Reinforcement Learning for LEO Satellite Momentum Stabilization (New Dataset)

Anirudh Aatresh, Joseph Breeden, Kevin Tan, Liliang Wang

April 22, 2022

1 Abstract

In this report, we consider the problem of stabilizing the internal momentum of a satellite in Low Earth Orbit (LEO) that is controlled by reaction wheels. Over time, the angular momentum of these wheels increases to counteract disturbances, and must eventually be unloaded to prevent the wheels from exceeding their maximum allowable angular velocities. We propose an offline Reinforcement Learning (RL) approach to developing a momentum stabilization control law using only magnetic actuators and the Earth's magnetic field. We present results for several variations of this offline RL algorithm with differing input sets and compare the performance of the developed policies in three different simulated disturbance environments. We conclude that RL can perform better than a classical control law, but questions remain regarding the stability of the RL approach.

2 Introduction

Space is commonly assumed to be a pure vacuum, but in reality, it is not, especially for satellites in Low Earth Orbits (LEO). Real satellites encounter disturbances including upper atmospheric drag, solar radiation pressure, gravity gradient, magnetic field interactions, etc. In the long term (months-years), these disturbance forces, especially drag, tend to make a spacecraft's orbit decay. Additionally, in the short term (hours), the resultant disturbance torques induce satellite rotations, which may conflict with mission objectives. Most commonly, onboard reaction wheels are used to counter these torques, but if the torque is non-zero-mean (as drag usually is), then these wheels will spin faster and faster until the wheels' angular momentum needs to be unloaded. A cost effective way to unload this momentum is to use magnetic torquer bars (MTBs) that exert a torque in the presence of the Earth's magnetic field to provide opposing torques while unloading stored momentum in the wheels.

However, existing linear control laws for MTBs provide solutions to this problem only when disturbances are fairly small. It is difficult to improve upon these control laws due to the irregularity of Earth's magnetic field. The momentum saturation problem is also heightened for CubeSats, which frequently have high area to mass ratios and small actuators. Reinforcement learning (RL) algorithms have been shown to provide effective solutions to complex control problems, and we attempt to extend these advancements to develop control laws that stabilize a satellite's angular momentum. We base our experiments in this project on a simulated environment developed by NASA called 42 [1], that is capable of emulating satellite attitude evolution given control inputs. Based on data generated using many runs of this simulator, we study the performance of offline actor-critic based algorithms to attempt to solve the momentum stabilization problem in high disturbance cases where existing control methods are inadequate. To the best of our knowledge, reinforcement learning methods have not previously been used to solve this problem, so the purpose of this project was to study their performance when trained on this newly created dataset.

We choose to learn the control laws offline, using previously collected datasets from 42. We do so in part because of the presence of uncontrolled states in the dynamics, and in order to leave open the possibility of learning a policy on real satellite data in the future, as it would be prohibitively dangerous to learn a policy online from scratch in a real satellite environment. We used three offline RL algorithms, 1) conservative Q-learning (CQL) [2] 2) twin delayed deep deterministic policy gradient with behavior cloning (TD3+BC) [3], and 3) conservative offline model-based policy optimization (COMBO) [4] to learn control policies offline. In addition to learning the policies, we also conducted RL hyperparameter tuning, for which we use fitted-Q evaluation (FQE) [5] as an offline estimator of the learned control law's performance to guide our hyperparameter search. The implementation of the above was assisted via the Python library d3rlpy [6], which provides implementations of the above algorithms with a Scikit-Learn style interface. We then evaluate the learned control laws by simulating their performance on a CubeSat in 42 [1], and find certain cases where the learned control laws outperform the baseline control law used to generate the data.

Statement about prior research: This project topic is motivated by one of the team members' prior work in industry with MTB control laws and 42. However, none of the team members had completed any prior research involving machine learning with satellites prior to this course.

3 Problem Formulation

A sample CubeSat as rendered in the 42 simulator is shown in Fig 1. The CubeSat is modelled as a rigid body with 6 positional degrees of freedom, 6 rotational degrees of freedom, and 3 internal reaction wheels. For simplicity, assume the CubeSat's orbital radius ($a = 6778$ km), eccentricity ($e = 0$), and inclination ($i = 80°$) are fixed. Within this selected class of orbits, the CubeSat's position is described entirely by the angles ν and Θ shown in Fig 2. Here, ν is the true anomaly, which represents the position of the satellite in its orbit; Ω is the right ascension of the ascending node of the orbit; Θ is defined as $\Theta = \Omega - \theta$, where θ captures the current rotation of the Earth in the day. Let β denote the angle between the local Sun vector and a vector normal to the orbital plane, so β describes the position of the CubeSat's orbit with respect to the Sun, which will influence the magnitude of atmospheric drag disturbances. The CubeSat's position is thus reduced to three degrees of freedom (ν, Θ, β). Note that the satellite's future orbit is given entirely by its initial conditions (i.e. we assume there are no onboard thrusters), so (ν, Θ, β) are uncontrolled states, though the dynamics of these states are known to high precision.

Next, assume the satellite's orientation is fixed in a Local-Vertical-Local-Horizontal (LVLH) frame in the orientation shown in Fig. 1 (where the magenta lines $\hat{l}_1, \hat{l}_2, \hat{l}_3$ are the LVLH frame) to eliminate all rotational degrees of freedom, and assume that the internal reaction wheels perfectly account for all disturbances. The satellite's internal momentum is $H \in \mathbb{R}^3$, and the complete state after these simplifications is $x = (\nu, \Theta, \beta, H) \in \mathbb{R}^6$. The wheel dynamics are

$$\dot{H} = T_{dist} + T_{mag} - \sqrt{\frac{\mu}{a^3}} \hat{h} \times H \tag{1}$$

where μ, a, and $\hat{h}$ are known quantities, T_{dist} is the disturbance torque, and T_{mag} is the torque from the onboard MTBs. Here, T_{dist} is a function of both the position (ν, Θ, β) and unknown parameters $(c_{10.7}, c_{ap})$ that vary between simulations. The magnetic torque satisfies

$$T_{mag} = u \times B \tag{2}$$

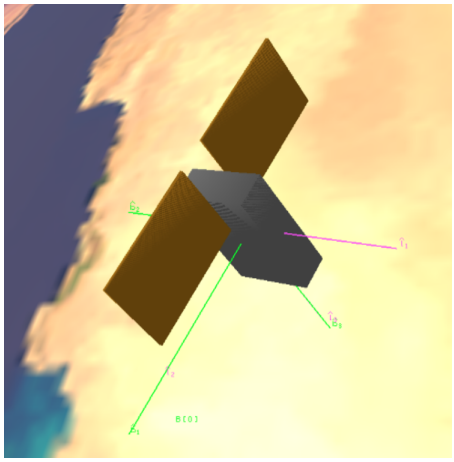

Figure 1: CubeSat

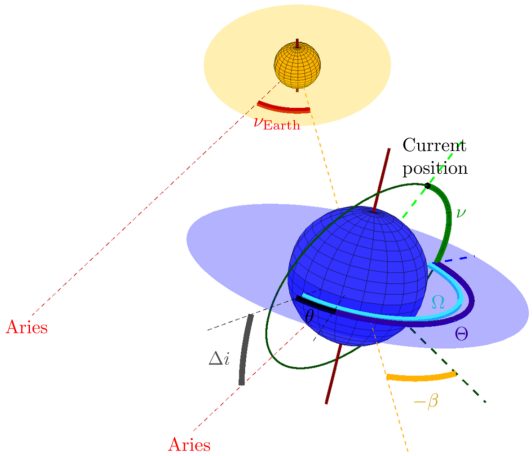

Figure 2: The satellite around the Earth and the Sun (not to scale)

where $u \in \mathbb{R}^3$ is the commanded satellite magnetic field, which is our control input, and B is the Earth's magnetic field. Due to the cross product in (2), we can only instantaneously exert torques T_{mag} in a 2D subspace of $\mathbb{R}^3$, and we assume that the actuators are limited to $\|u\|_\infty \leq u_{max}$. The quantity $B = B(\nu, \Theta) \in \mathbb{R}^3$ is the magnetic field of the Earth at the point that the satellite is currently passing through, which is a well-modelled but highly complex function. This complexity is what makes creating nonlinear controllers challenging, and is why we are interested in RL. For this project, we assume perfect measurement of all states (ν, Θ, β, H) are known to the controller, as the sensors used in practice are very precise.

The total internal momentum $\|H\|$ varies throughout an orbit, and the maximum expected value of $\|H\|$ determines the required size of the reaction wheels. Larger reaction wheels will require more mass, volume, and power, which are all expensive resources on a CubeSat. Thus, the objective is to find a policy $\pi(x)$ that minimizes this peak momentum. To better condition the learning algorithms, we instead use the instantaneous reward

$$r(x, u) = -\|H\| \,. \tag{3}$$

Note that the momentum plots encountered in this project often appear to diverge, as illustrated in Fig. 4 in Section 5. This is because the orientation chosen for experiments causes the CubeSat solar arrays to produce maximal torque from atmospheric drag. This particularly challenging hardware design was encountered in one of the author's prior work, and to date, no policy stabilizing such a CubeSat (for reasonably sized actuators) has been found. We are interested in whether RL-derived policies can solve this problem, and if not, whether they can still reduce the needed actuator sizing for more reasonably-designed hardware.

4 Methodology

4.1 Overview

The solution process can be broken down into the following three steps:

1. Collect data by running 42 many times,
2. Learn a policy offline using the collected data,
3. Test the generated policy in 42 with a few selected orbits and evaluate the results.

All three steps were repeated multiple times.

4.2 Data Collection with 42

We generated data sets using four different behavior policies: 1) a data set using the results of a nominal control law $u_{nom} = \frac{10^6}{\|B\|^2} H \times B$, 2) a data set where every control action was uniformly randomly sampled from the control bounds, 3) a data set using partially stochastic control inputs $u_{stoch} = u_{nom} + y$, $y \sim \mathcal{N}(0, 0.2\,\mathrm{diag}(|u_{nom,x}|, |u_{nom,y}|, |u_{nom,z}|))$, where $u_{nom} = [u_{nom,x}; u_{nom,y}; u_{nom,z}]$ and 4) a data set using constant control inputs for large sections of the orbit[1]. We started by choosing control inputs every 0.2 seconds (the time-step of the simulator), but later switched to sampling control inputs every 5 seconds to reduce the amount of data processed.

We also considered two different behaviors for the $(c_{10.7}, c_{ap})$ values during data generation. First, we set $c_{10.7} \sim \mathcal{U}(100, 230)$, $c_{ap} \sim \mathcal{U}(50, 100)$ at the beginning of every training simulation. This most realistically models real behavior, as these parameters vary very slowly. However, after computing initial results and noticing that our algorithms performed very differently when tested under different $(c_{10.7}, c_{ap})$ values, we decided to regenerate training data with sampling $(c_{10.7}, c_{ap})$ randomly from the above distributions every 5 minutes of simulated time. For simplicity, all orbits originated from the same initial conditions, and thus followed the same (ν, Θ, β) trajectories.

4.3 Algorithms Used to Learn Policies

Offline RL is known to be a difficult problem, with one notable issue being the over-estimation of the Q-function on out-of-dataset state-action tuples. To alleviate this issue, we explored the use of the following offline RL algorithms: 1) conservative Q-learning (CQL) [2], 2) twin delayed deep deterministic policy gradient with behavior cloning (TD3+BC) [3], and 3) conservative offline model-based policy optimization (COMBO) [4].

CQL produces a lower bound on the Q-value of a given policy, which is incorporated into the policy evaluation stage of soft actor-critic algorithm. TD3+BC improves TD3 by adding a behavior cloning term to TD3's objective function of deterministic policy gradient, to regularize the learned policy. The BC term penalizes actions different from those taken under the behavior policy. COMBO learns a probabilistic model of the state-transition dynamics from the data, and then regularizes a learned value function on unseen state-action pairs taken under rollouts of the model. In theory, these algorithms are capable of mitigating the problem of distributional shift between the behavior policy and the learned policy, which induces Q-function over-estimation in offline RL. However, in practice, we only proceed with extensive experiments on CQL and TD3+BC, as COMBO performed just as well as providing no control at all in early experiments. Though these early experiments only included four orbits of data from which to learn a dynamics model, and thus COMBO might perform better with more data, we decided to focus on the other two methods given the limited computing resources available.

In learning control laws offline, we used the d3rlpy [6] library's implementation of these offline RL algorithms. d3rlpy is a Python library implementing a wide variety of state-of-the-art offline RL methods, with a Scikit-Learn interface. In addition to the data set and algorithm selection, we also input to the d3rlpy toolbox the neural net architectures and hyperparameter choices described in the following sections.

[1]Dataset 4 was the result of a bug in the data generation code. By "large sections", we mean that the control input was piecewise constant and changed 1-4 times during each orbit. The exact number of changes was different during each orbit due to the bug. We include this data set because one of the policies learned from it performed surprisingly well.

4.4 Neural Function Approximation Architecture

We used three different neural network architectures in our experiments. The first neural network architecture used two hidden layers of 256 units each with ReLU activation functions for both the actor and the critic. This was the default choice within d3rlpy, and the primary choice for our experiments. In this report, we refer to this as architecture A.

The second neural network architecture had 1) five hidden layers of 12, 24, 36, 24, 12 hidden units respectively for the actor, and 2) four hidden layers of 12, 24, 24, 12 hidden units respectively for the critic. Each hidden layer used in the second set neural network architecture had a ReLU activation function, was batch normalized, and had a dropout rate of 0.2. We refer to this configuration as architecture B. We experimented with the use of this neural network architecture to better understand the influence of hidden layer sizes and network depths on model performance.

After noticing poor performance with the second neural network architecture, we found prior research by Sinha et. al. [7] explaining that deeper neural networks, such as that in architecture B, do not necessarily correspond to better performance. We therefore experimented with a modified version of the DenseNet architecture from Sinha et. al., with 1) four hidden layers of 24, 36, 36, 24 hidden units respectively for the actor, and 2) three hidden layers of 24, 36, 24 hidden units respectively for the critic. Like in architecture B, each hidden layer used in this third architecture had a ReLU activation function, was batch normalized, and had a dropout rate of 0.2. Unlike architecture B, however, we incorporated dense connections from the inputs to each of the layers, as Sinha et. al. suggest. We refer to this configuration as architecture C in Tables 1-2.

4.5 Policy Evaluation in 42

After learning control laws offline, we then evaluated the trained policies in 42. In each evaluation, we used the same initial conditions as in the dataset, but with three sets of unknown parameters $(c_{10.7}, c_{ap}) = (150, 50)$, $(c_{10.7}, c_{ap}) = (180, 70)$, and $(c_{10.7}, c_{ap}) = (210, 90)$, which were held constant throughout each evaluation orbit. The policies were ranked based on the metric

$$R = \sum_{j \in \{1,2,3\}} R_j, \quad R_j = \frac{1}{N} \sum_{i}^{N} r(x_j(t_i), u_j(t_i)) \tag{4}$$

where j denotes the set of unknown parameters. Here, R_1 denotes the set of unknowns indicating relatively small disturbances $(c_{10.7}, c_{ap}) = (150, 50)$, R_2 denotes the a moderate level of disturbances $(c_{10.7}, c_{ap}) = (180, 70)$, while R_3 denotes the set of unknowns indicating large disturbances $(c_{10.7}, c_{ap}) = (210, 90)$. We considered testing at even smaller and larger disturbance values, but for the neural nets to be effective at such values would require sampling the training data from wider distributions of $(c_{10.7}, c_{ap})$ than described in Section 4.2.

4.6 Confounding

Recall that the underlying dynamics model had two unknown parameters $(c_{10.7}, c_{ap})$ that affect the disturbance torque T_{dist}. These correspond to the upper atmospheric $F_{10.7}$ and Ap-indices (stored in $c_{10.7}$ and c_{ap} respectively), where the former is a measure of current solar activity while the latter is a measure of current geomagnetic activity. These indices greatly affect the magnitude of the disturbance T_{dist} and typically vary on daily or hourly timescales. Thus, $\|T_{dist}\|$ does not vary substantially on short timescales.

One can consider the effect of these indices to be a confounder γ drawn at the start of every episode (orbit) affecting the state transitions through its effect on the underlying dynamics model.

We experiment with attempting to account for this confounder via augmenting the state vector with an additional metric γ estimating the norm of the disturbance torque T_{dist} at any given point in time. Recall that the state transition dynamics evolve as in (1). Rearranging (1), we obtain

$$T_{dist} = \dot{H} - T_{mag} + \sqrt{\frac{\mu}{a^3}}\hat{h} \times H, \tag{5}$$

$$\gamma = \left\| H(t_{k+1}) - \left[H(t_k) + \left(-T_{mag}(t_k) + \sqrt{\frac{\mu}{a^3}}\hat{h} \times H(t_k) \right) (t_{k+1} - t_k) \right] \right\| \approx \|T_{dist}\|(t_{k+1} - t_k) \tag{6}$$

In the algorithm denoted "CQC+C" in Table 2, we feed the metric γ into the model as a seventh element in the state vector.

4.7 Hyperparameter Optimization

Hyperparameter optimization is an important step in training a learning model and discovering its true capability. Hyperparameters play a major role in influencing the convergence properties of an algorithm and determine how well the model learns during the training process. In our experiments, we opt for a random search of hyperparameters instead of a grid search as it has been shown previously to be more efficient and effective [8].

We perform this search on the hyperparameters of the CQL [2] and TD3+BC [3] algorithms, where for a given search iteration, the model was trained on a particular dataset for 50 epochs and evaluated with FQE [5] for another 50 epochs on the datasets generated by u_{stoch} and by the uniform random policy. We used FQE instead of online evaluation in 42 to help ease the demands on computational resources when performing hyperparameter searches.

Through our initial experiments in training CQL [2] and TD3+BC [3], we found that a particular policy tended to provide good evaluation results on the simulator 42 when the FQE [5] values were high on both datasets, though the FQE estimates were generally overly optimistic compared to the simulated value functions. Therefore, we refer to the "best" hyperparameters as those that maximize the harmonic mean of the FQE [5] values using both datasets. We performed 40 such search iterations, choosing random hyperparameters in each search iteration. The optimal hyperparameters found for each algorithm by the random search can be seen in Table 1.

The hyperparameter tuning process was very computationally expensive, even when evaluated using FQE. Due to a lack of time and the computation associated with this process, not all the hyperparameters in Table 1 could be evaluated in simulation for a more accurate representation of their performance. Moreover, although the results in Table 1 are from an extensive search on selected hyperparameters, there is room for optimizing on other hyperparameters, such as the number of training epochs and the architecture of the actor-critic frameworks (the sizes of hidden layers, activation functions, depth of networks etc), which could be taken up in future work.

5 Selected Results

We ran many iterations of training with different datasets and algorithms, and then implemented the resulting policies in 42. Results for several of these policies are enumerated in Table 2.

The first datasets that we trained on were collected at the same rate as the simulator time step, 0.2 seconds. This resulted in excessively large data files, which were difficult to run training algorithms on. That said, the CQL algorithm using these datasets with 100 orbit samples (cases 7 and 8 in Table 2) outperformed the linear controller u_{nom} when the disturbance was ($c_{10.7} = 180, c_{ap} = 70$). Here, the number of orbit samples was chosen as 100 based on the amount of

			Hyperparameters					FQE	
			Learning rate of actor	Learning rate of critic	Learning rate of temperature parameter (CQL only)	Number of steps in TD Calculation	Discount factor	Evaluation on data generated through the purely stochastic policy	Evaluation on data generated through stochastic linear policy
U	CQL	A	0.0067	0.0010	9.39E-05	3	0.99	-16.2422	-1.7636
U	CQL	B	0.0029	0.0044	5.29E-05	3	0.99	-14.8702	-1.2491
S	CQL	A	8.20E-04	0.0039	4.62E-05	7	0.99	-1.4474	-1.227
S	CQL	B	0.0033	0.0099	1E-05	1	0.99	-13.7167	-1.7394
U	TD3+BC	A	0.0055	0.0019	-	1	0.99	-3.7406	-1.7609
U	TD3+BC	B	0.0047	0.0009	-	3	0.99	-10.8529	-2.0589
S	TD3+BC	A	0.0012	0.0076	-	1	0.99	-9.0338	-3.0962
S	TD3+BC	B	0.0043	3.71E-05	-	3	0.99	-1.5773	-1.7096

Table 1: Hyperparameters of CQL [2] and TD3+BC [3] found through the random search operation. Legend: U = data generated through uniform random policy, S = data generated through u_{stoch}, A = neural network architecture set A, B = neural network architecture set B

memory available in the computer used for training. We were surprised that this improvement was achieved using only this many orbits, as we had originally generated much more data. However, these cases underperformed compared to u_{nom} at the other tested values of $(c_{10.7}, c_{ap})$. This is likely because $(c_{10.7} = 180, c_{ap} = 70)$ was near the average of the $(c_{10.7}, c_{ap})$ values under which training data was generated. Thus, these policies prove that it is possible to improve upon u_{nom} with reinforcement learning, but the policies generated are not yet robust enough to be useful.

Seeing these issues, we generated new training datasets with three changes. First, we sampled less frequently at only 5 second intervals. This seemed appropriate since the outputs of the policy u_{nom} do not vary rapidly on that time scale. Second, instead of setting the $(c_{10.7}, c_{ap})$ values at the beginning of each orbit, we varied these parameters every 5 minutes of simulated time. We hoped that this might make the policies more robust to variations in $(c_{10.7}, c_{ap})$. Third, we changed the training policies to be more stochastic (see Section 4.2 for a description of the policies), hoping that this might result in discovery of more optimal control actions. These changes resulted in several learned policies that performed better than u_{nom} in one or more of the disturbance environments, and three policies (cases 14, 20, and 21) that performed better than u_{nom} in all three disturbance environments. However, the performance of cases 11-21 varied drastically between each other, so there are still unanswered questions regarding what is the best way to generate training datasets.

At the same time, recognizing the importance of the unknown parameters $(c_{10.7}, c_{ap})$, we added the additional state variable γ representing the disturbance magnitude. Case 19 in Table 2 documents the results of this case. Unfortunately, the addition of the metric γ in (6) to approximate T_{dist} seemed to neither hurt nor help the learning of an effective control law. We theorize that this could be due to the fact that T_{dist} varies with the CubeSat's position in the orbit as well as the $F_{10.7}$ and Ap-indices. As the CubeSat's position in the orbit is already included in the state vector, our metric of T_{dist} is heavily nonlinearly correlated with the three states comprising the CubeSat's position in the orbit. It is possible that this impeded learning the relationship between the $F_{10.7}$ and Ap-indices (as observed through γ) and the appropriate actions to take for each value of $(c_{10.7}, c_{ap})$.

Seeing how the best performing case on nominal disturbances in the initial dataset came from the "approximately constant" policy, we then generated an additional training data set using this

Case	Dataset	Sampling Frequency	F10.7, AP Indices	# Orbits	Algorithm	# Epochs	Neural Net	R_1	R_2	R_3	R
					$u = 0$			-0.1033	-0.1535	-0.2126	-0.1565
					u_{nom}			-0.0245	-0.0364	-0.0517	-0.0375
1	P	0.2 s	Fixed	4	COMBO	5	A	-0.0965	-0.1467	-0.2058	-0.1497
2	C	0.2 s	Fixed	4	COMBO	5	A	-0.1009	-0.1511	-0.2102	-0.1541
3	P	0.2 s	Fixed	4	CQL	5	A	-0.0898	-0.1401	-0.1993	-0.1431
4	C	0.2 s	Fixed	4	CQL	5	A	-0.0222	-0.0655	-0.1246	-0.0708
5	P	0.2 s	Fixed	4	TD3+BC	5	A	-0.0280	-0.0748	-0.1575	-0.0868
6	C	0.2 s	Fixed	4	TD3+BC	5	A	-0.0346	-0.0827	-0.1257	-0.0810
7	P	0.2 s	Fixed	100	CQL	10	A	-0.0339	-0.0156	-0.0719	-0.0405
8	C	0.2 s	Fixed	100	CQL	10	A	-0.0363	-0.0127	-0.0653	-0.0831
9	P+C	0.2 s	Fixed	50+50	CQL	15	B	-0.0324	-0.0581	-0.0940	-0.0615
10	P+C	0.2 s	Fixed	100+100	CQL	15	A	-0.0142	-0.0440	-0.0822	-0.0468
11	U	5 s	Varying	2000	CQL	40	A	-0.0761	-0.1238	-0.1800	-0.1266
12	S	5 s	Varying	2000	CQL	20	A	-0.0171	-0.0310	-0.0531	-0.0337
13	S	5 s	Varying	2000	CQL	40	A	-0.0721	-0.0287	-0.0303	-0.0437
14	S	5 s	Varying	2000[a]	CQL	40	A	-0.0183	-0.0306	-0.0513	-0.0334
15	S	5 s	Varying	2000	CQL	40	B	-0.1706	-0.2201	-0.2789	-0.2232
16	S	5 s	Varying	2000	TD3+BC	40	A	-0.0243	-0.0381	-0.0596	-0.0407
17	S	5 s	Varying	2000	TD3+BC	40	B	-0.1737	-0.2215	-0.2781	-0.2244
18	C	5 s	Fixed	2000	CQL	40	A	-0.1626	-0.2104	-0.2737	-0.2156
19	S	5 s	Varying	2000	CQL+C	40	A	-0.0200	-0.0354	-0.0590	-0.0381
20	S2	5 s	Varying	2000	CQL	40	A	-0.0186	-0.0305	-0.0505	-0.0332
21	S	5 s	Varying	2000	CQL	40	C	-0.0204	-0.0324	-0.0515	-0.0348

Table 2: Legend: C = constant, P = u_{nom}, S = u_{stoch}, S2 = u_{stoch} for two orbits, U = uniform random, CQL+C = CQL with Confounder, R_1 = low disturbances, R_2 = nominal disturbances, R_3 = high disturbances. The best performing simulations are highlighted in yellow.

[a]The training for this policy used a different set of data than the prior row

policy with 5 second sampling frequency, which was run as case 18 in Table 2. However, case 18 performed terribly. We are not sure why this was the case, and future work should study whether this was the result of the differing sampling intervals or some other factor. Next, noting how the momentum curve tends to diverge at the end of the orbit (e.g. see Fig 4), we trained over data lasting two whole orbits, and achieved minor policy improvement in case 20, though this was very similar to case 14 which was trained on the first orbit of each run in the same dataset. Most of the experiments were done under neural net architecture A, as architecture B generally performed poorly in Table 2. The updates in architecture C fixed the most common errors with architecture B, and performed comparably to the original architecture A, though further testing is needed to evaluate the robustness of architecture C.

Next, we note some common behaviors for the state trajectories observed under the trained policies. First, Fig. 3 shows a plot of H under u_{nom} for several orbits. Note how the momentum on the x axis increases for the first two orbits, and then settles nearly into a periodic curve (the aperiodic variations are due to irregularities in the Earth's magnetic field). Recall how we only trained our learned policies over single orbit datasets. If Fig 3 is restricted to a single orbit timescale, then it will appear that the x axis momentum is diverging at the end of the first orbit. This is also observed in the neural net policy in Fig. 4, which shows the most common qualitative behavior of the satellite momentum across all the neural net policies. Recall from Section 3 that this is the expected behavior for this particular CubeSat hardware. However, some policies would over-correct for this positive x momentum, such as shown in Fig. 5. This policy prevented the apparent divergence of the x axis momentum (at the same $(c_{10.7}, c_{ap})$ values as Figs. 3-4), but still resulted in a low reward (3) because the momentum was not driven to zero. This was commonly a reason for achieving low rewards under the $(c_{10.7} = 150, c_{ap} = 50)$ disturbance indices. Qualitatively, Figs 4-5 show the most common behaviors for the momentum curves from the learned policies. Certain policies also resulted in actions similar to under $u = 0$ for unknown reasons, which we suspect originated from coding bugs. Lastly, Fig. 6 shows the control inputs during a single orbit run of u_{nom} and a run of the best performing policy on average. Note how the best policy only makes small modifications to the actions taken under u_{nom} to be slightly more aggressive in its control inputs, though it outperforms u_{nom} on all three cases and on average. This may be because u_{nom} is nearly optimal, or because case 20 was trained using data from u_{stoch}, which is similar to u_{nom}.

6 Conclusion

Despite the difficulty in the problem caused by the particularly challenging hardware design for the CubeSat, we've successfully managed to learn multiple control laws offline that outperform a baseline linear control law, both on average and in all three cases. This demonstrably showcases the performance of reinforcement learning methods in the application of CubeSat attitude control.

However, questions remain about the stability of the learned control laws on episodes longer than a single orbit, as the momentum curves appear to diverge at the end of each dataset. Still, as the policies were only learned on single orbit datasets, we hope and conjecture that increasing the amount of training data and the length of each episode in future work will improve the performance of our learned control laws. Additionally, the CubeSat was also kept in the same orientation relative to the Sun, and was also started from the same position relative to the Sun in our experiments. As such future work should also include learning control laws in situations where these are not fixed.

The performance of the learned control laws can be further improved when we note that these control laws were learned offline, with no online fine-tuning. Implementing online fine-tuning of the learned control law with online RL methods during deployment will likely further improve the

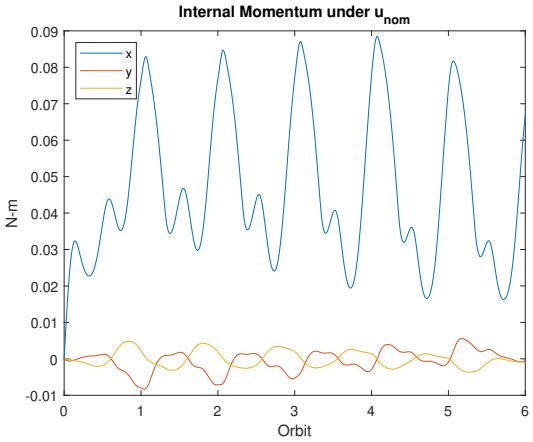

Figure 3: Momentum curve under u_{nom}

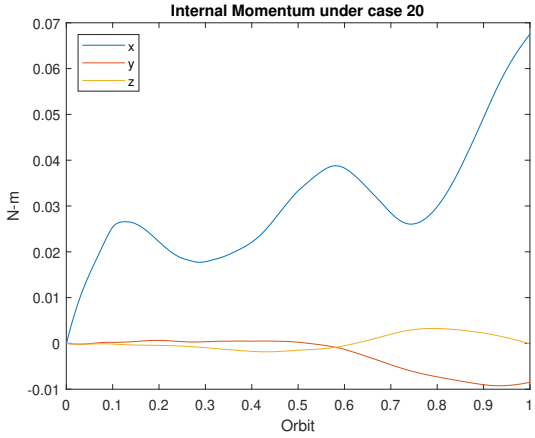

Figure 4: Momentum curve under the best performing controller tested

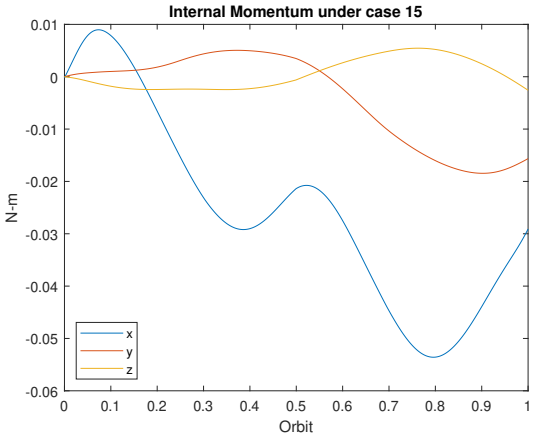

Figure 5: Momentum curve under a controller that overcompensates for the disturbance

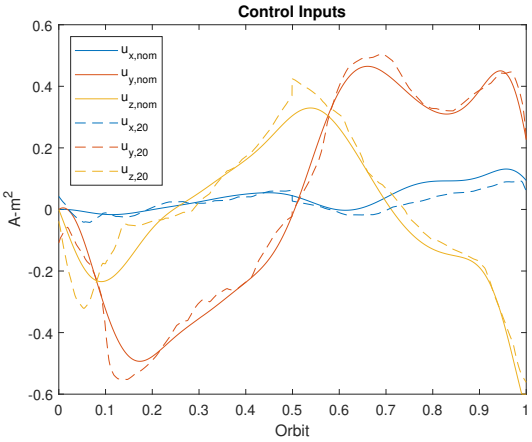

Figure 6: Control inputs during the cases in Figs. 3-4

Note that all of Figs. 3-6 are for the choice $(c_{10.7} = 180, c_{ap} = 70)$.

performance of the control laws while allowing them to be adaptive to changing conditions. One could also easily implement ensemble control laws that switch between a learned control law and a deterministic (linear) control law depending on whether the current state had been seen during the training data or not.

Other future work can include taking 1) another attempt at probabilistic model-based methods like COMBO with larger datasets to learn models of the underlying dynamics, though a much more sample-efficient method that is likelier to succeed is to instead 2) incorporate our knowledge of the dynamics given an (interval or point) estimate of the unknown parameters $(c_{10.7}, c_{ap})$. One could also 3) learn estimates of the unknown parameters $(c_{10.7}, c_{ap})$ via supervised learning methods instead of relying on the metric γ, 4) explore other neural network architectures for control policies that may yield better performance, and 5) perform a more extensive hyperparameter search with a more accurate evaluation scheme to find optimal hyperparameters that do well in simulation.

References

[1] NASA, "42: A Comprehensive General-Purpose Simulation of Attitude and Trajectory Dynamics and Control of Multiple Spacecraft Composed of Multiple Rigid or Flexible Bodies." `https://software.nasa.gov/software/GSC-16720-1`, 2015. Github Repository: `https://github.com/ericstoneking/42`.

[2] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[3] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[4] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, "COMBO: conservative offline model-based policy optimization," *CoRR*, vol. abs/2102.08363, 2021.

[5] H. Le, C. Voloshin, and Y. Yue, "Batch policy learning under constraints," in *International Conference on Machine Learning*, pp. 3703–3712, PMLR, 2019.

[6] M. I. Takuma Seno, "d3rlpy: An offline deep reinforcement library," in *NeurIPS 2021 Offline Reinforcement Learning Workshop*, December 2021.

[7] S. Sinha, H. Bharadhwaj, A. Srinivas, and A. Garg, "D2RL: deep dense architectures in reinforcement learning," *CoRR*, vol. abs/2010.09163, 2020.

[8] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.