated at every sub-goal of the mission may be more computationally efficient and result in more flexibility in initialization of the agent. Lastly, obstacle avoidance implementations may be considered since PPO has been demonstrated to be effective in complex and dynamic environments.

Finally, we tried to detail our most significant experiences learned in the process of training a PPO algorithm for this problem. It is difficult to make general rules or conventions for success in reinforcement learning since many aspects must be carefully tailored for specific use-cases. We found the study from Google Brain,[21] that bench-marked several combinations of parameters, environments, and models, to be a helpful resource.

ACKNOWLEDGEMENT

This work was supported by UCI's Undergraduate Research Opportunities Program (UROP) and the Air Force Office of Scientific Research (AFOSR) through the Air Force Research Laboratory (AFRL) Summer Faculty Fellowship Program.

REFERENCES

[1] D. Malyuta, Y. Yu, P. Elango, and B. Açıkmeşe, "Advances in trajectory optimization for space vehicle control," *Annual Reviews in Control*, Vol. 52, 2021, pp. 282–315.

[2] I. Garcia and J. P. How, "Trajectory optimization for satellite reconfiguration maneuvers with position and attitude constraints," *Proceedings of the 2005, American Control Conference, 2005.*, IEEE, 2005, pp. 889–894.

[3] E. N. Hartley, "A tutorial on model predictive control for spacecraft rendezvous," *2015 European Control Conference (ECC)*, IEEE, 2015, pp. 1355–1361.

[4] A. A. Soderlund, S. Phillips, A. Zaman, and C. D. Petersen, "Autonomous Satellite Rendezvous and Proximity Operations via Geometric Control Methods," *AIAA Scitech 2021 Forum*, 2021, p. 0075.

[5] A. Weiss, M. Baldwin, R. S. Erwin, and I. Kolmanovsky, "Model predictive control for spacecraft rendezvous and docking: Strategies for handling constraints and case studies," *IEEE Transactions on Control Systems Technology*, Vol. 23, No. 4, 2015, pp. 1638–1647.

[6] C. Jewison, R. S. Erwin, and A. Saenz-Otero, "Model predictive control with ellipsoid obstacle constraints for spacecraft rendezvous," *IFAC-PapersOnLine*, Vol. 48, No. 9, 2015, pp. 257–262.

[7] M. V. Paris, "Safe ARPOD for under-actuated CubeSat via reinforcement learning," *MA thesis. Politecnico di Milano*, 2021.

[8] K. Hovell and S. Ulrich, "On deep reinforcement learning for spacecraft guidance," *AIAA Scitech 2020 Forum*, 2020, p. 1600.

[9] K. Dunlap, M. Mote, K. Delsing, and K. L. Hobbs, "Run time assured reinforcement learning for safe satellite docking," *Journal of Aerospace Information Systems*, 2022, pp. 1–12.

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

[11] L. Federici, B. Benedikter, and A. Zavoli, "Machine learning techniques for autonomous spacecraft guidance during proximity operations," *AIAA Scitech 2021 Forum*, 2021, p. 0668.

[12] J. Broida and R. Linares, "Spacecraft rendezvous guidance in cluttered environments via reinforcement learning," *29th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Society, 2019, pp. 1–15.

[13] C. E. Oestreich, R. Linares, and R. Gondhalekar, "Autonomous six-degree-of-freedom spacecraft docking with rotating targets via reinforcement learning," *Journal of Aerospace Information Systems*, Vol. 18, No. 7, 2021, pp. 417–428.

[14] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[15] C. Jewison and R. S. Erwin, "A spacecraft benchmark problem for hybrid control and estimation," *2016 IEEE 55th Conference on Decision and Control (CDC)*, IEEE, 2016, pp. 3300–3305.

[16] B. Wie, *Space vehicle dynamics and control*. AIAA, 1998.

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

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

[19] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *CoRR*, Vol. abs/1502.05477, 2015.

[20] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," 2020.

[21] M. Andrychowicz, A. Raichuk, P. Stanczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study," *CoRR*, Vol. abs/2006.05990, 2020.