Satellite Trajectory Optimization under Thruster Degradation

Keenan Albee, Hailee Hettrick, and Oliver Jia-Richards

Abstract— **The Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES) platform has been onboard the International Space Station for 13 years. Due to this extended usage, thrusters on the satellites have failed, resulting in underactuation. This project explores satellite trajectory optimization of an underactuated satellite in a 2D plane. A normally underactuated satellite is considered as the base model with further complications introduced by failing one or more thrusters. Three trajectory optimization methods are considered: direct collocation with GPOPS-II, direct transcription, and LQR-RRT*. All three methods are tasked with moving the satellite between two stationary positions. The resulting trajectory and control input are then compared along with the overall computational time in order to determine which method to use for implementation on an actual satellite.**

I. INTRODUCTION

Handling satellite thruster degradation is motivated by a real issue the Space Systems Laboratory (SSL) has experienced with the Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES) platform aboard the International Space Station (ISS). Each satellite has 12 CO_2 thrusters that provide full contrabillity in the 6 degree-of-freedom environment. SPHERES has been operating on station for 13 years and has run 110 test sessions. As a consequence of this extended usage, several thrusters have become degraded, resulting in a substantial decrease in performance and requiring a software solution to continue test sessions.

This paper documents the development and comparison of three different trajectory optimization methods: collocation with GPOPS-II, direct transcription, and LQR-RRT*. The latter is an optimizing sampling-based planner while the two former ultimately rely on formulation of the trajectory optimization as a non-linear program (NLP) using a discrete set of decision variables.

II. BACKGROUND

A. GPOPS-II

Direct collocation uses polynomial splines to represent the state and control variables of the trajectory it optimizes. GPOPS-II is a commercially available solver that uses Gaussian quadrature collocation with an adaptive mesh. The essential constraint utilized for direct collocation is that the derivative of the spline at the collocation points is equivalent to the dynamics [1], [2]. For the problem in question in this project, GPOPS-II was used to determine the "true" optimal trajectory for comparison with the results of direct transcription and LQR-RRT*.

B. Direct Transcription

Direct transcription discretizes the dynamics of the system and solves for the state and control input at each time step in order to minimize the overall fuel usage while satisfying the dynamic constraints as well as any other state and control constraints.

C. LQR-RRT*

LQR-RRT* is an extension of the optimizing sampling-based planner RRT* to systems with underactuated dynamics. RRT*, proposed in 2011 as an asymptotically optimal version of RRT, relies on the ability to determine the cost-to-go for use in its `rewire` step and optimally solve the two-point boundary value problem in `steer`, a challenge for dynamical systems and especially those with "complicated" dynamics [3]. LQR-RRT* uses an infinite horizon LQR cost-to-go approximation to quickly estimate cost-to-go and effectively guide the addition of nodes to the tree [4]. Applied to this problem, LQR-RRT* offers the explicit evaluation of constraints of samping-based planning paired with a cost-to-go heuristic to allow for probabilistic optimality for the constrained, underactuated dynamics.

III. PROBLEM STATEMENT

A simplified model of a SPHERES satellite is used to test the three different trajectory optimization methods. Figure 1 shows the simplified model in both its nominal state, with all four thrusters working, and in a failed state, where one thruster is broken. In both cases the satellite is underactuated as no horizontal thrust can be produced, and therefore the satellite cannot instantaneously accelerate in the horizontal direction. However, in the nominal state, the satellite is able to produce pure torques on its body while in the failed state the satellite can only produce a pure torque in one direction. This makes the trajectory optimization problem more complex especially if the satellite has to turn left (in the scenario illustrated by Figure 1) in order to reach its goal.

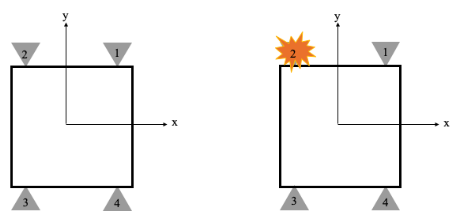

Fig. 1. Simplified SPHERES model with working (left) and failed thrusters (right).

Minimum fuel trajectories to move between an initial position and goal position were generated for the system with no thruster failure, with one thruster failure, and with two thruster failure.

The state vector of the fully actuated satellite is

$$\vec{q} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ h \\ \omega \end{bmatrix} \tag{1}$$

which has the dynamics described by

$$\dot{\vec{q}} = f(\vec{q}, \vec{u}) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ (u_1 + u_2 - u_3 - u_4)\frac{\sin(h)}{m} \\ (u_1 + u_2 - u_3 - u_4)\frac{\cos(h)}{m} \\ \omega \\ (-u_1 + u_2 - u_3 + u_4)\frac{l}{2I} \end{bmatrix}. \tag{2}$$

To mimic a failed thruster, u_i is set to 0 in the dynamics. Each trajectory optimization method adhered to the following state and input constraints.

$$0 \text{ N} \leq u_i(t) \leq 0.1 \text{ N} \tag{3a}$$
$$-50 \text{ m} \leq x(t) \leq 50 \text{ m} \tag{3b}$$
$$-50 \text{ m} \leq y(t) \leq 50 \text{ m} \tag{3c}$$
$$-50\frac{\text{m}}{\text{s}} \leq \dot{x}(t) \leq 50\frac{\text{m}}{\text{s}} \tag{3d}$$
$$-50\frac{\text{m}}{\text{s}} \leq \dot{y}(t) \leq 50\frac{\text{m}}{\text{s}} \tag{3e}$$
$$-5\pi \text{ rad} \leq h(t) \leq 5\pi \text{ rad} \tag{3f}$$
$$-\pi\frac{\text{rad}}{\text{s}} \leq \omega(t) \leq \pi\frac{\text{rad}}{\text{s}} \tag{3g}$$

Additionally, the spacecraft mass is specified as $m = 4.1$ kg, the side length is $l = 0.25$ m, and the rotational inertia is $I = \frac{1}{6}ml^2$ kg m^2. These parameters, dynamics, and constraints were upheld for each trajectory optimization method.

IV. Approach

A. GPOPS-II

GPOPS-II was used as the initial trajectory optimization method. The optimization setup is

$$\min_{\vec{q}(t)} \int_0^{t_f} \vec{u}^T \vec{u} \text{ dt} \tag{4}$$
$$\text{such that } \dot{\vec{q}}(t) = f(\vec{q}, \vec{u}) \tag{5}$$

with state and control constraints from Equation 3 allocated as bounds on the problem. Additionally, the final time was fixed at 60 seconds. The optimization was framed in the standard setup format for GPOPS-II allowing for quick evaluation of the optimal trajectory.

B. Direct Transcription

Direct transcription was used via Pydrake's MathematicalProgram with 50 knot points ($N = 50$). The optimization problem is

$$\min_{\vec{q}_1,\ldots,\vec{q}_N,\vec{u}_0,\ldots\vec{u}_{N-1}} \sum_{n=0}^{N-1} g(\vec{q}, \vec{u})\text{dt} \tag{6}$$
$$\text{such that } \vec{q}_{n+1} = \vec{q}_n + f(\vec{q}, \vec{u})\text{dt} \quad \forall\, n \in [0, N-1] \tag{7}$$
$$g(\vec{q}, \vec{u}) = \vec{u}^T \vec{u} \tag{8}$$
$$f(\vec{q}, \vec{u}) = \dot{\vec{q}} \tag{9}$$

Linear constraints were placed on each state and control input that imposed the relationships described in Equation 3. Additionally, a linear constraint was placed on the first state in time to ensure it was equivalent to the initial state of the trajectory. A constraint was placed on the final time of the trajectory such that the trajectory had to take the user-defined amount of time. Linear constraints were also used to enforce the dynamics of the system. To push the dynamics to the desired goal state, a constraint was placed on the final state error so that the error had to be within 0.001. Lastly, a quadratic cost was used on the control history such that the optimization sought to minimize the control effort (Equation 8).

C. LQR-RRT*

To develop a full LQR-RRT* algorithm, a kino-RRT-Euclidean algorithm was first constructed. This kino-RRT-Euclidean algorithm used Euclidean distance as its cost metric and made use of randomized inputs in its `steer` function to forward propagate the satellite dynamics. Due to controllability issues with the full LQR-RRT* algorithm, a Guided-Kino-RRT algorithm was developed using the LQR cost metric and a motion primitive steering function. This section will discuss the approach for each of these algorithms in turn.

The kino-RRT-Euclidean algorithm closely resembles RRT but with forward propagation of the dynamics, adhering to the following process:

1) Find a random sample and its cost in terms of Euclidean distance to the goal
2) Determine the nearest node on the existing tree
3) Steer the system towards the random state by selecting the input resulting in the lowest Euclidean distance from a set of randomized inputs
4) Check if the new node was within the bounds of the problem
5) Append the node to the tree
6) Determine if the new node satisfies the goal state tolerance
7) If the goal tolerance was not met, the algorithm returns to step 1

In contrast, the Guided-kino-RRT and LQR-RRT* variants of the algorithm utilized LQR cost-to-go as a distance metric, and the final LQR-RRT* algorithm also used the optimal LQR policy as the control input in the `steer` function. Perez et al. provides a concise explanation of LQR that is replicated here for completion [4]. The dynamics of the system in question are given by Equation 2. To approximately optimize a quadratic cost about the random sample $(\vec{q}_{rand}, \vec{u}_0)$, a first-order Taylor series expansion about the random sample yields

$$\dot{\vec{q}} = f(\vec{q}_{rand}, \vec{u}_0) + \frac{\partial f(\vec{q}_{rand}, \vec{u}_0)}{\partial \vec{q}}(\vec{q} - \vec{q}_{rand}) + ... \\ \frac{\partial f(\vec{q}_{rand}, \vec{u}_0)}{\partial \vec{u}}(\vec{u} - \vec{u}_0) \tag{10}$$

This expression simplifies to

$$\dot{\bar{\vec{q}}} \approx A(\vec{q}_{rand}, \vec{u}_0)\bar{\vec{q}} + B(\vec{q}_{rand}, \vec{u}_0)\bar{\vec{u}} \tag{11}$$

where $\bar{\vec{q}} = \vec{q}_{new} - \vec{q}_{rand}$ and $\vec{q}_{new}$ is the state yielded from an Euler integration of the satellite dynamics, which is determined in the `steer` function. The LQR cost function is of the form

$$J = \int_0^\infty \left(\bar{\vec{q}}^T Q \bar{\vec{q}} + \bar{\vec{u}}^T R \bar{\vec{u}}\right) \text{ dt.} \tag{12}$$

The optimal LQR policy is determined by first solving the algebraic Ricatti equation for P.

$$0 = PA + A^T P + Q - PBR^{-1}B^T P \tag{13}$$

Once P has been determined, the optimal value cost function is simply

$$V(\bar{\vec{q}}) = \bar{\vec{q}}^T P \bar{\vec{q}} \tag{14}$$

and the optimal policy is

$$\pi^*(\bar{\vec{q}}) = -K\bar{\vec{q}} \tag{15}$$
$$\text{where } K = R^{-1}B^T P.$$

As mentioned previously, LQR-RRT* relies on using the infinite-horizon LQR described above to create a locally optimal control policy for `steer`, and to create distance metrics used in `near`, `nearest`, `chooseParent`, and `rewire`. The LQR-RRT* algorithm is summarized in Algorithm 1. As described above, the LQR policy is found by linearization about the desired goal point, either $\vec{q}_{rand}$, $\vec{q}_{new}$, or $\vec{q}_{near}$, depending on when `steer` is needed. The `rewire` and `chooseParent` portions of the algorithm are illustrated in Figure 2.

LQR-RRT*, while successfully implemented, posed a few challenges detailed in the results section which led to poor solutions that rarely approached the goal. In response, a kino-RRT with LQR-supplied distance metric was used and a custom `steer` function was designed. In this case, rather than executing the LQR optimal policy derived from a linearized point, the steering module is provided with a set of "sensible" inputs to apply to the system, e.g. thruster activations for spinning, translating along y, activating single thrusters, etc. These inputs are applied at varying thrust levels and forward propagated over a set time step. The result that is closest to the desired point, using the LQR cost, is selected as $\vec{q}_{new}$.

Algorithm 1: LQR-RRT*

```
for $i = 1, ..., N$ do
    $\vec{q}_{\text{rand}} \leftarrow$ sample;
    $\vec{q}_{\text{nearest}} \leftarrow$ LQRNearest($V$, $\vec{q}_{\text{rand}}$);
    $\vec{q}_{\text{new}} \leftarrow$ LQRSteer($\vec{q}_{\text{nearest}}$, $\vec{q}_{\text{rand}}$);
    $\mathcal{X}_{\text{near}} \leftarrow$ LQRNear($\vec{q}_{\text{new}}$, $V$);
    for $\vec{q}_{near} \in \mathcal{X}_{near}$ do
        if cost($\sigma_{near,new}$) $< minCost$ then
            $\vec{q}_{min} \leftarrow \vec{q}_{near}; \sigma_{min} \leftarrow \sigma_{near,new}$;
        end
    end
    if inBounds($\sigma_{min}$) then
        $V \leftarrow V \cup \{\vec{q}_{\text{new}}\}$;
        $E \leftarrow V \cup \{(\vec{q}_{\text{min}}, \vec{q}_{\text{new}})\}$;
        rewire($V, E, \mathcal{X}_{near}, \vec{q}_{new}$);
    end
end
return $G = (V, E)$;
```

The Guided-Kino-RRT algorithm is summarized in Algorithm 2, which uses an LQR cost-to-go distance metric and a motion primitive steering function. Iterating over multiple input values for forward integration, the motion primitives used were:

- Single-thruster spin-translation
- Two-thruster translation (pure force pair)
- Two-thruster spin (pure torque pair)
- Three-thruster spin/translation

Algorithm 2: Guided-Kino-RRT

```
while $\vec{q}_{err} > tol$ do
    $\vec{q}_{\text{rand}} \leftarrow$ sample;
    $\vec{q}_{\text{nearest}} \leftarrow$ nearest($V$, $\vec{q}_{\text{rand}}$);
    $\vec{q}_{\text{new}} \leftarrow$ guidedSteer($\vec{q}_{\text{nearest}}$, $\vec{q}_{\text{rand}}$);
    if inBounds($\vec{q}_{new}$) then
        $V \leftarrow V \cup \{\vec{q}_{\text{new}}\}$;
        $E \leftarrow V \cup \{(\vec{q}_{\text{near}}, \vec{q}_{\text{new}})\}$;
    end
end
return $G = (V, E)$;
```

V. RESULTS & DISCUSSION

A. GPOPS-II & Direct Transcription

Three cases were tested using both the direct transcription and collocation methods: 1) no thruster failure, 2) one thruster failure, and 3) two thruster failure. For each case and optimization method, the initial state and final state were

$$\vec{q}(t=0) = \begin{bmatrix} 10 \\ 10 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \vec{q}(t=60) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{16}$$

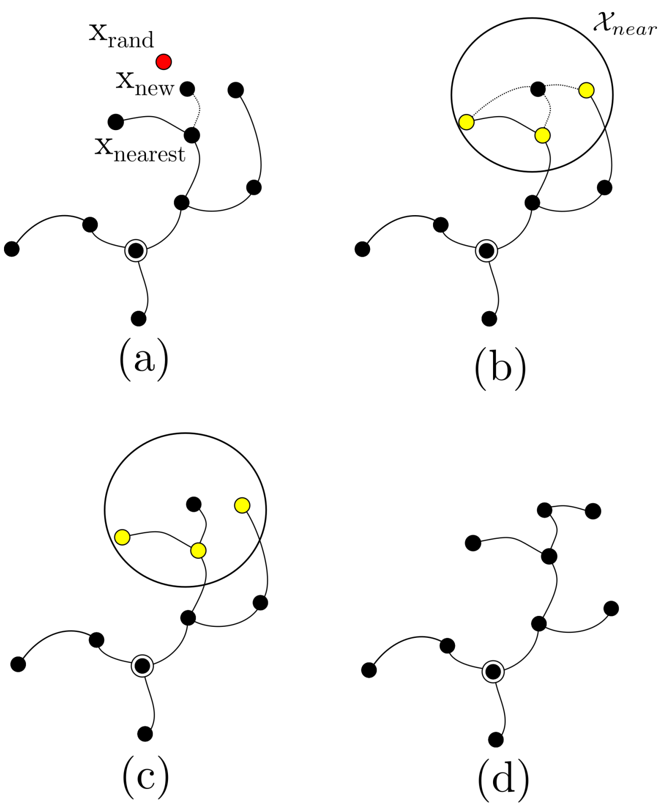

Fig. 2. LQR-RRT* introduces modifications to RRT* to account for differential constraints. Assuming an LQR-compatible problem (quadratic cost and linear dynamics), the selection of `LQRNearest` in (a) is performed using infinite-horizon LQR's cost-to-go to x_{rand}. The optimal input from this policy is followed to arrive at x_{new}. `Nearest` and `Rewire` steps used in (b) - (d) are performed using infinite-horizon LQR cost-to-go, and proceed as nominal RRT*.

where the trajectory is constrained to take 60 seconds to complete.

1) No Thruster Failure: First, each optimization method was compared in the case of no thruster failure. Figure 3 shows a comparison between GPOPS-II and direct transcription of the resulting trajectory, net force, and net torque. We can see minor visual differences in the trajectory between the two solutions. The GPOPS-II solution had a total cost of 0.1104 N^2 while the direct transcription solution had a total cost of 0.1358 N^2 giving a 23% decrease in optimally. The difference in cost can be attributed to the discrete control input of the direct transcription solution. In terms of computation time, GPOPS-II found the optimal solution on average 2.87 seconds while direct transcription took on average 5.28 seconds.

2) One Thruster Failure: Next, each optimization method was compared in the case of one thruster failure (thruster 1 shown in Figure 1). Figure 4 shows a comparison between GPOPS-II and direct transcription of the resulting trajectory, net force, and net torque. The GPOPS-II solution had a total cost of 0.1314 N^2 while the direct transcription solution had a total cost of 0.1603 N^2 giving a 22% decrease in

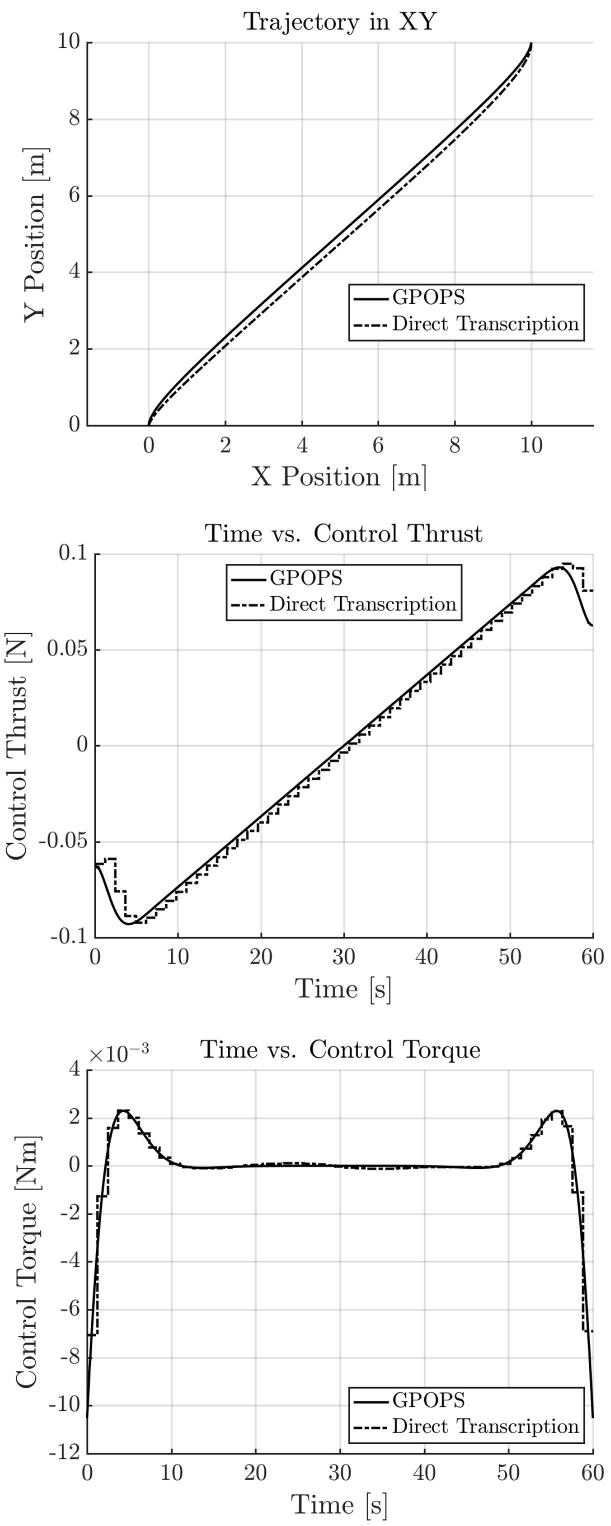

Fig. 3. Comparison between GPOPS-II and direct transcription of trajectory, net force, and net torque for no thruster failures.

optimally. In terms of computation time, GPOPS-II found the optimal solution on average 1.60 seconds while direct transcription took on average 3.77 seconds.

3) Two Thruster Failure: Lastly, each optimization method was compared in the case of two thruster failure (thrusters 1 and 2 in Figure 1). Figure 5 shows a comparison between GPOPS-II and direct transcription of the resulting trajectory, net force, and net torque. The GPOPS-II solution had a total cost of 0.1549 N^2 while the direct transcription solution had a total cost of 0.2005 N^2 giving a 29% decrease in optimally. In terms of computation time, GPOPS-II found the optimal solution on average 1.59 seconds while direct transcription took on average 2.83 seconds.

*B. LQR-RRT**

Only one case was tested with LQR-RRT* variants – no thruster failure on the nominally underactautated satellite – due to difficulties getting the algorithm to work correctly with our system. Figure 6 shows three exploration trees produced by running Guided-Kino-RRT with the red trajectory representing the trajectory that reached closest to the goal node. The tree explores the space fairly well, reaching the goal and terminating at a distance metric tolerance of 0.1. The presented paths are also clearly sub-optimal – while some paths show intuitively optimal motion, others contain wasteful looping, for instance. With a more precise steering function, it would be reasonable to once again add in `rewire`, creating an optimizing variant.

One of the main contributors to the poor performance of LQR-RRT* – and the reason it was abandoned after implementation – is that the linearized system is not stabilizable when linearized about the desired state with zero control input. Any velocity perpendicular to the thrust axis cannot be attenuated, and therefore LQR cannot find a finite cost-to-go function. To circumvent this problem, the linearization was taken about an assumed control input, similar to a quadcopter altitude controller linearized in a gravitational field. Linearizing around an assumed control input does not properly represent our dynamics but does allow LQR-RRT* to work, but poorly – results rarely reached the goal. This indicates that LQR-RRT* is not a good choice for path planning for our chosen system. Guided-Kino-RRT, however, produced reasonable feasible trajectories in real-time (seconds). It is a potentially useful alternative, particularly in highly cluttered environments. We note the following drawbacks of LQR-RRT*:

- The infinite horizon LQR policy does *not* guarantee fixed final state
- Certain systems may have linearizations that are not stabilizable, and for which LQR policies cannot be found
- The shrinking ball for `near` search radius is not justified for non-Euclidean spaces, as it is in [3]
- Linearization around a non-fixed point is likely a poor approximation of the dynamics

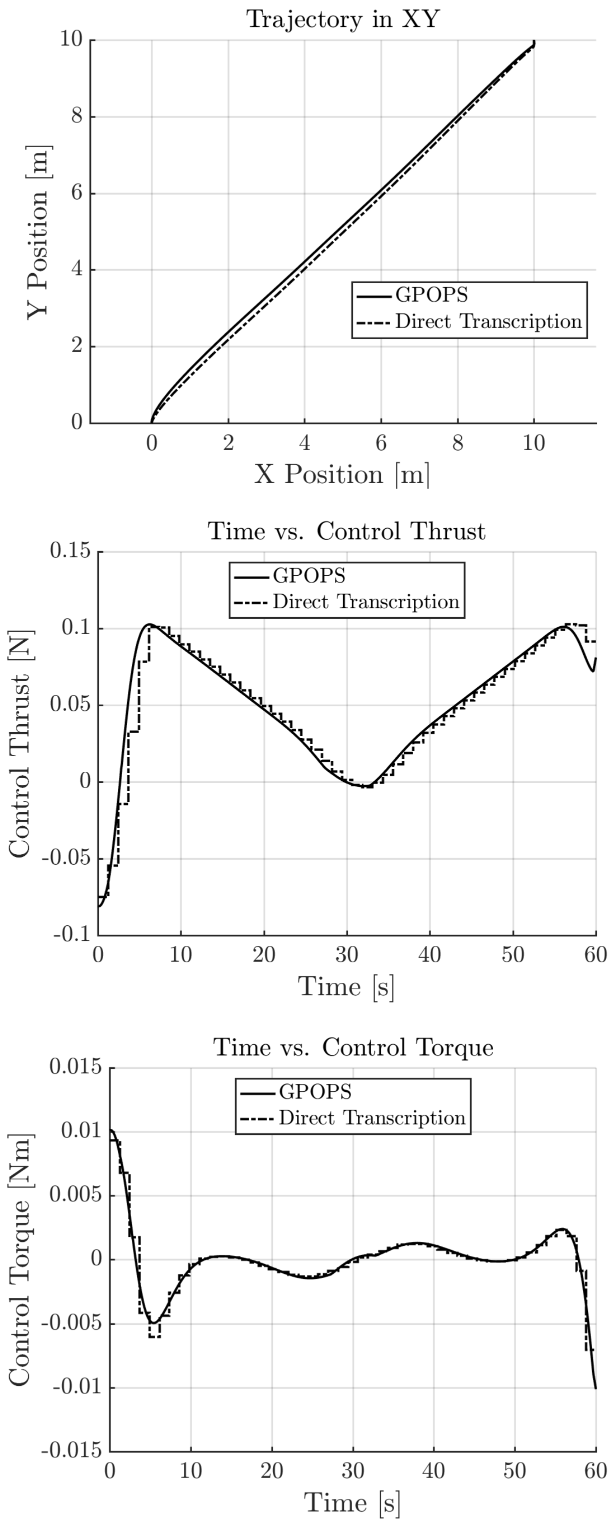

Fig. 4. Comparison between GPOPS-II and direct transcription of trajectory, net force, and net torque for one thruster failure.

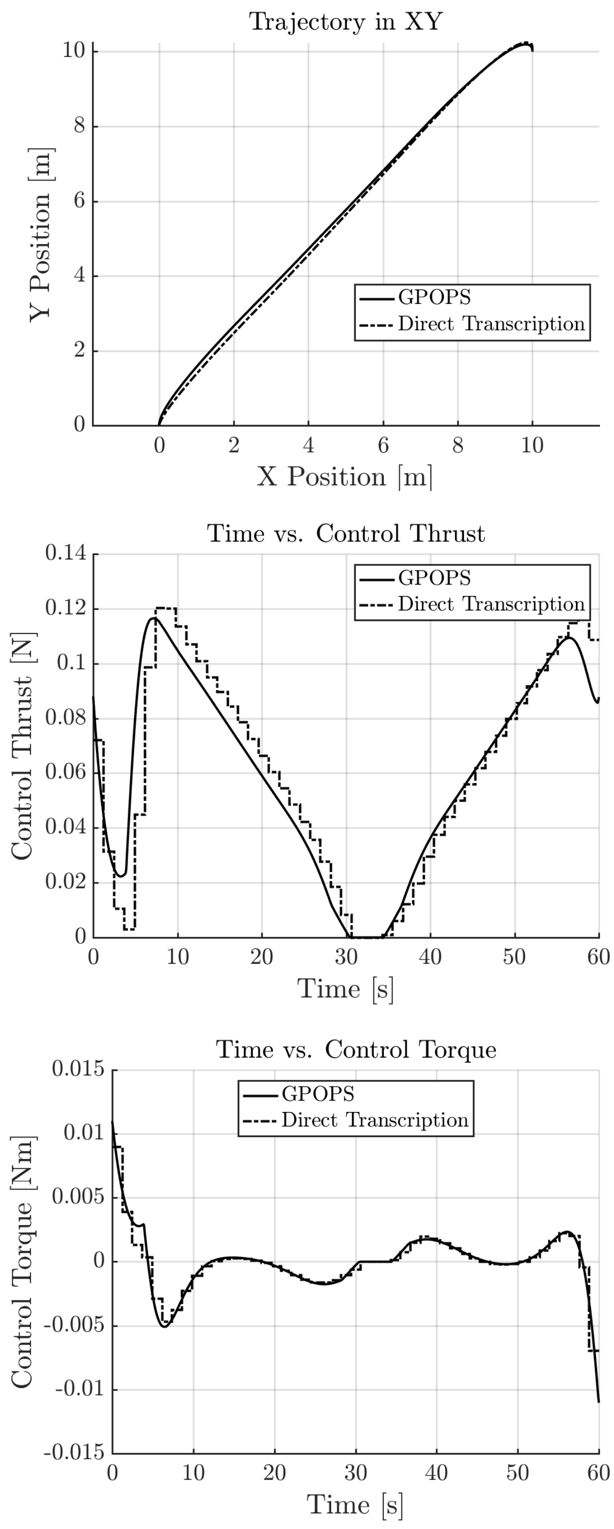

Fig. 5. Comparison between GPOPS-II and direct transcription of trajectory, net force, and net torque for two thruster failures.

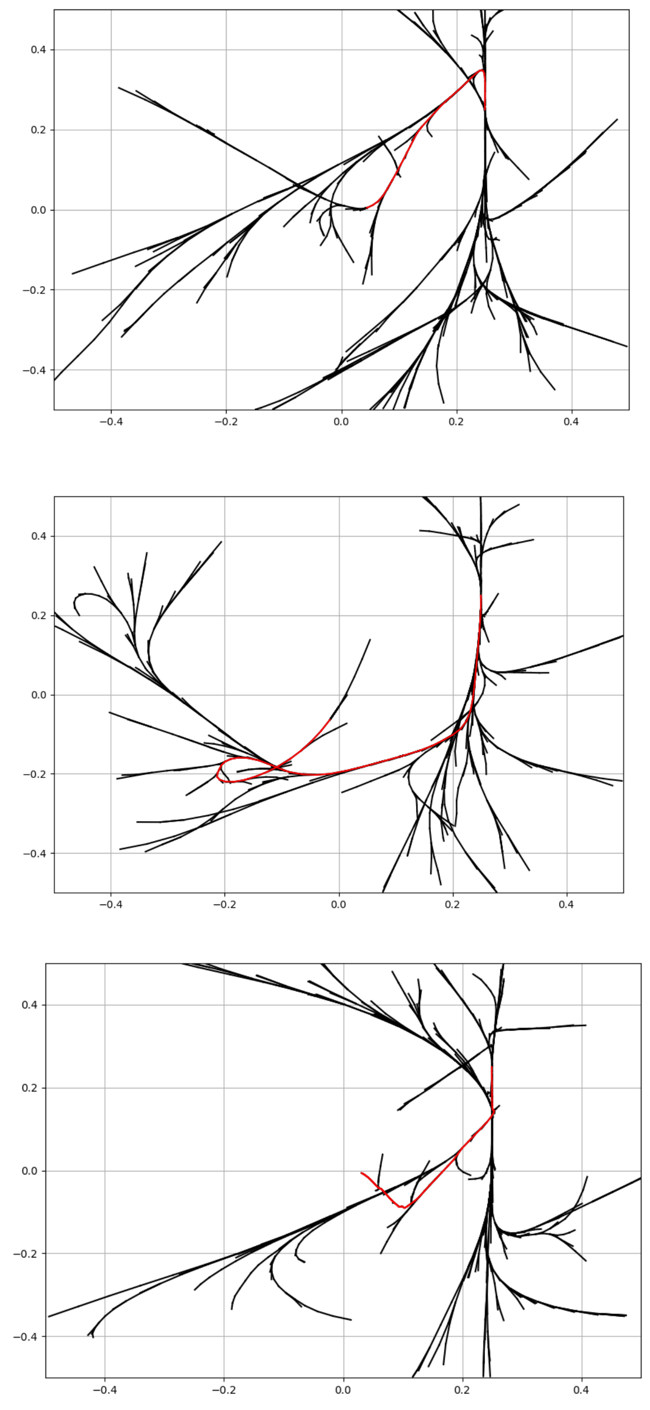

Fig. 6. Representative trajectories produced by Guided-Kino-RRT. Note the clearly sub-optimal looping behavior in the second image.

Animations of solutions from GPOPS-II, direct transcription, and kino-RRT-LQR can be found at https://bit.ly/2JPR1jk. The code used for this project can be found at https://bit.ly/2WRQtx5.

VI. CONCLUSION

Ultimately, the results of this project indicate that LQR-RRT* is not a reliable trajectory optimization method for a satellite of the aforementioned dynamics. However, direct transcription is promising and provides solutions that are comparable to more advanced methods (such as GPOPS-II) both in terms of the optimally and computation time. Sampling-based methods potentially have a place in environments where NLP-based solutions might fail, offering at least feasible solutions. Moving forward, for implementation on an actual spacecraft, direct transcription is the method of choice. Next steps for this research include modeling partially degraded thrusters rather than total failure, expanding the model to a 3D environment, adding obstacle avoidance, and implementing the trajectory optimization on the SPHERES hardware.

REFERENCES

[1] Charles R Hargraves and Stephen W Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.

[2] Russ Tedrake. Trajectory optimization. http://underactuated.csail.mit.edu/underactuated.html?chapter=trajopt, 2019.

[3] Sertac Karaman and Emilio Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *The International Journal of Robotics Research*, 2011.

[4] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *2012 IEEE International Conference on Robotics and Automation*, pages 2537–2542. IEEE, 2012.