Use Of Unity Game Engine And Unity ML-Agents And Proximal Policy Optimization With Imitation Learning And Analysis Of Reward Definition To Simulate Satellite Debris Rendezvous

Nathaniel Biddle

University of Cincinnati, College of Engineering and Applied Science, 2901 Woodside Drive, Cincinnati, OH 45219, USA

Low Earth Orbit debris must be captured and taken out of orbit at some point, or there is a risk of an increasing likelihood of collisions and potential for a domino effect, Kessler Syndrome. This paper explored the available functionality within Unity ML-Agents, a software package for the Unity game engine, to train using the Proximal Policy Optimization deep reinforcement learning algorithm. Unity and Unity-ML Agents were used to simulate and train a reinforcement learning policy for rendezvousing with and softly touching a target point on a satellite utilizing state-of-the-art algorithms made available in the package, fine-tuning hyperparameters, network settings, and reward functionality, as well as utilizing imitation learning to enhance results and reduce the number of episodes required to train. The Imitation Learning data generation utilized a standardized human input policy where a metronome and a specific number of clicks of each button being used as the control device for the soft touch was the most effective, each corresponding with a torque application for the next time step on a given joint. The analysis found that for the simulation of a precise satellite rendezvous and soft touch, PPO with BC and GAIL and no extrinsic rewards was a useful methodology, followed by with custom extrinsic rewards that were defined in the exploration, then followed by custom extrinsic rewards with no imitation learning, utilizing the particular extrinsic reward definition as described in this paper. Further exploration into the relationship between imitation learning and extrinsic reward paradigms could elucidate the specifics of how they interact with each other and whether there is an extrinsic reward model that is more fitting to pair with imitation learning than some others might be.

I. Introduction And Literature Review

More than 3 times as many satellites were launched into space between 2020 and 2023 than have been launched into space since the beginning of human history. Some speculate that number may increase to 100,000 or 200,000 during the next decade [1]. As the commercial space industry continues this unprecedented growth and acceleration, we must be good stewards of this new frontier. This means cleaning up after ourselves as we fill Low Earth Orbit (LEO) with tens of thousands of satellites and as the wheels begin to turn of the new space industrial age. If we do not, we risk subjecting ourselves to Kessler Syndrome, a scenario where space debris and/or pollution grows to a significant enough threshold that for each collision that occurs, it increases the likelihood of the next collision, and so on and so forth. [2] This could be catastrophic for LEO from both a safety and business standpoint, but also catastrophic for the environment and the effective functioning of Earth infrastructure. This necessitates solutions for debris cleanup in LEO. Solutions through simulation to enable these cleanup and servicing activities are being explored, such as utilizing creation of ground robotic platforms to allow for hardware-in-the-loop testing of control strategies. [3] Another type of solution for which will be

explored throughout this text, is a deep reinforcement learning control policy for a satellite rendezvous and capture via robotic means.

This paper simulates the dynamics, kinematics, and an AI control solution for the preliminary steps of such a rendezvous utilizing a simulated vehicle with an attached robotic arm, approaching the satellite and reaching a velocity in station with it, maintaining that velocity, and softly touching the target satellite with the end effector of the robotic arm. A deep reinforcement learning (RL) approach is taken, utilizing proximal policy optimization (PPO) with Behavioral Cloning (BC) and Generative Adversarial Imitation Learning (GAIL). The Unity Game Engine is utilized to simulate the environment, dynamics, and as the RL environment.

Unity is a game engine that has available to it functionality making it capable of providing an environment for research-quality physics simulation, with utilization of it as a research tool growing in academia. [4] Unity ML-Agents is a software package built for Unity that provides a python API between Unity and the broader computing environment it resides in, allowing for reinforcement learning using Unity as the simulation environment. It provides several implementations of state-of-the-art (SOTA) algorithms, including Proximal Policy Optimization (PPO), as well as a wrapper for OpenAI's "gym" wrapper, which provides an interface between learning algorithms and simulations. [4] Here a policy for an autonomous agent is learned using PPO, GAIL, and BC, enabling the desired satellite rendezvous and soft touch in a simulated zero gravity space environment.

The way that ML-Agents works is that there is a class interface that needs to be implemented, and the methods defined. The four main methods of importance are OnEpisodeBegin(), CollectObservations(), OnActionReceived(), and for either testing or for imitation learning use (in this case both), the Heuristic() method. The OnEpisodeBegin() method is used for all of your instantiation and reset of environmental elements for the beginning of a new episode. In CollectObservations(), you define what state information is available to the learning agent.

Researchers have done surveys of these algorithms and the ML-Agents Unity package in the past, in the context of, for example, autonomous car training, as a survey of the existing available SOTA algorithms made available by default in the environment, PPO, GAIL, and BC and/or via interfacing through the available OpenAI gym wrapper. [5] This paper focuses on fine-tuning the available SOTA algorithms as a control method for a satellite rendezvous and analyzing the differences in results when training without BC and GAIL, training with them but without extrinsic rewards, in sparse and shaped environments, and with different length and quality of demonstration data. There wasn't any clear existing paper regarding satellite rendezvous and soft touch utilizing Unity ML-Agents as the simulation platform and learning environment.

II. Mathematical Formulation[1]

The entirety of the physics model that was trained on consisted of a dynamics formulation for the vehicle and the target object, as well as a rudimentary collision detection system for the purposes of this exploration of reinforcement learning algorithms. The dynamics formulation utilized the Euler Integration method rather than using the build-in physics systems of Unity (an area for further exploration at a later point). The dynamics are described below as follows and implemented in the code files attached to this submission:

$$[\mathbf{M}(\mathbf{z})\ddot{\mathbf{z}}+\mathbf{N}(\mathbf{z},\dot{\mathbf{z}})\dot{\mathbf{z}}=\boldsymbol{\tau}]$$

where:

$$\mathbf{z}=\begin{bmatrix} x_v \\ y_v \\ \alpha_v \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$\mathbf{M}(\mathbf{z})=\begin{bmatrix} m_v & 0 & 0 & m_{\text{joint}_1}a_1\cos(\theta_1) & m_{\text{joint}_2}a_2\cos(\theta_2) & m_{\text{joint}_3}a_3\cos(\theta_3) \\ 0 & m_v & 0 & m_{\text{joint}_1}a_1\sin(\theta_1) & m_{\text{joint}_2}a_2\sin(\theta_2) & m_{\text{joint}_3}a_3\sin(\theta_3) \\ 0 & 0 & I_v & 0 & 0 & 0 \\ m_{\text{joint}_1}a_1\cos(\theta_1) & m_{\text{joint}_1}a_1\sin(\theta_1) & 0 & m_{\text{joint}_1} & 0 & 0 \\ m_{\text{joint}_2}a_2\cos(\theta_2) & m_{\text{joint}_2}a_2\sin(\theta_2) & 0 & 0 & m_{\text{joint}_2} & 0 \\ m_{\text{joint}_3}a_3\cos(\theta_3) & m_{\text{joint}_3}a_3\sin(\theta_3) & 0 & 0 & 0 & m_{\text{joint}_3} \end{bmatrix}$$

$$\mathbf{N}(\mathbf{z},\dot{\mathbf{z}})=\begin{bmatrix} 0 & -m_v\dot{\alpha}_v & 0 & 0 & 0 & 0 \\ m_v\dot{\alpha}_v & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -m_{\text{joint}_2}a_2\sin(\theta_2)\dot{\theta}_2 & -m_{\text{joint}_3}a_3\sin(\theta_3)\dot{\theta}_3 \\ 0 & 0 & 0 & m_{\text{joint}_2}a_2\sin(\theta_2)\dot{\theta}_2 & 0 & -m_{\text{joint}_3}a_3\sin(\theta_3)\dot{\theta}_3 \\ 0 & 0 & 0 & m_{\text{joint}_3}a_3\sin(\theta_3)\dot{\theta}_3 & m_{\text{joint}_3}a_3\sin(\theta_3)\dot{\theta}_3 & 0 \end{bmatrix}$$

$$\ddot{\mathbf{z}}=\mathbf{M}^{-1}(\mathbf{z})(\boldsymbol{\tau}-\mathbf{N}(\mathbf{z},\dot{\mathbf{z}})\dot{\mathbf{z}})$$

The dynamics of the target object are described simply by:

$$\mathbf{p}_o(t)=\mathbf{p}_o(0)+\mathbf{v}_o t$$

III. Control Methodology

Proximal Policy Optimization originated from OpenAI in 2017 as a new type of policy gradient method for RL. [6] Unity makes PPO with various configuration options available as one of the out-of-the-box learning algorithms. Through the project, I carried out reward definition, shaping, and tried many different methods to get an effective result, getting some good results, but often finding that it wouldn't reach the degree of efficacy that I would prefer and that would meet the requirements of the project. Having experimented with many options, imitation learning seemed to come out on top, with the hyperparameters and other configurations

[1] *(Formulas generated via LaTex Interpreter – code can be found in the DynamicsKinematics.cs file in the attached code in the deliverable)*

that will be described below. With the imitation learning methods used, Behavioral Cloning (BC) and GAIL (Generative Adversarial Imitation Learning), the agent would pre-train on my demonstration data, giving rewards based on how similar the movements are in regards to BC and in a blinded way for similarity in GAIL. The way GAIL works is that for a given action, it has to guess if the action was the demonstrator or if the action was the agent. If the action is the agent, it gets rewards for tricking GAIL into thinking it was the demonstrator, as that displays a convergence on the demonstration data.

Many reward methods and hyperparameter configurations for the PPO algorithm were attempted to come to a successful solution; some methodologies attempted included:

- Sparse rewards with no imitation learning
- Various precisions
- Shaped rewards
- Punishment
- Various weighting factors

In the end, I tested against shaped rewards and the imitation learning mentioned earlier in the paper.

My reward function is shown below (and included in the attached code in the deliverable, in the SatelliteAgent.cs file):

Distance to target:

$$R_{\text{distance}} = -\sqrt{(x_v - x_d)^2 + (y_v - y_d)^2}$$

Here (x_d, y_d) is the distance to the position where the servicing satellite is considered to be in station with the target object.

Penalty for high velocity:

$$R_{\text{velocity}} = \begin{cases} -0.1 & \text{if } |v_x| > 0.20 \\ -0.1 & \text{if } |v_y| > 0.20 \end{cases}$$

Here v_x and v_y are the velocities in the x axis and y axis of the servicing satellite.

In-Station Condition:

$$R_{\text{in\_station}} = \begin{cases} 1.0 & \text{if } d_x < \text{precision and } d_y < \text{precision} \\ 0.1 & \text{if } |v_x - v_{\text{target},x}| < 0.01 \text{ and } |v_y - v_{\text{target},y}| < 0.01 \end{cases}$$

This rewards the servicing satellite being close in the x and y axis relative to a precision constant defined in the code (in this case, 0.33f, a floating point value), and with a relative velocity difference of 0.01 or less in each axis.

Step Penalty As Time Constraint:

$$R_{\text{step}} = -0.0025$$

Every step, a small penalty is given, so that action is preferred.

End Episode Reward:

$$R_{\text{time}} = \{\text{end episode} \quad \text{if } t > 9999$$

This ends the episode if you reach the 100 second maximum in fixed time step updates.

Requirement Completion Reward:

$$R_{\text{collision}} = \{10000 \cdot (1 - \text{rel\_vel}) \quad \text{if rel\_vel} < 0.01$$

This greatly rewards touching the target point with a relative velocity at touch of less than 0.01 m/s, as that was the requirement (implicatively, it is less than 100 seconds, since the episode ends when 100 seconds in time steps is reached), the reward increasing the lower your relative velocity is.

PPO Hyperparameters

Batch_size	2048
Buffer_size	5000
Learning_Rate	1.0e-5
Beta	5.0e-4
Epsilon	2
Lambd	.99
Num_Epoch	10
Learning_Rate_Schedule	Linear
Beta_Schedule	Constant
Epsilon_Schedule	Linear

Neural Network Settings

Normalize	true
Hidden_units	256
Num_layers	3

Imitation Learning Settings

Type	GAIL	Behavioral Cloning
Demo	<attached_to_submission>	<attached_to_submission>
Strength	1.0	1.0

Whether to utilize extrinsic rewards was a setting within the configuration, a value between 0 and 1.0 for the weight with which extrinsic rewards, meaning the explicitly defined rewards functions that cause a call to the AddReward() method provided by Unity. When set, my reward functions were as follows:

The observations that were collected for use by the learning model were:

- Relative velocity on collisions (to try and meet the 0.01m/s threshold)
- End effector distance to the touch point
- The touch point coordinates themselves
- Base frame center point distance to touch point
- Vehicle position
- Vehicle velocity
- Target object position
- Goal vehicle position for being in station
- All thrust forces on the vehicle (force about x axis and y axis respectively)
- All torques acting on joints

The hardware and software environments were:

- Hardware environment
  - Windows 10 64-bit operating system, x64-based processor
  - CPU: AMD Ryzen 5 3600X 6-Core Processor 3.79 GHz
  - GPU: AMD Radeon RX 6700 XT 12GB
  - RAM: 64.0 GB
- Software environment
  - Pytorch: 2.2.2+cu121
  - Unity: 2023.2.10f1
  - ML-Agents:
    - Python package: 1.0.0
    - Unity package: 3.0.0

IV. Gathering Demonstration Data

When recording demonstration data for the two imitation learning portions of the control strategy, a simple controller utilizing arrow keys and a few other keys on the keyboard was used

to apply a static negative or positive torque or thrust, defined in the files attached to this submission. One method that I used that wasn't obvious to me at the beginning was to, once in station, send counted and timed torque commands utilizing a metronome in order to make the demonstration data more uniform. The demonstration recording was carried out 20+ times as follows:

1. Navigate via thrust commands to the in-station point and to match velocity with the object
2. Once in station, begin the torque command pattern, all beats occurring within a constant 105bpm, which was:
    a. 16 torque commands of 0.01f to joint 1 on 16 beats
    b. 6 to joint 2 on the next 6 beats, followed by 2 beats where I do nothing
    c. 1 to joint 3, followed by 7 where I do nothing
    d. 1 combination command, to both joint 1 and 2 on the next beat, followed by 7 doing nothing
    e. 1 command of -0.01f to joint 1 and 2 simultaneously followed by 7 beats
    f. Episode ends (almost always)

Almost 100% of the time that methodology enabled getting a reasonably soft touch on the debris. Small variability still existed though due to the fact that I could have timed some of them slightly wrong, the Update() method relative to the FixedUpdate() method (the physics simulation loop) could have been slightly different when I carried a given one out, as well as if a finger lingers on a key, it could be on for more than one loop, hypothetically, as the Update() method was being called roughly 90 frames per second (varying up and own slightly consistently), and the FixedUpdate() call for the physics simulation was being called once every 0.01 seconds (100 frames per second, statically). This methodology allowed me to be able to more efficiently record demonstration data with more reliable results, as the GAIL and BC configurations only allow attaching one demonstration file, and earlier demo files were not yielding as fruitful returns (though this is not definitively the cause, as there is some randomness due to the stochastic nature and I was making other configurations too), so if one tries to record 100 episodes of demonstrations and mess up 20 times, that could potentially corrupt the demonstration data. Future exploration opportunities may exist with methodologies to enhance human ability to generate demonstration data, through specific trainings and/or forms of carrying out tasks.

Two manual clamping methods I utilized were limiting angular velocity explicitly to a very large number (1 million), regardless of torque commands, for error avoidance and to disregard edge cases. Additionally, I decided to not issue any torque commands when not in-station via a simple conditional statement in the code, simplifying the problem to utilizing thrust in the x and y axes until the vehicle was in station. A simple change in the precision constant requirement for reward and the closeness requirement simultaneously didn't degrade the touch

moment such that the requirements wouldn't be met when successful and made it so that the algorithm was able to learn the methodology in far fewer episodes.

V. Results And Discussion

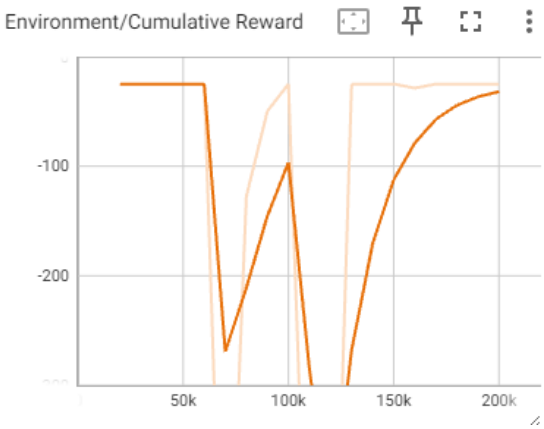
Behavioral Cloning and GAIL imitation learning strategies greatly reduced training time requirements and increased positive outcomes, and in this exploratory survey outperformed the use of the extrinsic reward paradigms that were attempted both with BC and GAIL, as well as without them. That isn't to say there aren't more effective pure PPO paradigms without imitation learning that would outperform (interestingly, the documentation points to extrinsic rewards plus BC and GAIL being highly likely to be the most effective combination, which seems intuitive and could potentially point to an issue with my particular extrinsic reward definition), but with the methods attempted in this exploration, purely training on BC and GAIL was most effective for me.

Trials of note in the latter portion of experimentation:

The below are 4 of the latter trials that were done, each labeled with the paradigm being utilized, results shown from TensorBoard, a utility that allows graph data to show how trials, episodes, reward, etc. increase. I show on a small number of episodes that the training as not on track with the first 3 and then the far better result on the 4th method:

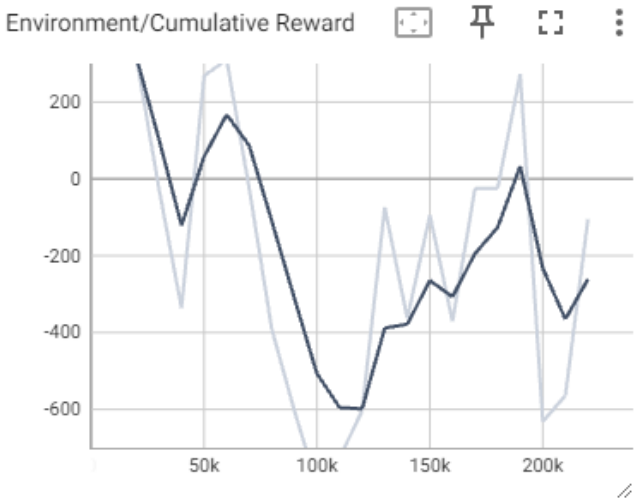
PPO without BC and GAIL and sparse rewards

On the small number of episodes trained on, this did not seem to be trending in any discernible direction, as was the case with similar reward structure on previous attempts with sparse rewards.



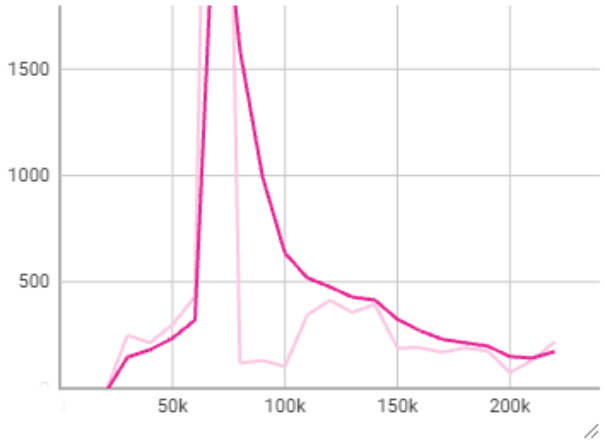
PPO without BC and GAIL and shaped rewards

This with my existing reward paradigm did not seem to be converging, that said it was on very few episodes as with the previous one, so it could be coincidental. Previous tests with similar reward paradigms didn't seem to converge though, either.



PPO with BC and GAIL and with extrinsic rewards, shaped ones

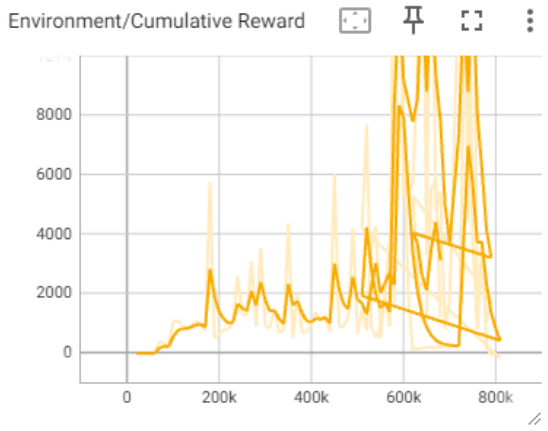
This had a successful early set of episodes, but then began to trend downward, which could mean that the BC and GAIL are conflicting and/or not compatible with each other, though it requires further examination and testing to confirm this.



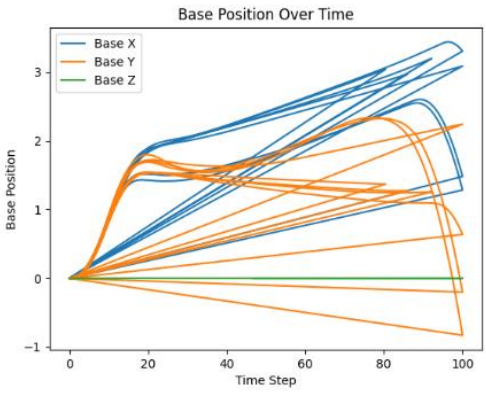
PPO with BC and GAIL and without extrinsic rewards

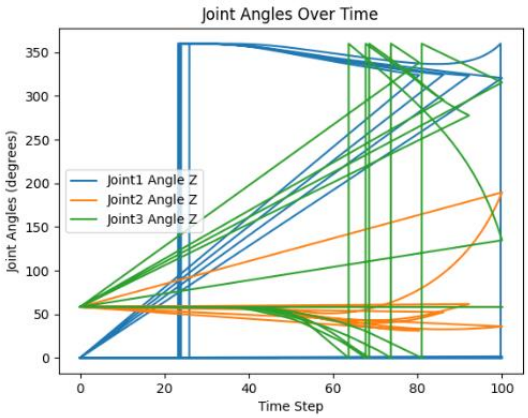
This was the most successful trial and policy across over 100 training attempts and policy generations of 50-100+ episodes per attempt (and some with over 1000 episodes, taking many hours each) with ML-Agents through the project. This training was carried out on less than 200 episodes utilizing just BC and GAIL. I also came to start using 'checkpoints' at this point, where earlier model states are saved so if, due to the fact that PPO is a stochastic algorithm, the training could be initialized again at a previous recent state with some of the learned behavioral policy

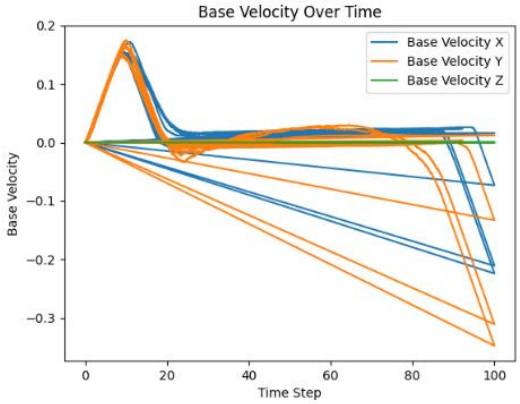
still in tact and without the divergence or less than desirable results from the last 5-10 episodes or so, thus allowing you to discard any bad luck and/or confirm whether there is a more systemic issue. The success of this trial points to the fact that it is important to have a defined extrinsic reward function that is compatible with BC and GAIL. If the behavior you are demonstrating is not compatible with the drive cultivated by the extrinsic reward definitions, then intuitively this would hinder training, though further exploration to confirm that would be prudent. The video included in the report deliverables, titled "ExampleOfPolicyInAction.mp4", utilizes this model and the graphs below are representative of performance of this model through a given episode. The attached DynamicsKinematics.cs, SatelliteAgent.cs, and SpaceJunkRemoval2D.yaml, respectively, are the dynamics code, agent definition code, and the yaml definition for the PPO algorithm, with details regarding the use (and disuse for previous cases, commented out) of BC and GAIL, and exclusion of extrinsic rewards in this run. It is in the DynamicsKinematics.cs that the soft touch calculation occurs, and it is logged in the example video. Additionally, in the submission comments, I'll leave a link to Google Drive directory that has the full code of the Unity project, which also contains the over 200 demonstration files from my testing out various methods of demonstration for imitation learning, for reference or questions.

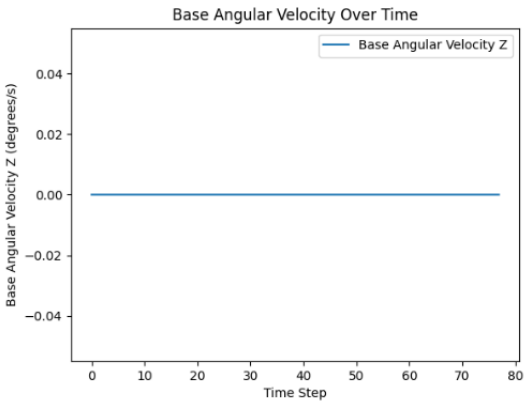


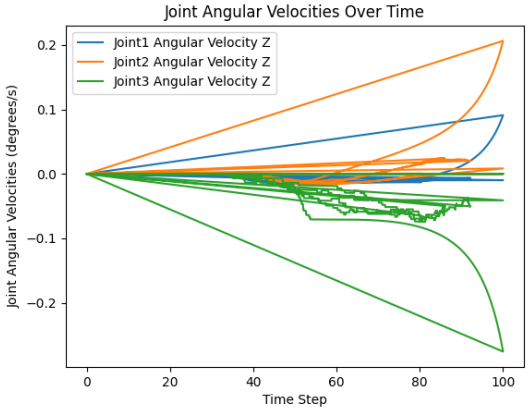
Some example state variables during runs utilizing the resultant model can be found here, including thrusts, joint velocities, torques, and positions:

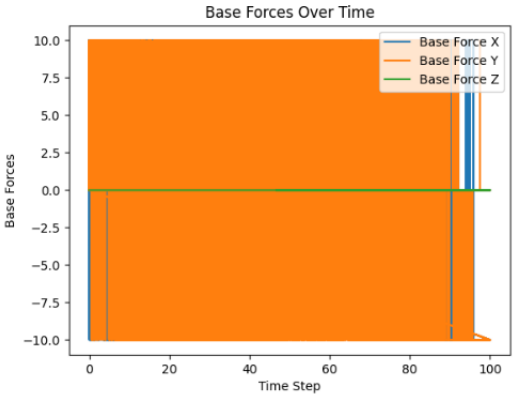


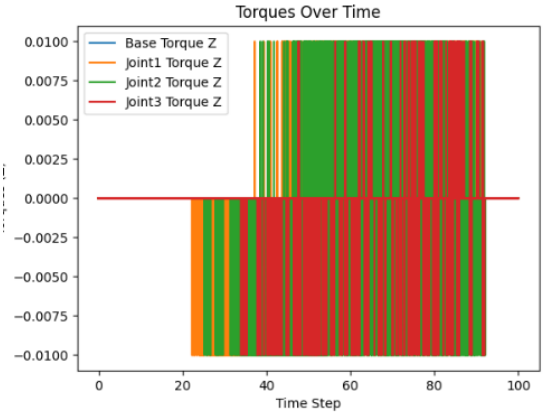












VI. Conclusion And Improvements For The Future

As the commercial space industry expands at the continued and seemingly exponential pace that it has been expanding at, it is important that we work toward ensuring that this industry is safe and sustainable as we attempt to be good stewards of space. Creating control policies and

coming to solutions to capture space debris is one of the ways in which that is done. This survey showed that Unity ML-Agents has capability to simulate the dynamics of a satellite servicer/debris system via the Euler Integration method, simulate collisions, and that deep reinforcement learning with imitation learning can be utilized to form a control policy for the task.

The Agent had a medium success rate, about 35-50%, with some trial runs occurring where 1 out of 2 would be successful and the other half would be partial successes, seemingly drifting away from the in-station point being one of the main areas where it needs to improve, but the progress on the task is promising, as it was able to complete it some of the time, so improving consistency with the policy is the next step that can be taken. With some more training episodes, it seems this could be enhanced greatly with not many more episodes. Training on more episodes (over 50 training attempts were tried, but not all of them did over 1000 episodes of training – some were less) and also training on more demonstration data is worthy of further exploration, as the results of this paper were carried out on a half hour or less of training data (due to constraints to one file of data for a given demonstration – enabling more than one demonstration file is worth pursuing as well, it seems, not for lack of effort, as over 200 demonstration recordings were generated), so more opportunity likely exists there. Precision is another area where there is room for further improvements, mostly in policy precision. The collision calculation could be more precise and had some interplay with the width of the rigid bodies within the physics simulation. Additionally, in order to get a successful result, the region for being considered in station was widened, which allowed a successful result as it appeared too strict relative to the number of episodes being trained on, as well as increase the range of velocities within which it could be hovering relative to the 0.02 m/s that the debris was moving to expedite training and get a good result. It is possible that more episodes and/or shaping rewards more effectively could have allowed for this. The same goes for experimenting more with hyperparameters, as there are vast combinations that can be explored.

Another opportunity for improvement and exploration into the utility of Unity and Reinforcement Learning for control systems for space debris capture is to push the boundaries of physics within Unity as a simulation engine, as there are other existing physics engines that it offers as an avenue of further exploration. Additionally, creating a more robust physics model from scratch and utilizing the other robust features that Unity has to offer could be of merit to explore further as well. Capturing debris on rendezvous without damaging the servicing vehicle or causing destruction (and potentially increasing Kessler syndrome effects) of the debris could be explored with a more robust physics and collision model.

Unity ML-Agents (and OpenAI gym) allows for environment randomization and there is likely utility in exploring both environment randomization and procedural creation of debris and obstacles to create a more generalized policy that can capture debris in many scenarios. It also allows for integration of ROS (Robotic Operating System) and sensor emulation (as well as hardware in the loop simulation), which can allow for use of ML-Agents with live hardware, as

well as with existing robotics initiatives steeped in ROS or that are custom, but have interfaces with the aforementioned technologies that can be integrated with ML-Agents and Unity.

As the commercial space industry continues to grow and AI efficacy continues to accelerate alongside it, deep reinforcement learning and other AI paradigms for generating control policies should be explored regularly to find all utility that can be found. This survey of deep reinforcement learning capabilities for PPO within Unity ML-Agents for generation of control policies found utility of imitation learning alongside deep reinforcement learning in the form of PPO. For simulating a servicing satellite rendezvousing with space debris, getting in station with it, then softly touching it, the resulting stochastic policy, enhanced by imitation learning, performed better in equivalent episodes than the other methods attempted, those being handcrafted extrinsic rewards, a mix of extrinsic and imitation learning, and for the non-imitation reward definitions, a mix of sparse and shaped rewards, with various focuses on weights and biases.

References

[1] R. Kauffman, Director, *Wild Wild Space.* [Film]. Hyperobject Industries, 2024.

[2] D. J. K. a. B. G. Cour-Palais, "Collision Frequency of Artificial Satellites: The Creation of a Debris Belt," *JOURNAL OF GEOPHYSICAL RESEARCH,* vol. 83, no. A6, 1 June 1978.

[3] A. C. a. D. Kim, "Ground Robotic Platform for Simulation of On-Orbit Servicing Missions," *Journal Of Aerospace Information Systems,* vol. 19, no. 7, pp. 480-493, 2022.

[4] V.-P. B. E. T. A. C. J. H. C. E. C. G. Y. G. H. H. M. M. D. L. Arthur Juliani, "Unity: A General Platform for Intelligent Agents," *ArXiv,* vol. https://arxiv.org/abs/1809.02627, 2018.

[5] R. M. R. M. a. R. D. Yusef Savid, "Simulated Autonomous Driving Using Reinforcement Learning: A Comparative Study on Unity's ML-Agents Framework," *Feature Papers in Information in 2023,* 2023.

[6] F. W. P. D. A. R. O. K. O. John Schulman, "Proximal Policy Optimization Algorithms," *https://arxiv.org/abs/1707.06347,* 2017.