Development of A Reinforcement Learning Framework for Spacecraft Operations

Daniel W. Kolano*
Stanford University, Stanford, CA, 94305

This paper introduces a framework for rapid development and iteration of reinforcement learning models for optimization of spacecraft operations based on the System Tool Kit. This framework allows development of reinforcement learning models for a wide variety of spacecraft operations from launch to orbit to deep space without the creation of underlying physical models. The utility of the framework is demonstrated through through a proof-of-concept spacecraft rendezvous problem, on which a soft actor-critic reinforcement learning model is trained with the help of the System Tool Kit interface. Results showed that the trained model showed moderate success in solving the proof-of-concept problem, but that the computational overhead of the training environment was unnecessarily large for such a simple problem. However, the framework was shown to scale at a favorable rate to more complex problems, making it a reasonable choice as problem complexity increases.

I. Nomenclature

A. Symbols

λ	=	longitude
ϕ	=	geodetic latitude
a	=	altitude
γ	=	azimuth
α	=	horizontal flight path angle
v	=	velocity
$\mathcal{U}(a,b)$	=	continuous uniform distribution over (a,b)
$\mathcal{A}$	=	action space
$\mathcal{S}$	=	state space
F	=	thrust
$\dot{m}$	=	mass flow rate
g	=	earth surface gravitational acceleration
x	=	x-coordinate (ICRF coordinate system [1])
y	=	y-coordinate (ICRF coordinate system)
r	=	reward

B. Subscripts

$_a$	=	agent
$_t$	=	target
$_x$	=	x-velocity (ICRF coordinate system)
$_y$	=	x-velocity (ICRF coordinate system)

*Student, Aeronautics and Astronautics

II. Introduction

There are a wide variety of potential applications of reinforcement learning methods to improving spacecraft operations and maneuvers. These include optimization of low-thrust trajectories [2], collision avoidance [3], and optimal distribution of constellation coverage [4]. In general, developing and validating high-fidelity models of these operations that can be linked to a learning environment can be the largest challenge in research into these areas. However, extremely powerful generalized tools for modelling of space operations already exist and have been widely used and validated throughout research and industry. One of the most powerful of these is the "System Tool Kit" (STK) developed by AGI [5]. This software allows the user to create and simulate scenarios including rockets, satellites, ground stations, constellations, and many more. The purpose of this work is to create a flexible framework that can interface between STK and the Tensorflow tf-agents reinforcement learning API, and demonstrate the use of this framework through a proof-of-concept model.

III. Problem Statement

Due to constraints on time and processing power, a highly simplified proof-of-concept problem was implemented and is presented here to demonstrate the functionality of the framework created. The problem selected is an orbital rendezvous between an active and passive satellite. The parameters of the problem are highly constrained to allow for rapid iteration and development, but remain easily-extensible.

The environment is initialized with an "agent" (active) and "target" (passive) satellite, with the goal being for the agent to match the target's position and velocity while using a minimum of fuel. The problem can be modeled as a simple Markov Decision Process, with the state being the current relative position and velocity of the agent (Through a judicious choice of orbit we can avoid including the target position), the action being the thrust applied by the agent, the transition function being the physics of orbital mechanics, and the reward being a function of the distance between the satellites and the thrust level applied.

A. Simulation Setup

The target satellite always placed in an equatorial geostationary orbit such that it's position in an Earth-relative reference frame will be constant. The agent is placed in a nearby equatorial orbit, but with longitude and altitude drawn from a uniform distribution over a small range centered on the target's initial location. The parameters of each satellite's orbits are given in Table 1.

Table 1 Initial orbital parameters of the agent and target satellites

Orbital Parameter	Agent	Target
Longitude, deg	$\mathcal{U}(-0.1, 0.1)$	0
Geodetic Latitude, deg	0	0
Altitude, km	$35,786 + \mathcal{U}(-250, 250)$	35,786
Horizontal Flight Path Angle, deg	0	0
Azimuth, deg	90	90
Velocity, km/s	3.07	3.07

By setting the geodetic latitude to 0 degrees and azimuth to 90 degrees, we generate co-planar equatorial orbits. This allows us to reduce the orbital mechanics from three to two dimensions as long as we provide no impulse away from this plane (and omit any effects due to third bodies, the Earth's non-spherical gravity well, etc.).

A "Mixed Sperical" coordinate system [6] was chosen for this implementation as this system allows for comparison of nearby co-planar orbits in a natural way. The vector between the positions of the satellites is simply the difference in their longitude and altitude: $[\gamma_t - \gamma_a, a_t - a_a]^T$. Other primary coordinate systems considered were simple Cartesian coordinates and classical Keplerian orbital elements. However, Cartesian coordinates account poorly for the fundamentally spherical nature of orbital mechanics, and Keplerian orbital elements make determining relative positions of nearby objects quite difficult.

The environment then propagates forward in steps of 3.5 seconds, with the agent taking an action at each timestep. This is repeated for an episode length of 500 seconds (except if an early stopping condition is met as described below), before the simulation is reset.

The enviroment was initially created using the STK user interface as a scenario with two satellites in Earth orbit. The constant parameters of the simulation can be set in this interface, and those that would need to change are left to the Python API to set up. For each timestep, the current state of the simulation is read from STK, then an action is chosen by the agent and sent to STK, and finally the simulation is propagated one timestep forward and the new state is read to calculate a reward. This environment is shown in Figure 1.

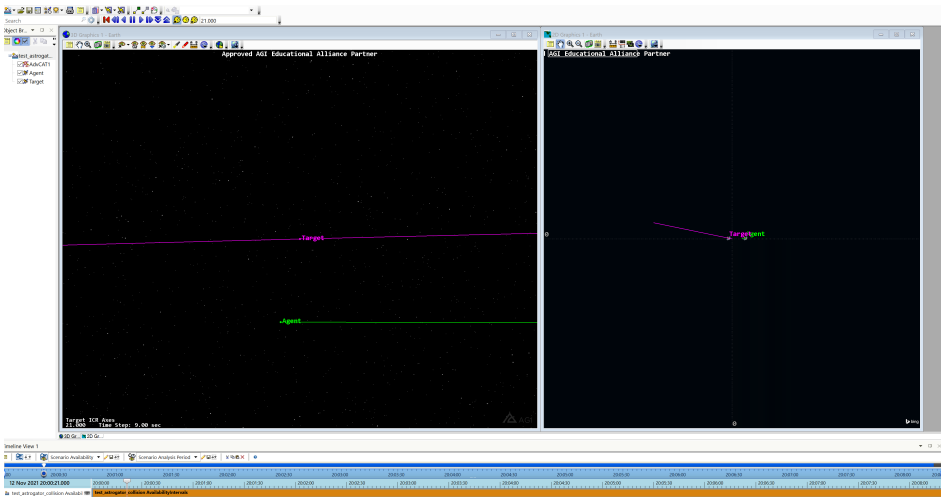

Fig. 1 The simulation environment with the agent satellite in green and the target in purple, with trajectories shown.

B. Action Space

The action space of the problem is simply the desired thrust-vector of the satellite in-plane:

$$\mathcal{A} = [F_x, F_y]^T \tag{1}$$

Where x and y are in an Earth ICRF coordinate system. Each component of the thrust vector is allowed to be on the continuous range $[-1, 1]$.

To model uncertainty in the actual engine efficiency of the engine and orientation of the satellite. The actual actions taken are drawn from a bivariate normal distribution centered on F_x, F_y with covariance matrix $\mathbf{I}_{2x2}[0.1, 0.1]^T$, giving resulting actual thrust vector components F'_x, F'_y.

The thrust efficiency η_F is then calculated as the two-norm of the thrust vector:

$$\eta_F = F_{applied}/F_{max} = \sqrt{F_x'^2 + F_y'^2} \tag{2}$$

C. State Representation

Only four continuous variables are needed to fully represent the state of the system given the constraints laid out above. These form chosen is given in Table 2

Table 2 State Space Variables

Variable ID	Formula	Range
1	v_x	[-180, 180]
2	v_y	[0, ∞]
3	$250(\lambda_t - \lambda_a)$	[-45,000, 45,000]
4	$a_t - a_a$	[35,786, $-\infty$]

Note that the difference in longitude $\lambda_t - \lambda_a$ is multiplied by a factor of 250 to keep it at the same magnitude as the remaining variables as the difference in longitude is generally much less than 1 degree.

D. Transition Function

The kinematics of the transition function are all handled through the STK interface. Actions are modeled and propagated through timesteps through the STK Astrogator framework, and kinematics of the passive satellite are propagated using a J2 perturbation model [7].

E. Rewards and Stopping Conditions

1. Episodic Reward

The reward at each timestep was calculated as follows:

$$r = -\sqrt{(x_t - x_a)^2 + (y_t - y_a)^2}/100 - 0.005\eta_F \tag{3}$$

This reward penalizes the agent linearly on $[-1, 0]$ based on its distance from the edge of a boundary region (described below), and linearly based on its applied thrust (analagous to a penalty for fuel consumption).

2. Early Stopping Conditions

An episode would be ended early under either of two conditions as given in Table 3.

Table 3 Early Stopping Conditions

Condition	Description	Reward (r)
$\sqrt{(x_t - x_a)^2 + (y_t - y_a)^2} > 100km$	Position out of bounds	$-2 *$ (# of remaining timesteps)
$\sqrt{(x_t - x_a)^2 + (y_t - y_a)^2} > 0.05km$ and $v < 0.01km/s$	Successful Rendezvous	$2 * (1 +$ # of remaining timesteps)

The position out of bounds condition was created to speed training by giving very low rewards for policies that took the satellite in the clear wrong direction, and the reward was chosen to be twice the worst possible reward if the episode was not ended early. The successful rendezvous condition rewards a rapid rendezvous more strongly.

If neither of these conditions were met before the end of the episode, the episode was simply ended with the standard reward.

IV. Approach

A. Prior Work

Hovell and Ulrich presented a similar problem in [8], however this problem had a far simplified kinematics model and a more complicated goal as they modeled the problem in an inertial reference frame with no external forces on the agent or target, and allowed the agent to directly input velocity commands rather than accelerations, but required the agent to dock from a specified direction. Their work used a distributed distributional deep deterministic policy gradient algorithm (D4PG), a type of actor-critic algorithm, to learn the state-action value function. This method showed success in a variety of docking scenarios including those with disturbances and obstacles.

B. Model

To model the Q-values of this problem, a Soft Actor-Critic (SAC) model [9] was selected. Simple Q-learning was attempted using discretizations of the state and action spaces, but was unable to capture the non-linear nature of the problem well. Additionally, as a major goal of this exploration was to show the applicability of this framework to more complex problems, the SAC model will be more easily-extensible to a wider range of problems, particularly those with both continuous action and state spaces. Finally, as described above, actor-critic methods are strong candidates for this genre of problem.

The SAC model uses a neural network to model the Q-value distribution over actions at each state. Thus a form of this neural network must be selected. In fact, a separate network is created for both the actor and the critic. As both the state and action space for this problem are rather small, through some trial and error it was determined that a relatively small network with hidden layers of size 32 and 16 for both the actor and critic were ideal. Larger networks would often

take a very long time to train out of local optima, causing the agent to gravitate towards random points in state space that may have provided a strong reward once. Smaller networks would not be able to incorporate enough information to provide an optimal solution.

These networks were trained in a traditional manner using experience replay and Adam optimizers with a learning rate of 0.0004, and evaluated every 2000 training steps on a separate environment.

V. Analysis and Results

A. Training

The training statistics are shown in Fig. 2. The agent was trained for 55,000 iterations over approximately 16 hours. This results in an average rate of approximately one iteration per second. As expected, the average returns increase over iterations and the average loss decreases. However, it is notable that 79% of the time for a simulation step to be taken was due to communication between Python and the STK program. This implies that this model could be scaled to more complex models within STK with relatively low performance impact on training provided a similar level of communication between environments is required.

The average initial return of under -300 is the result of the agent going out of bounds nearly immediately on each episode, and the eventual average returns above 0 show the agent eventually arriving within the target range consistently. As shown by the relatively high variation in average returns even at the end of training, this network is clearly not yet fully trained. However, due to constraints on usage of the computer the model was being trained on (e.g. I needed my computer for other work), training was halted at this point.

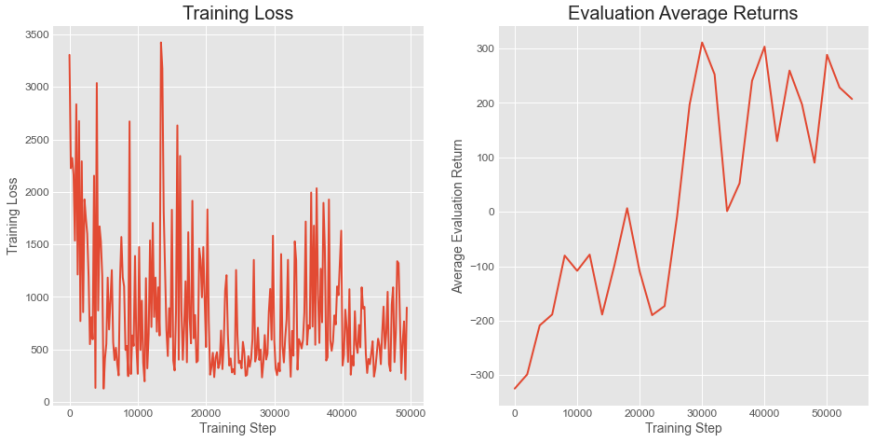

Fig. 2 Training performance for the agent over 55,000 training steps.

B. Learned Policy

Several examples of representative paths taken by the agent are shown in Figure 3. It can be seen that the agent does not consistently take the most direct path to the target location, but instead would turn onto a different track before continuing towards the target. Evaluation of the learned policy over 300 episodes showed that it successfully arrived at the target 98% of the time, with the failures occurring when it moved too slowly and the episode ended before it could arrive. In this evaluation, it was found that the agent took an average of 183 seconds to reach the target (if it did in fact get there).

VI. Discussion

One of the most notable aspects of the observed observed final policy of the agent is the tendency to often follow curved paths towards the target. This is particularly notable in the bottom track in Fig. 3, where the agent turns quite markedly "up" before turning back to the target. In observing these tendencies more closely, as well as the early rounds of training, it was found that the agent was gravitating towards "preferred tracks" that it had found to reach the target

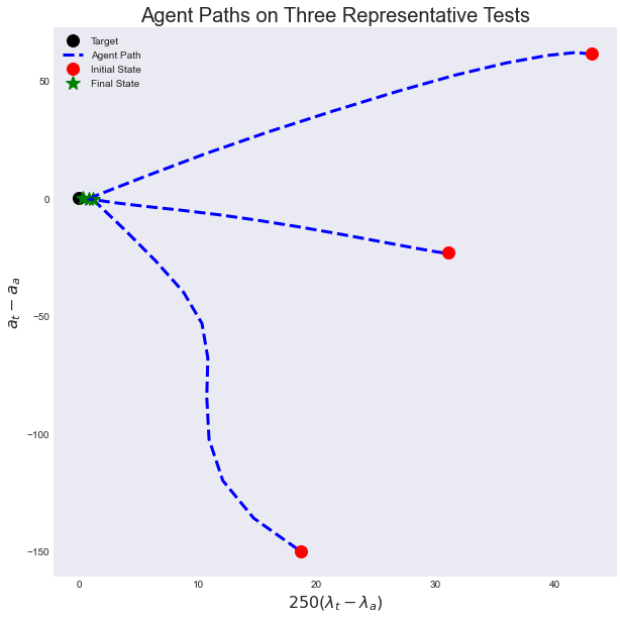

Fig. 3 Representative agent trajectories from several evaluation runs. Note the distinct changes in direction in the upper and lower paths.

early in training, much like my miniature horses and donkeys at home following the same winding paths through their fields so much that they wear away the grass, and then will go from other locations to follow that track since they have worn down the grass and know it will get them where they're going (Take a break and see Fig. 4 and Fig. 5 in Appendix A for some pictures of a miniature horse and a miniature donkey, and Fig. 6 for an example of some of these preferred tracks created by humans). In much the same way as my animals, the agent found that there was a good way to get to the target from a certain state, and thus assigned a high value to being in that state since it knows it can move from there to the target to get a high reward. It then recursively assigned higher values to actions that would take it towards the state that it could reach the target from, causing the agent to gravitate towards paths it had already taken, even if these were not the optimal strategy overall. A potential method of reducing this behavior in future work would be to increase the exploration parameter and reduce the learning rate, allowing the agent to find the more optimal tracks instead of defaulting to existing ones.

The training time taken for this problem is also far higher than would be ideal for a relatively simple problem such as this. However, as the training time will scale much slower than the complexity of the simulation when using this framework, and the investment in setting up a simulation always remains quite low compared to creating it from scratch, this framework becomes far more attractive when considering more complex situations such as modelling constellations or complex orbital transfers using gravity assists. Additionally, the computational time could easily be reduced significantly if parallel simulations are run in separate instances of STK, and experiences are batched together for replay.

VII. Conclusion

In this work a framework was created for creation of reinforcement learning environments leveraging the capabilities of the STK software. A proof-of-concept orbital rendezvous problem was presented and modeled using this framework and a Soft Actor-Critic learning model. The results of this model demonstrated the capabilities of this framework as a baseline for training reinforcement learning models. The computational overhead of utilizing the STK framework was shown to be a high burden for small problems such as that presented here, but the benefits of a pre-built and validated simulation environment with built-in visualization capabilities grow with the complexity of the problem being solved, while the computational overhead grows at a much slower rate, making this framework a reasonable choice for more complex problems.

Appendix A

Fig. 4 A miniature horse. Her name is Maitai

Fig. 5 A miniature donkey. Her name is Brighty.

Appendix B: Code

All code is available on Github.

Fig. 6 Some established but potentially suboptimal paths in the Pearson-Arastradero Preserve

References

[1] Charlot, P., Jacobs, C. S., Gordon, D., Lambert, S., de Witt, A., Böhm, J., Fey, A. L., Heinkelmann, R., Skurikhina, E., Titov, O., and et al., "The third realization of the International Celestial Reference Frame by Very Long Baseline Interferometry," *Astronomy amp; Astrophysics*, Vol. 644, 2020. https://doi.org/10.1051/0004-6361/202038368.

[2] Zavoli, A., and Federici, L., "Reinforcement learning for robust trajectory design of interplanetary missions," *Journal of Guidance, Control, and Dynamics*, Vol. 44, No. 8, 2021, p. 1440–1453. https://doi.org/10.2514/1.g005794.

[3] Kabir, M. R., Faysal, T. I., Hossain, M. S., Shorno, J. N., and Siddique, S., "A satellite collision avoidance system based on general regression neural network," *2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, 2020. https://doi.org/10.1109/bdcat50828.2020.00004.

[4] Chang, J., Duquette, R., Thai, K., Zhou, T., Pham, M., Lin, V., and Wood, K., "Combining genetic algorithms and machine learning for exploring the Navigation Satellite Constellation Design Tradespace," *Proceedings of the 33rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2020)*, 2020. https://doi.org/10.33012/2020.17616.

[5] AGI, "Systems tool kit (STK)," , 2021. URL https://www.agi.com/products/stk.

[6] AGI, "Mixed spherical coordinate type," , 0AD. URL https://help.agi.com/stk/11.0.1/Content/stk/vehSat_coordType_mixedSpherical.htm#:~:text=The%20Mixed%20Spherical%20coordinate%20type,parameters%20with%20inertial%20velocity%20parameters.&text=The%20object%27s%20position%20above%20or,surface%20of%20the%20reference%20ellipsoid.

[7] Solutions, A., 2014. URL https://ai-solutions.com/_freeflyeruniversityguide/j2_perturbation.htm.

[8] Hovell, K., and Ulrich, S., "Deep Reinforcement Learning for Spacecraft Proximity Operations Guidance," *Journal of Spacecraft and Rockets*, Vol. 58, No. 2, 2021, p. 254–264. https://doi.org/10.2514/1.a34838.

[9] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 80, edited by J. Dy and A. Krause, PMLR, 2018, pp. 1861–1870. URL https://proceedings.mlr.press/v80/haarnoja18b.html.