SPACER(R) CRAFT RENDEZVOUS GUIDANCE IN CLUTTERED ENVIRONMENTS VIA ARTIFICIAL POTENTIAL FUNCTIONS AND REINFORCEMENT LEARNING

Brian Gaudet, Richard Linares, and Roberto Furfaro

The primary contribution of this work is to use Artificial Potential Functions (APF) for generating trajectories to be used as initial guesses for General Pseudospectral Optimal Control (or GPOPS). This work demonstrates dramatic speed up for GPOPS solution times, giving an average trajectory generation time of around 6 seconds. With this level of performance, the trajectory generation could occur on board the spacecraft based off of its current state estimate. In the type of scenarios that this algorithm is designed for (rendezvous, orbital transfer), this work can execute in near-real-time. This worked also improves the trajectory tracking controller performance, achieving continuous thrust fuel efficiency equal to the GPOPS optimal solution, and pulsed thrust fuel efficiency about 25% worse than the GPOPS optimal solution.

INTRODUCTION

Over the past few decades, rendezvous guidance problems have been steadily gaining importance especially in servicing space assets where autonomous rendezvous is needed to extend the lifetime of space systems and reduce the overall mission cost. Recently, fuel-efficient guidance algorithms that can implement time and space constraints have been developed. For example, Henshaw proposed an optimal control computational method where the cost function is a combination of fuel usage, control effort, time-of-flight and obstacle constraints. Although, the optimization is executed considering a 6-DOF spacecraft model, optimal control approaches are computationally expensive, sensitive to the initial guess and not feasible for on-board, real-time implementation. Richards et al. developed a fast and robust code for autonomous rendezvous in highly cluttered environments. However, real-time implementation may be limited by the simplifying assumptions (e.g. obstacles are static, no models for uncertainties and disturbance). More recently, Boyarko, et al. solved the problem of rendezvous with a tumbling target using optimal control, but in their work they state that the computational time is too long for real time trajectory generation.

Artificial Potential Function Guidance (APFG) algorithms, first proposed in (Reference 1) are a promising approach to rendezvous guidance. Such algorithms define an appropriate scalar function to describe an attractive potential field around the target vehicle. Obstacles and possible path constraints introduce regions of high repulsive potential around the undesired regions. The potential

---

*Co-Founder, DeepAnalytX LLC, 1130 Swall Meadows Rd, Bishop CA 93514.
†Charles Stark Draper Assistant Professor, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139. E-mail: linaresr@mit.edu, Senior Member AIAA.
‡Professor, Department of Systems & Industrial Engineering, University of Arizona, Tucson, AZ, 85721. Email: robertof@email.arizona.edu, Member AIAA.
takes the form of a velocity field that maps the spacecraft’s position to a target velocity. A velocity field tracking controller is then used to map the difference between the spacecraft’s velocity (as estimated by the navigation system) and the target velocity to a commanded thrust vector. More recently, Munoz, et. al.\(^2\) developed an adaptive APFG algorithm that provided an increase in fuel efficiency, which was later tested in hardware by Zapulla et al.\(^3\) Although an improvement in fuel efficiency was noted as compared to prior APFG algorithms, fuel efficiency was not compared to that obtainable using optimal control, and the authors note abnormal behavior when the spacecraft is in proximity to an obstacle. To date, the work with APFG algorithms has neglected to demonstrate performance with obstacles that are in relative motion with respect the rendezvous target.

![Figure 1. Sample GPOPS Trajectories (L) and non-adaptive APF Trajectories (R).](image)

This work we developed a 3-DOF APF guidance law implementation that was used for an initial solution to a rendezvous problem solved using optimal control. It turns out that the APF guidance law we developed for the initial optimal control solution was reasonably fuel efficient, with fuel efficiency only 20\% worse than optimal (see Table 1), as defined by the fuel efficiency obtained using the GPOPS\(^4\) optimal control package, and performed well in proximity to obstacles. For the case of a rendezvous maneuver with the spacecraft initially 1-km from the target, two obstacles, and a keep-out zone around the target, this translated into only a 300g difference in propellant usage. We attribute the performance gains over prior work to a novel choice of attractive and repulsive field functions that to our knowledge have not been previously used to implement an APF guidance law. Another factor behind the performance gains is our use of a non-linear velocity field-tracking controller. Importantly, the field parameters were static, given the possibility of additional performance gains using an adaptive approach. One issue we found for the case where the obstacles are in motion with respect to the target is that in rare cases, the keep-out zone around obstacles is slightly violated, and fuel efficiency was slightly reduced. This may be a common problem with APF guidance systems, as to our knowledge, work in this area has only addressed the case of static obstacles. Figure 1 gives a sample of 20 GPOPS and APF trajectories (not generated from the same set of initial conditions). Keep-out zones are represented as spheres, and obstacle motion is denoted by the thick colored lines. Note that the APF trajectories are less smooth; it is likely that the adaptive APF that will be developed in this work will look more like the GPOPS trajectories and have improved fuel efficiency.

There is clearly the potential to improve our APFG implementation by making the field adaptive, specifically by having the obstacle repulsive field amplitude depend on the relative position and
Table 1. Fuel Efficiency with Static and Dynamic Obstacles. This work develops a method that uses the APF trajectories to initialize GPOPS for real-time optimal control.

<table>
<thead>
<tr>
<th></th>
<th>Fuel Consumption (kg)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Static-Obstacles</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>Optimal (GPOPS)</td>
<td>2.8</td>
</tr>
<tr>
<td>Non-Adaptive APF</td>
<td>3.36</td>
</tr>
</tbody>
</table>

velocity of the spacecraft and obstacle, as opposed to position only. One method to learn an adaptive field that will achieve robust and fuel-efficient rendezvous is reinforcement learning (RL). In the RL framework, an agent learns through repeated interactions with a simulated or real environment how to successfully complete a task by mapping observations to actions. The environment functions as a simulator that initializes an episode by randomly generating an internal state, and passing an observation (a function of the state) to the agent. At each step of the episode, the agent uses the observation to generate an action and passes the action back to the environment, which then generates the next state and a scalar reward signal. The observation corresponding to the next state and reward signal are then passed to the agent. The environment terminates an episode when the agent completes the task or fails in some task-specific manner. The agent maps the observation to an action through a policy, and it is the goal of the agent to learn a policy that maximizes the expected sum of future discounted rewards over some distribution of trajectories.

**PROBLEM FORMULATION**

**Equations of Motion**

For our analysis, we consider a rendezvous problem where a deputy spacecraft flying in relative motion with respect to a chief spacecraft, is required to execute a continuous guidance command that drives the system from an initial position to rendezvous with the chief. More specifically, we are interested in numerically finding open-loop solutions that minimize a specified cost (e.g. fuel expenditure, time of flight, etc.). Here, we focus on the minimum-fuel constrained rendezvous problem, which is defined as follows:

**Constrained Minimum-Fuel Rendezvous Problem:** Find the thrust program that minimizes the following cost function:

$$\min_{t_f,T} \int_{t_0}^{t_f} \|T\|dt$$

subject to

$$x = f(x, T)$$

$$x(t_0) = x_{\text{start}}$$

$$x(t_f) = x_{\text{final}}$$

$$\psi_i(x(t), t) = 0, \quad i = 1, \ldots, N$$

Here $$\psi_i(x(t), t)$$ represent the path and Time-Of-Flight (TOF) constraints that are necessary to compute fuel-efficient paths for collision-avoidance with obstacles during autonomous rendezvous. For this study, we considered a Clohessy-Wiltshire (CW) model, which represents the linearized
dynamics of the relative motion for a chief spacecraft in circular orbit. Such assumption is not restrictive and can be generally removed for cases where such assumptions are violated (e.g. chief in a high elliptical orbit). The constrained minimum-fuel rendezvous problem does not have an analytical solution and must be solved numerically. The open-loop, fuel-optimal thrust program can be computed via optimal control software packages such as the General Pseudospectral Optimal Control Software (GPOPS). For the above problem formulation, the solution returned by GPOPS is a sequence of times and associated states in $\mathbb{R}^6$ (position, velocity), and a thrust vector in $\mathbb{R}^3$.

**POLICY GRADIENT REINFORCEMENT LEARNING**

In this work, we explored using reinforcement learning to design a trajectory-tracking controller. A brief introduction to reinforcement learning (RL) follows. In the RL framework, an agent learns through repeated interactions with an environment how to successfully complete a task by mapping observations to actions. The environment functions as a simulator that initializes an episode by randomly generating an internal state, and then based off of the actions received by the agent, generating the next state and a scalar reward signal. At each step of the episode, an observation is generated from the internal state, and given to the agent. The agent uses this observation to generate an action that is sent to the environment; the environment then uses the action and the current state to generate the next state and reward. The reward and the observation corresponding to the next state are then passed to the agent.

The environment can terminate an episode, with the termination signaled to the agent via a done signal. The termination could be due to the agent completing the task or the agent doing something counterproductive, like entering a keep-out zone. Each episode results in a trajectory defined by observations, actions, and rewards; a step in the trajectory beginning at time $t$ can be represented as $o_t, a_t, r_t$, where $o_t$ is the observation provided by the environment, $a_t$ the action taken by the agent using the observation, and $r_t$ the reward returned by the environment to the agent. The reward can be a function of both the next observation $o_{t+1}$ and the action $a_t$. In the case where we want to construct a potential-based reward function, the reward can also be a function of the current observation. In this work, we will assume a navigation system provides a state estimate to the guidance system; consequently, the observations will be the ground-truth states of the spacecraft and obstacles in the target-centered reference frame, potentially offset to model state estimation error.

Initially, the agent implements a random policy, which allows the agent to explore the state space and begin learning which actions are to be preferred given an observation. As the agent gains experience, the amount of exploration is decreased, allowing the agent to exploit this experience. For most applications (unless a stochastic policy is required), when the policy is deployed in the field, exploration is turned off, as exploration gets quite expensive using an actual spacecraft. The safe method of continuous learning in the field is to have the spacecraft send back telemetry data, which can be used to improve the environment’s dynamic model, and update the policy via simulated experience.

In the RL framework, we can define the value of being in a given state $V^\pi(s)$ as the sum of future rewards $r$, discounted using $\gamma$, expected if the agent starts in that state and follows some policy $\pi(s,a)$ that gives the probability of taking some action $a$ when in state $s$:

$$V^\pi(s_t) = E \left[ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} \right]$$  \hspace{1cm} (2)
With these definitions, the agent’s goal is to learn a policy that maximizes the expected value of initial states drawn from some initial state distribution.

One approach to learning the policy mapping observations to actions is the policy gradient method. The policy gradient method adjusts the policy’s parameters to increase the probability of actions that tend to give higher than expected rewards, and decrease the probability of actions that give lower than expected rewards. Specifically, over a given trajectory the advantage function maps the states $s_t$ encountered over the trajectory at time $t$ to the difference between: 1.) The actual discounted rewards obtained starting in state $s_t$ and following policy $\pi$ for the remainder of the trajectory and 2.) The expected value of being in state $s_t$, as estimated by the value function. The calculation of the advantage function is shown below, where $r(s,a)$ is a reward function returning the reward obtained from taking action $a$ in state $s$, and $\gamma$ is the discount rate.

$$A^\pi(s_t, a_t) = E \left[ \sum_{\tau=t}^n \gamma^{\tau-t} r(s_\tau, a_\tau) \right] - V^\pi(s_t) \quad (3)$$

We can define an objective function that tries to maximize the rewards expected over the trajectories $\tau$ induced by following policy $\pi$ over some distribution of initial conditions, where we approximate the expectation by Monte Carlo, i.e., by looking at the rewards that result over a sample of $N$ trajectories, as shown below.

$$J(\theta) \approx \frac{1}{N} \sum_{\tau} \sum_{t} \gamma^{\tau-t} r(s_\tau, a_\tau) \quad (4)$$

Our goal is then to find the parameter vector $\theta$ that maximizes this objective function. It can be shown that we can maximize the above objective function by following the policy gradient given below, where again the expectation is approximated by Monte Carlo sampling over a batch of $N$ trajectories. Here $s(\tau, t)$ denotes the state following trajectory $\tau$ at time $t$.

$$\nabla J(\theta) \approx \frac{1}{N} \sum_{\tau} \sum_{t} \log \pi_{\theta}(s_\tau, a_\tau) A^\pi(s_t, a_t) \quad (5)$$

An important aspect of RL is exploration, where during the learning phase, the agent will intentionally take an action that it knows to be suboptimal. This can lead the agent to previously unexplored regions of state space that ultimately lead it to learn a better policy. Typically the RL agent will take an action that is drawn from some probability distribution over possible actions, as learning proceeds the distribution becomes more peaked at optimal actions. If the RL agent learns an internal model of its environment, the agent can plan by exploring multiple steps into the future.

Deep reinforcement learning uses function approximators (such as a neural network) to approximate the value and policy functions, which allows extending the RL framework to high dimensional state and action spaces. Indeed, deep RL has proven successful at tasks such as learning to play Atari games from pixel level input, autonomous helicopter acrobatics, mastering the game of Go, and bipedal robots learning to walk.

**FOCA-IOC ALGORITHM**

A functional view of the FOCA-IOC algorithm is shown in Figure. A navigation system provides the following information, assuming a target centered reference frame:
1. An estimate of the spacecraft and obstacle states a short time in the future (we will refer to this time as $t_{gen}$, the maximum time it will take to compute an optimal trajectory)

2. Periodic estimates of spacecraft and obstacle states, which allows the tracking controller to track the trajectory. The state estimation frequency depends on the application.

3. A dynamics model that allows accurate simulation of the spacecraft and obstacle states; this model is used both by GPOPS and the APF trajectory generator.

A trajectory is quickly (less than 1 second) generated using an artificial potential function (APF) trajectory generator, with initial conditions set to where the spacecraft will be $t_{gen}$ seconds in the future, as estimated by the navigation system. The resulting trajectory is stored and given to the optimal control software (GPOPS) as an initial guess. GPOPS then uses this guess in its optimal trajectory computation. This trajectory is then tracked using a trajectory-tracking controller, which outputs a thrust vector that keeps the spacecraft following the desired trajectory. If while following the trajectory the spacecraft’s navigation system estimates that an obstacle is misbehaving (not following the estimated trajectory), or another previously undetected obstacle is detected, the trajectory can be quickly re-calculated based off of the new information, using the current trajectory as a guess.

We found that using an artificial potential function (APF) to create the initial guess for the optimal control solution significantly reduced both the run time and the variation in run times for the optimal control solution. We obtained further reduction in run time variability by adding path constraints one at a time, and using the solution with $n - 1$ path constraints as the initial guess for the problem with $n$ path constraints. Although recent work$^{11–13}$ has explored trajectory generation for spacecraft rendezvous, these methods did not allow exact modeling of non-linear dynamics and the use of arbitrary non-linear constraints, as is possible with the GPOPS package.

The ability to generate real-time optimal trajectories allowed us to develop the FOCA-IOC (Fast Optimal Conjunction Avoidance with Integrated Optimal Control) guidance system in our Phase I work. A spacecraft implementing the FOCA-IOC guidance algorithm will have the ability to quickly and autonomously generate a fuel-optimal trajectory that implements conjunction avoidance with respect to any obstacles detected by the navigation system as well as the target structure, and then track that trajectory to the target. Moreover, if while the spacecraft is following the trajectory the navigation system detects a new obstacle, or that an existing obstacle is not following its predicted trajectory, FOCA-IOC can quickly regenerate a new trajectory, and follow that trajectory to the target. This gives an unprecedented level of robustness for spacecraft autonomous rendezvous in dynamic cluttered environments. The guidance system is also applicable to the in-orbit assembly of structures from multiple components.

Given the estimated spacecraft state $t_{gen}$ seconds into the future and a dynamics model, both provided by the navigation system, the APF trajectory generator will quickly (less than 1 second) generate a non-optimal, but functional trajectory from the future spacecraft initial state to the target. The trajectory is generated by simulating the spacecraft trajectory using an APF and the velocity field tracking controller discussed below. The resulting trajectory (position, velocity, and thrust) is stored and given to the optimal control software (GPOPS) as an initial guess. The APF calculates a velocity field that drives the spacecraft towards the target. Note that the velocity field is calculated without any knowledge of obstacles; the field simply outputs a velocity as a function of position that drives the spacecraft to the target. The reason obstacles are neglected is that in the general case of
multiple obstacles in relative motion with respect to the target, it is possible for the generated APF trajectories to become unstable. Specifically, the magnitude of the field is calculated as

\[ v(d) = v_0 \left( 1 - \exp \left( \frac{-d}{\sigma} \right) \right) \]  

(6)

where \( d = ||r_{SC}|| \) is the spacecraft position in the target centered reference frame, and \( \sigma \) and \( v_0 \) were chosen to be 10 m and 2 m/s, respectively. The direction vector is calculated as \( v = r_{SC}/||r_{SC}|| \).

A simple non-linear controller calculates a thrust vector as a function of the velocity field. We used a non-linear controller because we found this gave improved fuel efficiency for the APF trajectories, thereby giving a better guess for GPOPS. The controller gives thrust as a function of \( v_{error} \), the error between the simulated spacecraft velocity and the target velocity as given by the velocity field. Specifically, the function is

\[ T = 1.5 \text{sign} (v_{error}) ||v_{error}||^2 m_o \]  

(7)

where \( m_o \) is the nominal mass of the spacecraft. The thrust is then limited to the thrust capability of the spacecraft.

**OPTIMAL TRAJECTORY GENERATOR**

GPOPS uses the trajectory provided by the APF generator as an initial guess for the solution of the optimal trajectory generation problem. We found that we could get better bounds on maximum run time by taking a multi-step approach to optimal trajectory generation, adding obstacles one at a time, and using the previous GPOPS run as a guess for the next. For example, if there were three obstacles with associated keep-out zones, the optimization flow would be:

1. Generate optimal trajectory assuming no obstacles, using APF trajectory as guess
2. Generate optimal trajectory with path constraints for one obstacle, using previous optimal trajectory as guess
3. Generate optimal trajectory with path constraints for two obstacles, using previous optimal trajectory as guess
4. Generate optimal trajectory with path constraints for three obstacles, using previous optimal trajectory as guess

The optimal trajectory returned by GPOPS consists of spacecraft position, velocity, and thrust vectors at sparse time samples along the trajectory. To create a dense trajectory of evenly spaced time samples, we interpolate the trajectory using cubic splines. The trajectory tracker can then use the interpolation function to obtain the target position and velocity given the current elapsed time since the start of the trajectory. This trajectory is then tracked using a trajectory-tracking controller, which outputs a thrust vector that keeps the spacecraft following the desired trajectory. If while following the trajectory the spacecraft’s navigation system estimates that an obstacle is misbehaving (not following the estimated trajectory), or another previously undetected obstacle is detected, the trajectory can be quickly re-calculated based off of the new information, using the current trajectory as a guess.
In environments with a very large number of obstacles, run time can also be reduced if when running the trajectory optimization, we only include obstacles that are predicted to result in conjunction. Specifically, we would run an initial trajectory optimization without any obstacles, and then run that trajectory forward using our estimated dynamics model. If this trajectory results in conjunction with one or more obstacles, we would then re-optimize the trajectory with those obstacles included in the optimization problem, using the trial trajectory as a guess. The new trajectory might then cause conjunction with one or more other obstacles, resulting in a new trajectory optimization, using the previous trajectory as an initial solution. In general, this should result only a small subset of the total number of obstacles detected by the navigation system being included in the final optimized trajectory, and will not add to total run time, as we already add obstacle path constraints one at a time when optimizing a trajectory.

TRAJECTORY TRACKING CONTROLLER

The optimal trajectory is tracked using a trajectory-tracking controller, which outputs a thrust vector that keeps the spacecraft following the desired trajectory. Most of our results are presented using a linear quadratic regulator (LQR), which gives an acceleration vector \( \mathbf{a} \) as a linear function of the tracking error. The commanded acceleration is then converted to a commanded thrust using the spacecraft’s nominal mass. For pulsed thruster applications, the controller will only thrust when the commanded thrust in a given direction exceeds 3 times the engine’s maximum thrust. We also present limited results using a trajectory tracking-controller derived using reinforcement learning (RL).

An LQR uses linear dynamics and a quadratic cost function. The dynamics are given by \( \mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \), where \( \mathbf{x} \) is the state vector, \( \mathbf{A} \) the dynamics matrix, and \( \mathbf{B} \) the control matrix. To populate the \( \mathbf{A} \) matrix, we assume simplified equations of motion with state vector \( \mathbf{x} = [r, v]^T \) and \( v = \dot{r} \), where \( r \) and \( v \) are the spacecraft’s position and velocity, respectively. Accuracy and fuel efficiency are traded off using the \( \mathbf{Q} \) and \( \mathbf{R} \) matrices, respectively.

For the continuous thrust case, the LQRs system matrix (a linearized model of the system dynamics) is

\[
\begin{align*}
A(t) &= \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, & B(t) &= \begin{bmatrix} 0 \\ I \end{bmatrix} \\
Q(t) &= \text{diag}(\mathbf{q}), & R(t) &= I
\end{align*}
\]

where \( I \) is a \( 3 \times 3 \) identity matrix, \( 0 \) a \( 3 \times 3 \) zero matrix, \( \text{diag}(\mathbf{x}) \) is a diagonal matrix with the diagonal elements given by the vector \( \mathbf{x} \), and \( \mathbf{q} = [0.1, 0.1, 0.1, 1.0, 1.0, 1.0]^T \). For the pulsed thrust case, we used different \( \mathbf{Q} \) and \( \mathbf{R} \) matrices, with \( \mathbf{q} = [0.2, 0.2, 0.2, 2.0, 2.0, 2.0]^T \) and \( R = 4I \). The controller gain matrix is then given by

\[
K = R^{-1}B^TP
\]

\( P \) is found by solving the algebraic Riccati equation

\[
A^T P + PA - PBR^{-1}B^TP + Q = 0
\]

The controller acceleration is then given by \( \mathbf{a} = -K\mathbf{e} - \mathbf{u}_{\text{env}} \), where \( \mathbf{e} = \mathbf{x}_{\text{sc}} - \mathbf{x}_{\text{traj}}(t) \), \( \mathbf{x}_{\text{sc}} \) is the spacecraft state, \( \mathbf{x}_{\text{traj}}(t) \) is the target state given by the trajectory for time \( t \), and \( \mathbf{u}_{\text{env}} \) is the acceleration imparted by environmental forces, which given a model of the environment, would
generally be known. The commanded acceleration is then converted to a thrust by multiplying it by the spacecraft’s nominal mass.

For pulsed thrust engines, the actual thrust is calculated as follows, where \( i \in \{1, 2, 3\} \), \( T_c[i] \) is the commanded thrust in direction \( i \) as given by the LQR, and \( T_{\text{max}} \) the spacecraft’s maximum thrust.

\[
T = \begin{cases} 
0 & \text{for } |T_c[i]| < 3T_{\text{max}} \\
T_{\text{max}} & \text{for } |T_c[i]| > 3T_{\text{max}} \\
-T_{\text{max}} & \text{for } |T_c[i]| < -3T_{\text{max}} 
\end{cases}
\] (11)

This results in a sparse, and fairly fuel-efficient pulse schedule. The factor of 3 was chosen empirically, and could probably be increased if we used a non-linear trajectory tracker that would exhibit less overshoot for larger tracking errors.

**REINFORCEMENT LEARNING (RL) AGENT CONTROLLER**

In an attempt to improve the fuel efficiency with pulsed thrust engines, we used reinforcement learning to train an RL agent to act as a controller. In the RL framework, an agent learns an optimal policy (a mapping of a state to an action) by interacting with its environment. In more detail, the agent samples an action from its policy based off of its current state. This sampled action is used to take a step in the environment, which returns a reward, a Boolean indication as to whether the episode has terminated, and the next state. The rewards are used to learn a value function, a mapping of the agent’s state to the value of being in that state. The value function can in turn be used to evaluate and improve the agent’s current policy.

The environment/agent interface was chosen to facilitate learning a fuel-efficient policy that maps the agent’s state to an action. The agent’s state is the error between the spacecraft’s state (position and velocity) and the desired state given the trajectory to be followed, and the action is, for each of the three spatial dimensions, either positive full thrust, negative full thrust, or no thrust.

Here the value function approximator learns a mapping from states to values, where the value is the discounted sum of rewards expected when starting in that state and following the policy. The dataset used for learning the mapping of observations to values is constructed by accumulating observations (the tracking error) and the discounted sum of rewards (target value) for multiple trajectories. Over the same set of trajectories, the value function is used to compute a predicted value for the set of observations. This predicted value is used in conjunction with the rewards obtained over those trajectories to create the advantages, which is a measure of the trajectory’s sum of discounted rewards as compared to a baseline (the predicted value). Here we create the advantage vector using the generalized advantage estimator [20]. The advantages and rewards accumulated over the trajectories are used to update the policy (the mapping of states to actions) in a direction that maximizes the expected value over the distribution of possible initial states. The policy update increases the probability of generating actions that result in a positive advantage, and vice versa.

At each interaction between the agent and environment, an action (thrust vector) is sampled by the policy, and the environment returns the tracking error, which is treated as an observation by the agent, a reward, and a done signal indicating that the trajectory has ended. At each time step, the reward function gives the reward \( r \) as a function of the thrust vector \( T \).

\[
r = 1.0 - 0.8 \frac{\|T\|}{T_{\text{max}}} \] (12)
In words, a positive reward is gained at each time step, which incentivizes the agent to complete the trajectory, which terminates if the tracking error exceeds a threshold. A negative reward proportional to control effort discourages unnecessary use of the thrusters. We also developed a continuous thrust RL controller, which used a slightly modified reward function:

\[ r = \exp \left( -\frac{\|p_{\text{err}}\|^2}{k} \right) - 0.1 \frac{\|T\|}{T_{\text{max}}} \]  

We did not have much time to spend on optimizing the RL agent; indeed, we just used an existing agent implementing proximal policy optimization without making any changes. The agent was designed to learn the open AI gym mujoco environments. 1000 trajectories with two static obstacles were used for learning, and the policy was tested on a novel set of trajectories generated using one static obstacle and two dynamic obstacles. Results for both RL controllers are given in the results section.

RESULTS

To demonstrate the FOCA-IOC algorithm, we integrated Matlab and GPOPS into our Python simulator. The simulator is 3-DOF. We assume a spacecraft with a full mass of 1000kg, with engines that can generate a thrust vector of up to 50N in each of the directions aligned with the axes of the target centered reference frame. We model both the case of continuous thrust (throttleable) and pulsed thruster engines. The navigation system is modeled as having a state estimation update period of 1 second.

The equations of motion for the spacecraft and obstacles are given by the Clohessy-Wiltshire equations, with the spacecraft equations of motion shown below, where \( r \) is the spacecraft’s position vector, \( T \) the spacecraft thrust vector, \( m \) the spacecraft’s mass, \( I_{\text{sp}} \) the engine’s specific impulse, and \( g_0 \) the acceleration of gravity on Earth. Here we use \( n = \sqrt{\mu/(r_T^3)} \), where \( \mu = 398600 \times 10^9 \) and \( r_T = (400 + 6378) \times 10^3 \).

\[
\ddot{x} - 3n^2 x - 2n \dot{y} = \frac{T_x}{m} \tag{14a}
\]
\[
\ddot{y} + 2n \dot{x} = \frac{T_y}{m} \tag{14b}
\]
\[
\ddot{z} + n^2 \dot{z} = \frac{T_z}{m} \tag{14c}
\]
\[
\dot{m} = -\frac{\|T\|}{I_{\text{sp}} g_0} \tag{14d}
\]
The equations of motion are integrated using 4th order Runge-Kutta integration with a 1 second time step. Motion of obstacles is given by the same equations, except that the thrust terms are zero, as we assume in this work that the obstacles are unpowered.

Figure 3. Sample Trajectories for Continuous Thrust RL Controller.

The spacecraft initial conditions are chosen randomly for a given simulation. For example, the initial position could be randomly chosen somewhere on the surface of a 1-km radius sphere centered on the target, and the initial velocity with a random direction vector, and a magnitude between 0 and 5 m/s. Note that we assumed the spacecraft and obstacle initial conditions are the estimated states $t_{gen}$ seconds into the future, as predicted by the navigation system, and therefore are a good estimate of the spacecraft’s state when it starts the trajectory. Sample trajectories are shown below in Figure for the case of two obstacles, each with an associated spherical keep-out zone. Obstacle trajectories have a heavier line weight, and each obstacle is shown at its final position. Note the additional third obstacle slightly offset from the origin. This represents the structure of the target spacecraft, where the goal of the rendezvous is to reach a location at the origin, while avoiding the structure of the target spacecraft. We assume that from this point a docking routine would take over, closing the gap of 10-m. The optimal control problem was constrained with a maximum time of flight equal to 900s.

The simulator can optionally be configured to model various non-ideal effects, including:

1. Uncertainties in the spacecraft’s mass used to convert acceleration to thrust
2. Initial condition prediction uncertainty. This is in addition to the randomness in initial condition generation. Specifically, the spacecraft’s initial state as seen by the trajectory-tracking
controller would be corrupted by adding uniform random noise. This is used to model the effects of errors in predicting the spacecraft’s state $t_{gen}$ seconds into the future to account for the time it takes to generate the optimal trajectory.

3. Whether or not the trajectory-tracking controller has access to the system dynamics

4. Environmental noise. Environmental noise would combine the effects of engine noise (the actual thrust won’t always be equal to the commanded thrust), solar wind, atmospheric drag, or any other un-modeled force.

5. Sensor Noise

### Simulation Results with No Noise

The simulation parameters are given in Table 1, where most fields were described earlier. The GPOPS intervals and nodes per interval fields roughly describe the resolution used by GPOPS to solve the problem. The nodes per interval are automatically doubled whenever GPOPS does a mesh iteration. Note that the GPOPS maximum thrust constraint indicates the maximum norm of the thrust vector, whereas the Spacecraft maximum thrust simulation parameter indicates the maximum thrust along an axis in $\mathbb{R}^3$, which we thought was more realistic for the applications of spacecraft rendezvous and asteroid close proximity applications.
Sample trajectories, GPOPS statistics, continuous thrust performance, and pulsed thrust performance are shown below in Figure 3, Figure 2, Figure 4, and Figure 5, respectively. Note that the minimum thrust magnitude reported by GPOPS is negative. This was investigated, and it turns out that this only occurs on the last trajectory point output by GPOPS. Perhaps it is a bug in the version we are using, but it does not appear to have any material impact. The maximum absolute value of Hamiltonian field is an indication of how well GPOPS was able to solve the optimal control problem. When the maximum Hamiltonian gets around 1e-2, there is usually something wrong with the trajectory. The GPOPS run times are on average around 6 seconds, with a standard deviation of 3 seconds, but there are some outliers, with the maximum run time of close to one minute. Note that fuel efficiency using continuous thrust is identical to that of GPOPS. Pulsed thrust fuel consumption is approximately 40% (1.2kg) higher than the continuous thrust case. There is a slight violation of keep-out zones. In the following sections, the simulation parameters are as given in Table , except with the addition of the indicated noise source.

<table>
<thead>
<tr>
<th>batch run stats:</th>
<th>mean</th>
<th>std</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Used (kg):</td>
<td>4.73</td>
<td>1.38</td>
<td>8.74</td>
</tr>
<tr>
<td>Final Position Error (m):</td>
<td>0.88</td>
<td>0.09</td>
<td>1.02</td>
</tr>
<tr>
<td>Final Velocity Error (m/s):</td>
<td>0.03</td>
<td>0.01</td>
<td>0.07</td>
</tr>
<tr>
<td>Norm Position Tracking Error (m):</td>
<td>0.89</td>
<td>0.54</td>
<td>9.35</td>
</tr>
<tr>
<td>Norm Velocity Tracking Error (m/s):</td>
<td>0.04</td>
<td>0.07</td>
<td>0.90</td>
</tr>
<tr>
<td>Velocity (m/s):</td>
<td>1.50</td>
<td>0.74</td>
<td>5.55</td>
</tr>
<tr>
<td>Thrust (N):</td>
<td>10.04</td>
<td>22.04</td>
<td>86.60</td>
</tr>
<tr>
<td>Trajectory Duration (s):</td>
<td>898.81</td>
<td>6.13</td>
<td>899.00</td>
</tr>
<tr>
<td>Error:</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spacecraft minimum distance to surface of RSO 0 (m):</td>
<td>-3</td>
<td>IDX: 500</td>
<td></td>
</tr>
<tr>
<td>Spacecraft minimum distance to surface of RSO 1 (m):</td>
<td>-2</td>
<td>IDX: 852</td>
<td></td>
</tr>
<tr>
<td>Spacecraft minimum distance to surface of RSO 2 (m):</td>
<td>-5</td>
<td>IDX: 382</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Results with Pulsed Thrusters and Initial Condition Prediction Error.

Initial Condition Prediction Error

Figure 6 shows results where we added a prediction error to the navigation system’s estimate of the spacecraft’s state at the start of the trajectory. The position error ($\in \mathbb{R}^3$) was uniformly distributed between -5 m and 5 m, whereas the velocity error ($\in \mathbb{R}^3$) was uniformly distributed between -0.5 m/s and 0.5 m/s. These values are unrealistically large, but we wanted to find where such errors began to impact performance. Note the increase in maximum tracking error due to initial distance to start of trajectory, and increased fuel consumption due to the controller pulling in to the trajectory.

RL Controller Results

The simulation results for the continuous thrust and pulsed thrust RL controllers are given in Figure 8 and Figure 7, respectively. The agent learned using trajectories generated with two static obstacles. After learning, the agent’s performance was tested on trajectories generated with two dynamics obstacles and a single static obstacle. Although on average fuel efficiency is slightly worse than the LQR controller, the variance in fuel consumption is lower with the RL controller. It is likely performance could have been improved using one or more of the following techniques: Learning an action conditional model of environment and using the model to predict the future trajectory before taking an action (planning), Using a softmax distribution for the policy: In this
study, we used an agent with a Gaussian policy and discretized the actions. Pre-train a policy using supervised learning, and use reinforcement learning to improve the policy, and Learning on a larger and more varied set of trajectories.

CONCLUSION

The small violation of keep-out zones around obstacles is not due to tracking error, as the maximum tracking error in the continuous thrust simulations was 0.29m (see Figure 21). It is also not due to the GPOPS trajectory, as all GPOPS runs met the path constraints. The likely cause of the violation is the sparse representation of the optimal trajectory returned by GPOPS. For example, using 10 intervals and 4 nodes per interval results in a trajectory with 40 points. When such a trajectory is interpolated using cubic splines, the resulting trajectory is slightly different than the ordinary differential equations integrated within GPOPS. In this version of GPOPS, there does not appear to be a way to increase the resolution of the returned optimal trajectory, and if we increase the number of intervals and nodes per interval, the GPOPS run time increases. However, the violation is small, and can be compensated for using a margin of safety when calculating the keep-out zones. An alternative to using a margin of safety would be to replace the cubic splines interpolation with an integration of the equations of motion (executed in either Matlab or Python) between the points returned by GPOPS. This should result in a near-perfect match between the GPOPS trajectory and the trajectory tracked by the controller. This will be investigated in Future work. Although GPOPS

<table>
<thead>
<tr>
<th>batch run stats:</th>
<th>mean</th>
<th>std</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Used (kg):</td>
<td>3.24</td>
<td>0.88</td>
<td>6.27</td>
</tr>
<tr>
<td>Final Position Error (m):</td>
<td>0.02</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>Final Velocity Error (m/s):</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>Norm Position Tracking Error (m):</td>
<td>0.02</td>
<td>0.01</td>
<td>0.16</td>
</tr>
<tr>
<td>Norm Velocity Tracking Error (m/s):</td>
<td>0.00</td>
<td>0.00</td>
<td>0.11</td>
</tr>
<tr>
<td>Velocity (m/s):</td>
<td>1.46</td>
<td>0.70</td>
<td>5.00</td>
</tr>
<tr>
<td>Thrust (N):</td>
<td>7.43</td>
<td>4.83</td>
<td>69.68</td>
</tr>
<tr>
<td>Trajectory Duration (s)</td>
<td>898.75</td>
<td>8.03</td>
<td>899.00</td>
</tr>
</tbody>
</table>

Error: False

Spacecraft minimum distance to surface of RS0 0 (m): -2 IDX: 897
Spacecraft minimum distance to surface of RS0 1 (m): -3 IDX: 998
Spacecraft minimum distance to surface of RS0 2 (m): -5 IDX: 678
run times have a mean and standard deviation of around 6 and 3 seconds respectively, occasional outliers give run times as high as 60 seconds. Although this is more than fast enough for the likely applications (rendezvous, orbital transfer, asteroid close proximity operations, and moon landing from orbit), further work may be able to reduce the maximum run time.

REFERENCES


