Available online at www.sciencedirect.com

ScienceDirect

Advances in Space Research 65 (2020) 1723–1741

ADVANCES IN SPACE RESEARCH (a COSPAR publication)

www.elsevier.com/locate/asr

Deep reinforcement learning for six degree-of-freedom planetary landing ☆

Brian Gaudet [a], Richard Linares [b,*], Roberto Furfaro [a]

[a] Department of Systems and Industrial Engineering, University of Arizona, United States
[b] Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, United States

Received 19 August 2019; received in revised form 20 December 2019; accepted 27 December 2019
Available online 7 January 2020

Abstract

This work develops a deep reinforcement learning based approach for Six Degree-of-Freedom (DOF) planetary powered descent and landing. Future Mars missions will require advanced guidance, navigation, and control algorithms for the powered descent phase to target specific surface locations and achieve pinpoint accuracy (landing error ellipse <5 m radius). This requires both a navigation system capable of estimating the lander's state in real-time and a guidance and control system that can map the estimated lander state to a commanded thrust for each lander engine. In this paper, we present a novel integrated guidance and control algorithm designed by applying the principles of reinforcement learning theory. The latter is used to learn a policy mapping the lander's estimated state directly to a commanded thrust for each engine, resulting in accurate and almost fuel-optimal trajectories over a realistic deployment ellipse. Specifically, we use proximal policy optimization, a policy gradient method, to learn the policy. Another contribution of this paper is the use of different discount rates for terminal and shaping rewards, which significantly enhances optimization performance. We present simulation results demonstrating the guidance and control system's performance in a 6-DOF simulation environment and demonstrate robustness to noise and system parameter uncertainty.

Keywords: Reinforcement learning; Mars landing; Integrated guidance and control; Artificial intelligence; Autonomous maneuvers

1. Introduction

Future unconstrained, science-driven, robotic and human missions to large and small planetary bodies will require a high degree of landing accuracy. Indeed, the next generation of planetary landers will need more advanced guidance and control capabilities to satisfy the increasingly stringent accuracy requirements. The latter is driven by the desire to explore regions on planets (e.g. Mars) and satellites (e.g. Moon) that have the potential to yield the highest scientific return. In the case of Mars, the most demanding mission phase is probably the powered descent, where the goal is to achieve a soft pinpoint landing, which we will define as the norm of the position error less than 5 m and the magnitude of the landing velocity below 2 m/s, with the velocity vector directed primarily downward, and negligible deviation from an upright attitude and zero rotational velocity. During a typical Entry, Descent and Landing (EDL) as implemented in past robotic missions to Mars (Shotwell, 2005; Braun and Manning, 2007), the lander's sensors (radar altimeters) are effectively blind until the heat shield is jettisoned, at which point the lander's guidance, navigation, and control system must quickly estimate the lander's state from a prior distribution extending

☆ Presented at 2018 AAS/AAIA Astrodynamics Specialist Conference.

* Corresponding author.
E-mail address: linaresr@mit.edu (R. Linares).

https://doi.org/10.1016/j.asr.2019.12.030
0273-1177/

several km downrange and crossrange, and then use the sequence of state estimates to achieve a soft landing at the target position, typically within one minute of the heat shield being jettisoned.

For the case of the robotic exploration of Mars, landing accuracy is important for several reasons. First, given accurate high-resolution maps of the Mars surface, the pinpoint-landing problem subsumes the hazard avoidance problem, as a hazard-free landing site can be targeted. Second, delivering a rover closer to a location of scientific interest reduces the risk of the rover malfunctioning before it reaches the desired site. In fact, some sites might be inaccessible to a rover unless the lander can deliver the rover with pinpoint accuracy. Moreover, reducing the distance the rover is required to travel can relax the design requirements for the rover, with a potential reduction in rover mass. Clearly, landing accuracy on the order of several meters is desirable as it would both reduce mission risk and extend the scope of feasible missions.

Several recent works have applied lossless convexification of non-convex control problems to the 3-DOF planetary landing Açkmeşe et al. (2013), Acikmese and Ploen (2007), Sagliano (2018), Wang and Cui (2018) and have achieved satisfactory accuracy. Imitation learning has also been applied to the powered descent phase problem Sánchez-Sánchez and Izzo (2018) for several 2-DOF landing problems. However, it has been shown in Ross et al. (2011) that a naive approach of imitation learning will likely fail for more realistic (and complex) problems due to the distribution mismatch problem, which they solve using a dataset aggregation algorithm (DAgger). Guided policy search (Levine and Koltun, 2013) is another approach to allow an agent to learn from expert demonstrations that does not suffer from the distribution mismatch problem.

In this work, we develop an integrated guidance and control system that learns a global policy over the region of state space defined by the deployment ellipse and potential landing sites. This global policy maps the navigation system's estimate of the lander's state directly to commands specifying thrust levels for each engine. The global policy is learned using reinforcement learning (RL). Learning involves simulated interaction between an agent instantiating the policy and the environment over many episodes with randomly generated initial conditions that cover the deployment ellipse. Path constraints are imposed on the lander's attitude during the powered descent phase. Further, each engine's thrust is constrained to remain within specified lower and upper limits. Note that there is no guarantee on the optimality of trajectories induced by the policy, although in practice it is possible to get close to optimal performance by tuning the reward function. In order to suggest problems where the RL approach may be especially useful, we include a comparison of RL and optimal control approaches to guidance and control in Table 1.

Previous work using RL to solve control problems has focused on applications in robotics, with very few published works addressing problems in aerospace guidance and control. The first application of RL to a problem in the aerospace domain was applied to autonomous helicopter flight (Ng, 2003). More recently, in Waslander et al. (2005) the authors compared reinforcement learning to sliding mode control and linear control in a quadrotor control application. The authors used policy iteration (Sutton and Barto, 1998) in a model-based formulation, where a model was learning from flight data using weighted least squares. Both the sliding mode and RL methods resulted in stable controllers, whereas the linear control methods failed. In previous work, we achieved good results in a 2-DOF environment, but over a limited range of initial conditions (Gaudet and Furfaro, 2014). More recently, RL has been applied to missile homing phase interception using angle only measurements (Gaudet et al., 2019a) and an adaptive fully integrated guidance, navigation, and control for asteroid close proximity missions (Gaudet et al., 2019c). And in Gaudet et al. (2019b), the authors demonstrate a 3-DOF adaptive guidance system for Mars powered descent phase optimized via reinforcement meta-learning.

To our knowledge, this is the first published work demonstrating an RL-derived integrated guidance and control system applied to planetary landing in a 6-DOF environment. Open-AI gym has one environment for a 3-DOF planar lunar landing[1], but the problem is simplified by using a range of initial conditions that are unrealistically reduced from that required for an actual lunar landing (the crossrange dispersion is only three times the width of the landing site). In our work, we have found that solutions that work over a limited range of initial conditions often fail when the initial condition range is extended. Our ultimate goal is to use RL to develop a fully integrated guidance, navigation, and control system, where the policy maps raw sensor observations directly to actuator commands, i.e., radar altimeter readings to commanded thrust for the lander's individual engines. The RL framework can be applied to solve many different types of guidance and control problems, including missile homing phase guidance, exoatmospheric intercepts, hypersonic reentry maneuvering for planetary landings, and booster recovery via powered landing.

This paper is organized as follows. In Section 2, we give some background on optimization via reinforcement learning. The problem formulation is then given in Section 3. In Section 4, we show the results from policy optimization and testing for 6-DOF planetary landing scenarios. In Section 5, a comparison between RL-based 3-DOF/6-DOF closed-loop policy and open-loop optimal trajectory derived using direct methods is reported. In Section 6, conclusions and future work are reported.

[1] https://gym.openai.com/envs/LunarLander-v2/.

Table 1
A comparison of optimal control and RL Tedrake (2015)

Optimal control	Reinforcement learning
Single trajectory (except for trivial cases where HJB equations can be solved)	Global over theatre of operations
Unbounded run time except for special cases such as convex constraints	Extremely fast run time for trained policy (<1 ms in this work)
Dynamics need to be represented as ODE, possibly constraining fidelity of model used in optimization	No constraints on dynamics representation. Agent can learn in a high fidelity simulator (i.e., Navier-Stokes modeling of aerodynamics)
Open Loop (requires a controller to track the optimal trajectory)	Closed Loop (Integrated guidance and control)
Output feedback (co-optimization of state estimation and guidance law) an open problem for non-linear systems	Can learn from raw sensor outputs allowing fully integrated GNC (pixels to actuator commands). Can learn to compensate for sensor distortion
Requires full state feedback	Does not require full state feedback
Elegantly handles state constraints	State constraints handled either via large negative rewards and episode termination or more recently, modification of policy gradient algorithm. Control constraints straightforward to implement
Deterministic Optimization	Stochastic Optimization, learning does not converge every time, may need to run multiple policy optimizations

2. Background: reinforcement learning

2.1. Reinforcement learning overview

RL algorithms can be broken down into two major classes: value function methods and policy gradient methods. Value function-based algorithms learn a mapping between a state-action tuple and the sum of the future discounted rewards received when starting in that specific state and taking that specific action. Actions with the highest value are then greedily chosen, with limited exploration achieved by choosing a random action with some small probability. Since all possible actions must be considered, these methods only work with discrete action spaces. First proposed by Watkins and Dayan (1992), recent work in value function-based algorithms include deep Q networks (Mnih et al., 2015). The latter use experience replay to remove temporal correlation between samples and target networks which results in a reduced instability of combining bootstrapping with non-linear function approximators. Deep Q networks have proven effective at control tasks requiring the mapping of pixel level observations directly to control actions, as demonstrated in Mnih et al. (2015), although they are less effective at control problems where the dimensionality of the dynamics is larger, such as problems in robotic control, and are not suitable for problems requiring continuous action spaces. Deterministic policy gradient algorithms, first proposed by Silver et al. (2014), and including the DDPG algorithm (Lillicrap et al., 2015), can be employed as an alternative to methods based on Deep Q learning. These algorithms also learn a mapping between state-action tuples and values. However, rather than taking actions that globally maximize the value function, the algorithm maintains a separately parameterized policy mapping states to actions, and uses the chain rule to follow the gradient of the value function with respect to the action, given that the action is a function of the state as parameterized by the policy parameters, $\boldsymbol{\theta}$.

Generally, policy gradient algorithms learn a direct stochastic mapping between an observation and action, with the policy network's parameters updated such that the probability of actions leading to higher future rewards is increased. First proposed by Williams (1992), these algorithms suffer from high variance, which can be substantially reduced by using a state value function baseline. With a baseline, actions that result in future discounted rewards higher than the estimated value of being in that state, as given by the state value function, have their probability increased. In this approach, a state value function mapping a state to the expected sum of future discounted rewards is learned concurrently with a stochastic policy mapping states to actions. For continuous action spaces, it is common to parameterize the policy as a Gaussian distribution with a diagonal covariance matrix, where the policy outputs the mean and variance of the actions conditioned on the state. Recently, Schulman et al. (2015) have proven that policy gradient methods with stochastic policies can have monotonic improvement guarantees, provided that the policy changes during optimization as measured by the Kullback-Leibler (KL) divergence are constrained to be within certain bounds. The policy optimization problem is posed as a constrained optimization problem that ensures the KL divergence between policy updates remains within specified bounds. The authors used this result to develop the trust region policy optimization algorithm (TRPO), which has proven effective at solving high-dimensional control problems. Later, Schulman et al. (2017) proposed the proximal policy optimization (PPO) method that uses a heuristic to keep KL divergence between policy updates at a level that in practice ensures monotonic improvement during optimization. In practice, PPO works slightly better than TRPO, and has the advantage of not requiring second-order optimization methods. Policy gradient methods are much less sample efficient than methods based on Deep Q learning as they operate on policy. The latter means that policy gradient methods cannot take advantage of sample reuse through techniques such

as experience replay (Mnih et al., 2015). However, policy gradient methods are more stable, and work with minimal hyperparameter tuning, and tend to perform better in systems with high dimensional dynamics.

2.2. Policy gradient method

In the RL framework, an agent learns through episodic interaction with an environment how to successfully complete a task by learning a *policy* that maps observations to actions. The environment initializes an episode by randomly generating an internal state, mapping this internal state to an observation, and passing the observation to the agent. These observations could be a corrupted version of the internal state (to model sensor noise) or could be raw sensor outputs such as Doppler radar altimeter readings, or a multi-channel pixel map from an electro-optical sensor. 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 a scalar reward signal. 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 violating a constraint. Initially, the agent's actions are random, which allows the agent to explore the state space and begin learning the value of experiencing a given observation, and which actions are to be preferred as a function of this observation. Here the value of an observation is the expected sum of discounted rewards received after experiencing that observation; this is similar to the cost-to-go in optimal control. 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 lander. The safe method of continuous learning in the field is to have the lander send back telemetry data, which can be used to improve the environment's dynamics model and update the policy via simulated experience.

In the following discussion, the vector $\mathbf{x}_k$ denotes the observation provided by the environment to the agent. Note that in general $\mathbf{x}_k$ does not need to satisfy the Markov property. In those cases where it does not, several techniques have proven successful in practice. In one approach, observations spanning multiple time steps are concatenated, allowing the agent access to a short history of observations, which helps the agent infer the motion of objects in consecutive observations. This was the approach used in Mnih et al. (2015). In another approach, a recurrent neural network is used for the policy and value function implementations. The recurrent network allows the agent to infer motion from observations, similar to the way a recursive Bayesian filter can infer velocity from a history of position measurements. The use of recurrent network layers has proven effective in supervised learning tasks where a video stream needs to be mapped to a label (Baccouche et al., 2011).

Each episode results in a trajectory defined by observation, actions, and rewards; a step in the trajectory at time t_k can be represented as $(\mathbf{x}_k, \mathbf{u}_k, r_k)$, where $\mathbf{x}_k$ is the observation provided by the environment, $\mathbf{u}_k$ the action taken by the agent using the observation, and r_k the reward returned by the environment to the agent. The reward can be a function of both the observation $\mathbf{x}_k$ and the action $\mathbf{u}_k$. The reward is typically discounted to allow for infinite horizons and to facilitate temporal credit assignment. Then, the sum of discounted rewards for a trajectory can be defined as

$$r(\boldsymbol{\tau}) = \sum_{k=0}^{T} \gamma^k r_k(\mathbf{x}_k, \mathbf{u}_k) \tag{1}$$

where $\boldsymbol{\tau} = [\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_T, \mathbf{u}_T]$ denotes the trajectory, T is the total number of time steps, and $\gamma \in [0, 1)$ denotes the discount factor. The objective function the RL methods seek to optimize is given by

$$J(\boldsymbol{\theta}) = \mathbb{E}_{p(\boldsymbol{\tau})}[r(\boldsymbol{\tau})] = \int_{\mathbb{T}} r(\boldsymbol{\tau}) p_{\boldsymbol{\theta}}(\tau) d\boldsymbol{\tau} \tag{2}$$

where

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \left[\prod_{k=0}^{T} p(\mathbf{x}_{k+1} | \mathbf{x}_k, \mathbf{u}_{\boldsymbol{\theta}}) \right] p(\mathbf{x}_0) \tag{3}$$

where $\mathbb{E}_{p(\boldsymbol{\tau})}[\cdot]$ denotes the expectation over trajectories, $\mathbb{T}$ is the set of trajectories induced by the policy, and in general $\mathbf{u}_{\boldsymbol{\theta}}$ may be deterministic or stochastic function of the policy parameters, $\boldsymbol{\theta}$. However, it was noticed by Williams (1992) that if the policy is chosen to be stochastic, where $\mathbf{u}_k \sim \pi_{\boldsymbol{\theta}}(\mathbf{u}_k | \mathbf{x}_k)$ is a PDF for $\mathbf{u}_k$ conditioned on $\mathbf{x}_k$, then a simple policy gradient expression can be found:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \mathbb{E}_{p(\boldsymbol{\tau})} \left[\sum_{k=0}^{T} r_k(\mathbf{x}_k, \mathbf{u}_k) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k | \mathbf{x}_k) \right] \\ &\approx \sum_{i=0}^{M} \sum_{k=0}^{T} r_k(\mathbf{x}_k^i, \mathbf{u}_k^i) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k^i | \mathbf{x}_k^i) \end{aligned} \tag{4}$$

where the integral over $\boldsymbol{\tau}$ is approximated with M samples from $\boldsymbol{\tau}^i \sim p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ which are Monte Carlo roll-outs of the policy given the environment's transition PDF, $p(\mathbf{x}_{k+1} | \mathbf{x}_k)$. The expression in Eq. (4) is called the policy gradient and the form of this equation is referred to as the REINFORCE method by Williams (1992). Since the development of the REINFORCE method, additional theoretical work has led to improvements in the performance of policy gradient-based methods. In particular, it was shown that the reward $r_k(\mathbf{x}_k, \mathbf{u}_k)$ in Eq. (4) can be replaced with state-action value function $Q^{\pi}(\mathbf{x}_k, \mathbf{u}_k)$, this result is known as the Policy Gradient Theorem. Furthermore, the variance of the policy gradient estimate that is derived from the

Monte Carlo roll-outs, $\boldsymbol{\tau}^i$, is reduced by subtracting a state-dependent basis from $Q^\pi(\mathbf{x}_k, \mathbf{u}_k)$. This basis is commonly chosen to be the state value function $V^\pi(\mathbf{x}_k)$, and we can define $A^\pi(\mathbf{x}_k, \mathbf{u}_k) = Q^\pi(\mathbf{x}_k, \mathbf{u}_k) - V^\pi(\mathbf{x}_k)$. The function $A^\pi(\mathbf{x}_k, \mathbf{u}_k)$ is referred to as the Advantage function for the policy π and this function measures the relative reward between taking a general action $\mathbf{u}_k$ versus the action provided by the policy π. This method is known as the Advantage-Actor-Critic (A2C) Method. The policy gradient for the A2C method is given by

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \sum_{i=0}^{M} \sum_{k=0}^{T} A^\pi_{\mathbf{w}}(\mathbf{x}^i_k, \mathbf{u}^i_k) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}^i_k | \mathbf{x}^i_k) \tag{5}$$

where the $\mathbf{w}$ subscript denotes a function parameterized by $\mathbf{w}$.

2.3. Proximal policy optimization

The PPO approach (Schulman et al., 2017) is a type of policy gradient which has demonstrated state-of-the-art performance for many RL benchmark problems. The PPO approach is developed using the properties of the TRPO Method (Schulman et al., 2015). The TRPO method formulates the policy optimization problem using a constraint to restrict the size of the gradient step taken during each iteration (Sorensen, 1982). The TRPO method policy update is calculated using the following problem statement:

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{minimize}} \quad & \mathbb{E}_{p(\tau)}\left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{u}_k|\mathbf{x}_k)} A^\pi_{\mathbf{w}}(\mathbf{x}_k, \mathbf{u}_k)\right] \\ \text{subject to} \quad & \mathbb{E}_{p(\tau)}\left[\text{KL}\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k), \pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{u}_k|\mathbf{x}_k)\right)\right] \leqslant \delta \end{aligned} \tag{6}$$

where the function $\text{KL}(\cdot,\cdot)$ is the Kullback-Leibler (KL) divergence Kullback and Leibler (1951). The parameter δ is a tuning parameter but the theory justifying the TRPO methods proves monotonic improvement in the policy performance if the policy change in each iteration is bounded a parameter C. The parameter C is computed using the KL divergence. Reference Schulman et al. (2015) computes a closed-form expression for C but this expression leads to prohibitively small steps, and therefore, Eq. (6) with a fixed bound is used. Additionally, Eq. (6) is approximately solved using the conjugate gradient algorithm, which approximates the constrained optimization problem given by Eq. (6) with a linearized objective function and a quadratic approximation for the constraint. The PPO method approximates the TRPO optimization process by accounting for the policy adjustment constraint with a clipped objective function. The objective function used with PPO can be expressed in terms of the probability ratio $p_k(\boldsymbol{\theta})$ given by,

$$p_k(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{u}_k|\mathbf{x}_k)} \tag{7}$$

where the PPO objective function is then as follows:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{p(\tau)}\left[\min\left[p_k(\boldsymbol{\theta}), \text{clip}(p_k(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\right] A^\pi_{\mathbf{w}}(\mathbf{x}_k, \mathbf{u}_k)\right] \tag{8}$$

This clipped objective function has been shown to maintain the KL divergence constraints, which aids convergence by insuring that the policy does not change drastically between updates. PPO uses an approximation to the advantage function that is the difference between the empirical return and a state value function baseline, given by the following:

$$A^\pi_{\mathbf{w}}(\mathbf{x}_k, \mathbf{u}_k) = \left[\sum_{\ell=k}^{T} \gamma^{\ell-k} r(\mathbf{x}_\ell, \mathbf{u}_\ell)\right] - \mathbf{V}^\pi_{\mathbf{w}}(\mathbf{x}_\mathbf{k}) \tag{9}$$

Here the value function $V^\pi_{\mathbf{w}}$ is learned using the cost function given by

$$L(\mathbf{w}) = \sum_{i=1}^{M} \left(V^\pi_{\mathbf{w}}(\mathbf{x}^i_k) - \left[\sum_{\ell=k}^{T} \gamma^{\ell-k} r(\mathbf{u}^i_\ell, \mathbf{x}^i_\ell)\right] \right)^2 \tag{10}$$

In practice, policy gradient algorithms update the policy using a batch of trajectories (roll-outs) collected by interaction with the environment. Each trajectory is associated with a single episode, with a sample from a trajectory collected at step k consisting of observation $\mathbf{x}_k$, action $\mathbf{u}_k$, and reward $r_k(\mathbf{x}_k, \mathbf{u}_k)$. Finally, gradient ascent is performed on $\boldsymbol{\theta}$ and gradient descent on $\mathbf{w}$ and update equations are given by

$$\mathbf{w}^+ = \mathbf{w}^- - \beta_{\mathbf{w}} \nabla_{\mathbf{w}} L(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^-} \tag{11}$$

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta}^- + \beta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-} \tag{12}$$

where $\beta_{\mathbf{w}}$ and $\beta_{\boldsymbol{\theta}}$ are the learning rates for the value function, $V^\pi_{\mathbf{w}}(\mathbf{x}_k)$, and policy, $\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)$, respectively.

3. Problem formulation

3.1. Equations of motion

The force $\mathbf{F}^B$ and torque $\mathbf{L}^B$ in the lander's body frame for a given commanded thrust depends on the placement of the thrusters in the lander structure. We can describe the placement of each thruster through a body-frame direction vector $\mathbf{d}$ and a position vector $\mathbf{r}$, both in $\mathbb{R}^3$. The direction vector is a unit vector giving the direction of the body frame force that results when the thruster is fired. The position vector gives the body frame location where the force resulting from the thruster firing is applied for the purposes of computing torque. For a lander with k thrusters, the body frame force and torque associated with one or more thrusters firing is given by,

$$\mathbf{F}^B = \sum_{i=1}^{k} \mathbf{d}^{(i)} T^{(i)}_{cmd} \tag{13a}$$

$$\mathbf{L}^B = \sum_{i=1}^{k} \mathbf{r}^{(i)} \times \mathbf{F}^{(i)}_B \tag{13b}$$

where $T_{cmd_i} \in [T_{min}, T_{max}]$ is the commanded thrust for thruster i, T_{min} and T_{max} are a thruster's minimum and maximum thrust, $\mathbf{d}^{(i)}$ is the direction vector for thruster i, and $\mathbf{r}^{(i)}$ is the position of thruster i. The total body frame force and torque are calculated by summing the individual forces and torques. The dynamics model uses the lander's current attitude $\mathbf{q}$ to convert the body frame thrust vector to the inertial frame, given by

$$\mathbf{F}^N = A_B^N(\mathbf{q})^T \mathbf{F}^B \tag{14}$$

where $A_B^N(\mathbf{q})$ is the direction cosine matrix mapping the inertial frame to body frame obtained from the current attitude parameter $\mathbf{q}$. The attitude matrix is related to the quaternion by the following expression (Shuster, 1993):

$$A_B^N(\mathbf{q}) = \Xi^T(\mathbf{q})\Psi(\mathbf{q}) \tag{15}$$

where

$$\Xi(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times 3} + [\boldsymbol{\varrho}\times] \\ -\boldsymbol{\varrho}^T \end{bmatrix} \tag{16a}$$

$$\Psi(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times 3} - [\boldsymbol{\varrho}\times] \\ -\boldsymbol{\varrho}^T \end{bmatrix} \tag{16b}$$

where the quaternion is divided into the $\boldsymbol{\varrho}$ and q_4, the vector and scalar components, respectively, and $\mathbf{q}^T = [\boldsymbol{\varrho} \quad q_4]^T$. The rotational velocities $\boldsymbol{\omega}_{B/N}$ are then obtained by integrating the Euler rotational equations of motion, as follows Junkins and Schaub (2009):

$$\mathbf{J}\dot{\boldsymbol{\omega}}_{B/N} = -[\boldsymbol{\omega}_{B/N}\times]\mathbf{J}\boldsymbol{\omega}_{B/N} + \mathbf{L}^B + \mathbf{L}_{\text{env}}^B \tag{17}$$

with

$$[\mathbf{a}\times] \equiv \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{18}$$

for any general 3×1 vector $\mathbf{a}$ defined such that $[\mathbf{a}\times]\mathbf{b} = \mathbf{a} \times \mathbf{b}$. The vector $\mathbf{L}^B$ is the body frame torque as given in Eq. (13b), $\mathbf{L}_{\text{env}}$ is the body frame torque from external disturbances, and $\mathbf{J}$ is the lander's inertia tensor. The lander's attitude is then updated by integrating the differential kinematic given by

$$\dot{\mathbf{q}} = \frac{1}{2}\Xi(\mathbf{q})\boldsymbol{\omega}_{B/N} \tag{19}$$

where the lander's attitude is parameterized using the quaternion representation and $\boldsymbol{\omega}_i$ denotes the i^{th} component of the rotational velocity vector $\boldsymbol{\omega}_{B/N}$. The translational motion is modeled as follows

$$\dot{\mathbf{r}} = \mathbf{v} \tag{20a}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{F}^N + \mathbf{F}_{\text{env}}^N}{m} + \mathbf{g} \tag{20b}$$

$$\dot{m} = -\frac{\sum_i^k \|\mathbf{F}^{B(i)}\|}{I_{\text{sp}} g_{\text{ref}}} \tag{20c}$$

where $\mathbf{F}^{N(i)}$ is the inertial frame force as given in Eq. (14), k is the number of thrusters, $g_{\text{ref}} = 9.8$ m/s^2, $\mathbf{g} = [0 \quad 0 \quad -3.7114]$m/s^2 is used for Mars, $I_{\text{sp}} = 225$ s, and the spacecraft's mass is m. $\mathbf{F}_{\text{env}}$ is a vector of normally distributed random variables representing environmental disturbances such as wind and variations in atmospheric density. The random force vector $\mathbf{F}_{\text{env}}$ is used during training to increase the policy's robustness to external disturbances. It should be noted that, although $\mathbf{F}_{\text{env}}$ is modeled as a normally distributed random variable for this work, a random-walk stochastic framework is a more realistic disturbance model for dynamics systems. As a rough estimate of the force caused by wind gusts, a cross-sectional lander area of 100 square feet was assumed, and we used the Cornell University wind pressure calculator to calculate the resulting wind pressure in pounds per square foot, divided this by 168 to account for the difference in average surface air pressure between the two planets, multiplied by the assumed cross-section of the lander, and then converted the force to Newtons. Using this rough approximation, a wind of 100 m/s would result in a force of 162 N. Note that the Mars GRAM model shows a strong jet near 5-km altitude and 70 degrees N latitude with winds reaching 100 m/s, so it is reasonable to expect a Lander to be able to handle this case.

For purposes of modeling the lander's moments of inertia, we model the lander as a uniform density ellipsoid, with inertia matrix given by

$$\mathbf{J} = \frac{m}{5}\begin{bmatrix} b^2 + c^2 & 0 & 0 \\ 0 & a^2 + c^2 & 0 \\ 0 & 0 & a^2 + b^2 \end{bmatrix} \tag{21}$$

where a, b, and c correspond to the body frame x, y, and z axes. m is the lander's mass, which is updated as shown in Eq. (20c). We assume the lander has a wet mass of 2000 kg and four throttleable thrusters with a minimum and maximum thrust magnitude of 1000 N and 5000 N respectively. The four thrusters are located in the lander body frame as shown in Table 2, where x, y, and z are the body frame axes. Roll is about the x-axis, yaw is about the z-axis, and pitch is about the y-axis. Note that this thruster configuration does not allow any direct control of the rotational velocity around the z-axis. However, the lander's yaw will change during the trajectory, but due to coupling with pitch via roll rather than due to torque caused by thrust. Direct yaw control could be implemented by positioning the thrusters at a slight angle to the body-frame z-axis, which

Table 2
Body frame thruster locations.

Thruster	x (m)	y (m)	z (m)
1	0	−2	−1
2	0	2	−1
3	−2	0	−1
4	2	0	−1

might result in faster convergence of policy optimization. The lander's translational motion is described using a target-centered inertial reference frame. The navigation system provides updates to the guidance system every 0.2 s, and we integrate the equations of motion using fourth order Runge-Kutta integration with a time step of 0.05 s.

3.2. Problem statement

Consider the powered descent problem on a large planetary body such as Mars. In this section we describe the RL set-up that is employed to find a 6-DOF closed-loop policy that generates quasi-fuel optimal trajectories capable of (a) driving the lander to the desired location to the planetary surface with pin-point accuracy, and (b) satisfying flight and systems constraints (e.g. glide slope, thrust and attitude constraints). To set the stage for the problem formulation in the RL framework, consider the standard formulation for a typical trajectory optimization problem (see for example Acikmese and Ploen, 2007):

Minimum-Fuel Problem: Find the thrust program and flight time that minimize the following cost functions:

$$\max_{t_f,\mathbf{T}} \quad m_L(t_f) = \min_{t_f,\mathbf{T}} \int_0^{t_f} \|\mathbf{T}\| dt \tag{22}$$

subject to the following constraints (equations of motion):

$$\ddot{\mathbf{r}} = \mathbf{g} + \frac{\mathbf{T}}{m} \tag{23a}$$

$$\dot{m} = \frac{\|\mathbf{T}\|}{I_{\text{sp}} g_{\text{ref}}} \tag{23b}$$

and the following boundary conditions:

$$\mathbf{r}(0) = \mathbf{r}_0 \tag{24a}$$

$$\mathbf{v}(0) = \dot{\mathbf{r}}(0) = \mathbf{v}_0 \tag{24b}$$

$$\mathbf{r}(t_f) = \mathbf{r}_f \tag{24c}$$

$$\mathbf{v}(t_f) = \dot{\mathbf{r}}(t_f) = \mathbf{v}_f \tag{24d}$$

And additional flight (glide slope) and thrust constraints:

$$0 < T_{min} < \|\mathbf{T}\| < T_{max} \tag{25a}$$

$$\theta_{\text{alt}} = \arctan\left(\frac{\sqrt{r_y(t)^2 + r_z(t)^2}}{r_x(t)}\right) < \tilde{\theta}_{alt} \tag{25b}$$

Generally, one uses an inertial target-centered reference frame that neglects the rotation of Mars. Since the powered descent phase for a typical Mars mission is initiated at an altitude that is low w.r.t. the planet's radius, the gravity $\mathbf{g}$ is assumed to be constant (i.e., motion in a uniform gravitational field). The lander's position is given by the vector $\mathbf{r} \in \mathbb{R}^3$ and $\mathbf{r}^T = [r_x \quad r_y \quad r_z]^T$, where r_x, r_y, and r_z are the lander's downrange, crossrange, and altitude in the target centered reference frame. Likewise, the velocity is described by a vector $\mathbf{v} \in \mathbb{R}^3$ and $\mathbf{v}^T = [v_x \quad v_y \quad v_z]^T$, where v_x, v_y, and v_z are the lander's downrange, crossrange, and descent components, respectively. The vectors $\mathbf{r_0}, \mathbf{v_0}$, and $\mathbf{r_f}, \mathbf{v_f}$ are the initial and final position/velocity vetors, respectively. Importantly, the thrust magnitude $\|\mathbf{T}\|$ is limited between a maximum and minimum. The angle $\tilde{\theta}_{\text{alt}} < \frac{\pi}{2}$ represents the glide slope constraint. This is cast as a typical optimal control problem and generally solved using numerical methods (e.g. Acikmese and Ploen, 2007; Lu, 2017). For a 6-DOF problem, one needs to provide additional information about the attitude of the lander. The body frame is defined with the z-axis passing vertically through the lander and orthogonal x and y axes completing the body reference frame. Although a more formal 6-DOF trajectory and attitude optimal control problem can be done (e.g. Szmuk and Acikmese, 2018), the RL framework is defined such that the agent (i.e., the lander) responds to a single cost (reward) signal which needs to be generally minimized (maximized) during the search process. Indeed, with the goal of the lander arriving at the origin of the target-centered reference frame at some specified velocity vector $\mathbf{v} = \dot{\mathbf{r}}$ and at a specified attitude, we can define the RL-based landing problem as follows:

RL 6-DOF Closed-Loop Landing Problem:
Minimize:

1. Terminal Position Error: $\|\mathbf{r}\|$ at $t = t_f$
2. Terminal Velocity Error: $\|\mathbf{v}\|$ at $t = t_f$
3. Terminal Attitude Error: Magnitude of Pitch ϕ and Roll θ at $t = t_f$
4. Terminal Rotational Velocity Error: $\|\boldsymbol{\omega}\|$ at $t = t_f$
5. Control Effort: $\sum \|\mathbf{T}\|$ where the sum is over a trajectory and T is the total thrust

Subject to:

1. Terminal Glideslope: $\arctan(\|v_z\|/\|v_{x,y}\|) > 79$ degrees over final 2 m of descent (soft constraint)
2. Attitude Constraints: Magnitude of Pitch ϕ and Roll θ less than 80 degrees (hard constraint)
3. Equations of Motion (set by the environment)

Here, hard constraints are imposed by terminating the episode with a negative reward, whereas soft constraints are encouraged through rewards but without premature termination of the episode.

3.3. Implementation details

In order to facilitate reproduction of our results, we include in this section several techniques we used in our implementation. We adjust the clipping parameter ϵ from Eq. (8) to target a KL divergence between policy updates of 0.001, which has worked well in other RL problems (Schulman et al., 2017). The policy and value function is learned concurrently, as the estimated value of a state is policy dependent. We use a Gaussian distribution with mean $\pi_{\theta}(\mathbf{u}_k|\mathbf{x}_k)$ and a diagonal covariance matrix for the

action distribution in the policy. Because the log probabilities are calculated using the exploration variance, the degree of exploration automatically adapts during learning such that the objective function is maximized.

We use the ADAM optimizer (Kingma and Ba, 2014) to adjust the learning rate for both the policy and value function networks. We use an approximation to the KL divergence for a Gaussian that is given by the mean square difference between the pre- and post-log probabilities of policy actions. This approximation is close to the exact KL divergence and is used in the open AI's baseline PPO implementation which is referred to as PPO2 (Schulman et al., 2017).

$$\epsilon = \begin{cases} \min(\epsilon_{max}, 1.5\epsilon) & \text{if } \mathrm{kl} < \frac{1}{2}\mathrm{kl}_{\mathrm{targ}} \\ \max(\epsilon_{min}, \frac{1}{1.5}\epsilon) & \text{if } \mathrm{kl} > 2\mathrm{kl}_{\mathrm{targ}} \end{cases} \tag{26}$$

$$\zeta = \begin{cases} 1.5\zeta & \text{if } \mathrm{kl} < \frac{1}{2}\mathrm{kl}_{\mathrm{targ}} \text{ and } \epsilon > \frac{1}{2}\epsilon_{\max} \text{ and } \zeta < \zeta_{\max} \\ \frac{1}{1.5}\zeta & \text{if } \mathrm{kl} > 2\mathrm{kl}_{\mathrm{targ}} \text{ and } \epsilon < 2\epsilon_{\min} \text{ and } \zeta > \zeta_{\min} \end{cases} \tag{27}$$

We adjust the clipping parameter ϵ based off of the KL divergence between policy updates as shown in Eq. (26), where kl is the KL divergence between policy updates, $\mathrm{kl}_{\mathrm{targ}}$ is the target KL divergence, and $\epsilon_{\max}$ and $\epsilon_{\min}$ are set to 0.5 and 0.01. A similar implementation was suggested in Reference Schulman et al. (2017). We also adjust the ADAM step size by multiplying it by a parameter ζ as shown in (27), where $\zeta_{\min}$ and $\zeta_{\max}$ are set to 0.1 and 10. A learning rate adjustment was used in Schulman et al. (2017) for the roboschool environments, but the exact implementation was not divulged.

When approximating functions using artificial neural networks, it is important to scale the inputs to avoid saturating the activation functions at each layer. Most implementations scale the network input using statistics calculated over a given rollout. Specifically, each element of an input vector is scaled by first subtracting the mean of the element and then dividing by three times the standard deviation, with the statistics calculated over a batch of rollouts. A rollout is a batch of episodic trajectories collected by the agent interacting with the environment that is used to update the policy and value function. We use a variation on this approach that adjusts the statistics used for scaling incrementally over the entire optimization, which boosts performance by avoiding discontinuities in the scaling statistics. It is also important to ensure that the magnitude of the neural network outputs are reasonably close to unity, although full normalization is not required. For policy parameter learning, we scale the action such that maximum thrust for an engine corresponds to a value of one. For value function parameter learning, we use a heuristic that multiplies the rewards accumulated over an episode by a factor of $1-\gamma$.

3.4. RL problem formulation

In order to apply the reinforcement learning framework developed in Section 2.3 to a particular problem, we need to define an environment and reward function and specify the policy and value function network architectures. To test the policy, the trained policy is substituted for the agent. The action taken by the agent based on the observation provided by the dynamics model is interpreted as a thrust command by the thruster model, which passes a body frame force and torque to the dynamics model, which computes the next state. An episode terminates when the lander's altitude falls below zero or the attitude constraint (the only hard constraint) is violated. The initial condition generator generates random initial conditions for the lander with values uniformly distributed between the minimum and the maximum values given in Table 3, which corresponds to a 4 km^2 deployment ellipse. Although a 4 km^2 deployment ellipse may seem unrealistically small for the Mars powered descent phase application, the reduced range of initial conditions results in a simpler problem that allows us to iterate quickly over different hyperparameter settings in the reward function. Later in this work, we extend the deployment ellipse to 9 km^2.

To speed up the design process, we developed a 3-DOF RL environment where we could quickly assess the impact of different reward functions and hyperparameter settings. A policy optimized in the 3-DOF RL environment converges around 70 times faster than in the 6-DOF environment. The reward function for the 3-DOF environment is identical to that of the 6-DOF environment. We show optimization learning curves for the 3-DOF environment to allow comparison. The 3-DOF results are also used to compare performance with the integrated guidance and control system, where for a perfectly integrated guidance and control system, we expect performance similar to the 3-DOF case.

The policy and value functions are implemented using four-layer neural networks with tanh activations on each hidden layer. The network architectures are as shown in

Table 3
Lander initial conditions for optimization.

Parameter	min	max
Downrange Position (m)	0	2000
Crossrange Position (m)	−1000	1000
Elevation Position (m)	2300	2400
Downrange Velocity (m/s)	−70	−10
Crossrange Velocity (m/s)	−30	30
Elevation Velocity (m/s)	−90	−70
Yaw (deg)	−22.5	22.5
Pitch (deg)	22.5	67.5
Roll (deg)	−22.5	22.5
Rotational Velocity Yaw Axis (deg/s)	0.00	0.00
Rotational Velocity Pitch Axis (deg/s)	−0.60	0.60
Rotational Velocity Roll Axis (deg/s)	−0.60	0.60

Table 4
Policy and value function network architecture.

	Policy Network		Value Network	
Layer	# units	Activation	# units	Activation
Hidden 1	$10 * \text{obs\_dim}$	tanh	$10 * \text{obs\_dim}$	tanh
Hidden 2	$\sqrt{n_{h1} * n_{h3}}$	tanh	$\sqrt{n_{h1} * n_{h3}}$	tanh
Hidden 3	$10 * \text{act\_dim}$	tanh	5	tanh
Output	act_dim	Linear	1	Linear

Table 4, where n_{hi} is the number of units in layer i, obs_dim is the observation dimension, and act_dim is the action dimension.

The most challenging part of solving the Mars landing problem using RL was the development of a reward function that works well in a sparse reward setting. If we only reward the agent for making a soft pinpoint landing at the correct attitude and with close to zero rotational velocity, the agent would never see the reward within a realistic number of episodes, as the probability of achieving such a landing using random actions in a 6-DOF environment with realistic initial conditions is exceedingly low. The sparse reward problem is typically addressed using inverse reinforcement learning (Ng et al., 2000), where a per time step reward function is learned from expert demonstrations. With a reward given at each step of agent-environment interaction, the rewards are no longer sparse. In theory, demonstrations using optimal control could provide trajectories for inverse RL, but this would be very computationally expensive for 6-DOF trajectories, and many optimal control packages have trouble with complex non-convex and/or non-differentiable constraints.

Instead, we chose a different approach, where we engineer a reward function that, at each time step, provides hints to the agent (referred to as "shaping rewards") that drive it towards a soft pinpoint landing. The recommended approach for such a reward shaping function is to make the reward a difference of potentials, in which case theoretical results have shown that the additional reward does not change the optimal policy (Ng, 2003). However, we were unable to find a potential function that worked well with the difference of potentials approach.

Our solution was to choose a shaping reward that keeps the agent's velocity vector aligned with the line of sight vector. Such a shaping reward results in a pinpoint, but not necessarily soft, landing. To achieve the soft landing, the agent estimates time-to-go as the ratio of the range and the magnitude of the lander's velocity and reduces the targeted velocity as time-to-go decreases. It is also important that the lander's terminal velocity be directed predominantly downward, the lander's terminal attitude is upright, and there are negligible terminal rotational velocity components. To achieve these requirements, we use the piecewise reward shaping function given in Eqs. (28a)–(28e), where τ_1 and τ_2 are hyperparameters and v_o is set to the magnitude of the lander's velocity at the start of the powered descent phase. We see that the shaping rewards take the form of a velocity field that maps the lander's position to a target velocity. In words, we target a location 15 m above the desired landing site and target a z-component of landing velocity equal to −2 m/s. Below 15 m, the downrange and crossrange velocity components of the target velocity field are set to zero, which encourages a vertical descent. Targeting a vertical descent has the beneficial side effect of encouraging the agent to keep the attitude level with no rotational velocity. This results in a good landing attitude with small rotational velocity components and a velocity directed primarily downward.

$$\mathbf{v}_{targ} = -v_o\left(\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|}\right)\left(1-\exp\left(-\frac{t_{go}}{\tau}\right)\right) \tag{28a}$$

$$t_{go} = \frac{\|\hat{\mathbf{r}}\|}{\|\hat{\mathbf{v}}\|} \tag{28b}$$

$$\hat{\mathbf{r}} = \begin{cases} \mathbf{r} - [0 \quad 0 \quad 15], & \text{if } r_z > 15 \\ [0 \quad 0 \quad r_z], & \text{otherwise} \end{cases} \tag{28c}$$

$$\hat{\mathbf{v}} = \begin{cases} \mathbf{v} - [0 \quad 0 \quad -2], & \text{if } r_z > 15 \\ \mathbf{v} - [0 \quad 0 \quad -1], & \text{otherwise} \end{cases} \tag{28d}$$

$$\tau = \begin{cases} \tau_1, & \text{if } r_z > 15 \\ \tau_2, & \text{otherwise} \end{cases} \tag{28e}$$

Finally, we provide a terminal reward bonus when the lander reaches an altitude of zero, and the terminal position, velocity, attitude, and rotational velocity are within specified limits. The reward function is then given by the following:

$$\begin{aligned} r \quad &= \alpha\|\mathbf{v} - \mathbf{v}_{targ}\| + \beta\|\mathbf{F}_B\| + \gamma \text{any}(\mathbf{q}(t) > \mathbf{q}_{lim}) \\ &+\delta\sum_{i=1}^{3} -\max(0, \mathbf{q}_i - \mathbf{q}_{mgn_i}) + \eta \\ &+\kappa(r_z < 0 \text{ and } \|\mathbf{r}\| < r_{lim} \text{ and } \|\mathbf{v}\| < v_{lim} \\ &\text{and all}(\mathbf{q} < \mathrm{q}_{\mathrm{lim}}) \text{ and all}(\boldsymbol{\omega} < \omega_{\mathrm{lim}} \text{ and } gs < gs_{lim})) \end{aligned} \tag{29}$$

where the various terms are described in the following:

1. α weights a term penalizing the error in tracking the target velocity (negative reward).
2. β weights a term penalizing control effort (negative reward).
3. γ weights a term penalizing exceeding yaw, pitch, and roll limits. The "any" function is set to Boolean True if any elements of its argument are Boolean True (just like the Python np.any() function). If any attitude component exceeds its limit, the episode is terminated. (negative reward)
4. δ weights a term that increases as the lander's attitude (Euler 321) passes a threshold q_{mgn}; this gives the agent a hint that it is approaching the attitude limit q_{lim} (negative reward)
5. η is a constant positive term that encourages the agent to keep making progress along the trajectory. Since all other rewards are negative, without this term, an agent would be incentivized to violate the attitude constraint

and prematurely terminate the episode to maximize the total discounted rewards received starting from the initial state. (positive reward)

6. κ is a bonus given for a successful landing, where terminal position, velocity, attitude, rotational velocity, and glideslope gs are all within specified limits. The "all" function is set to Boolean True if all elements of its argument are Boolean True (just like the Python np.all() function). The limits are $r_{lim} = 5$ m, $v_{lim} = 2m/s$ m, $q_{lim} = 0.2$ rad (except for yaw, which is not limited), $gs_{lim} = 79$ degrees, and $w_{lim} = 0.2$ rad/s. (positive reward)

This reward function allows the agent to trade off between tracking the target velocity given in Eq. (28a), conserving fuel, satisfying the attitude constraints, and maximizing the reward bonus given for a good landing. Note that the constraints are not hard constraints such as might be imposed in an optimal control problem solved using collocation methods. However, the consequences of violating the constraints (a large negative reward and termination of the episode) are sufficient to ensure they are not violated once learning has converged. Hyperparameter settings and coefficients used in this work are given in Table 5, note that due to lander symmetry, we do not impose any limits on the lander's yaw. There is a certain amount of intuition that goes into choosing some of these hyperparameter settings, as is generally the case when applying reinforcement learning to complex problems. In this work, it took a few hours of thought to determine initial values, and around a week of simulation to fine-tune the values.

The observation given to the agent during learning and testing is given by

$$\text{obs} = [\,\mathbf{v}_{\text{error}} \quad \mathbf{q} \quad \boldsymbol{\omega} \quad r_z \quad t_{\text{go}}\,] \tag{30}$$

where $\mathbf{v}_{\text{error}} = \mathbf{v} - \mathbf{v}_{\text{targ}}$, with $\mathbf{v}_{\text{targ}}$ given in Eq. (28a), the lander's estimated altitude, the time-to-go, as well as an estimate of the lander's attitude ($\mathbf{q}$) and rotational velocity ($\boldsymbol{\omega}$). Note that aside from the altitude, the lander translational coordinates do not appear in the observation. This results in a policy with good generalization in that the policy's behavior can extend to areas of the full state space that were not experienced during learning. To provide a robust final policy, we optimize with parameter uncertainty.

Table 5
Hyperparameter settings.

Parameter	Value
v_o (m/s)	$\|\mathbf{v}_o\|$
τ_1 (s)	20
τ_2 (s)	100
α	−0.01
β	−0.05
γ	−100
δ	−20
η	0.01
κ	10
$\mathbf{q}_{lim}$ (deg)	[360 80 80]
$\mathbf{q}_{mgn}$ (deg)	[360 67 67]

Specifically, at the beginning of each episode both the lander's initial mass and the acceleration due to gravity are perturbed to give a random value within 5% and 2% of nominal, respectively. In addition, we apply random uniform noise to the inertia tensor at the beginning of each episode. There is no physical significance to the inertia tensor noise; it is just a method of introducing parameter uncertainty in order to learn a robust policy.

It turns out that the when a terminal reward is used (as we do), it is advantageous to use a relatively large discount rate. However, it is also advantageous to use a lower discount rate for the shaping rewards. To our knowledge, a method of resolving this conflict has not been reported in the literature. In this work, we resolve the conflict by introducing a framework for accommodating multiple discount rates. Let γ_1 be the discount rate used to discount $r_1(k)$, the reward function term in Eq. (29) that is associated with the κ coefficient. Moreover, let γ_2 be the discount rate used to discount $r_2(k)$, the sum of all other terms in the reward function. We can then rewrite the Eq. (10) and (9) in terms of these rewards and discount rates, as given by the following:

$$J(\mathbf{w}) = \sum_{i=1}^{M} \left(V_{\mathbf{w}}^{\pi}(\mathbf{x}_k^i) - \left[\sum_{l=k}^{T} \gamma_1^{\ell-k} r_1(\mathbf{u}_\ell^i, \mathbf{x}_\ell^i) + \gamma_2^{\ell-k} r_2(\mathbf{u}_\ell^i, \mathbf{x}_\ell^i) \right] \right)^2 \tag{31a}$$

$$A_{\mathbf{w}}^{\pi}(\mathbf{x}_k, \mathbf{u}_k) = \left[\sum_{\ell=k}^{T} \gamma_1^{\ell-k} r_1(\mathbf{u}_\ell, \mathbf{x}_\ell) + \gamma_2^{\ell-k} r_2(\mathbf{u}_\ell, \mathbf{x}_\ell) \right] - V_{\mathbf{w}}^{\pi}(\mathbf{x}_k) \tag{31b}$$

Although the approach is simple, the performance improvement is significant, despite the fact that we had to give up the use of generalized advantage estimation (Schulman et al., 2015b) as it is not compatible with multiple discount rates. Without the use of multiple discount rates, the performance was actually worsened by including the terminal reward term.

4. Results

Code to reproduce results can be found at: github.com/Aerospace-AI/AAS-18-290-6DOF

4.1. Policy optimization

Rollouts are generated by the agent interacting with the environment for 120 episodes, with the resulting trajectories used to compute the advantages and update the value and policy function approximators. Optimization was terminated after 300,000 episodes. Learning curves are shown in Figs. 1–5. In Fig. 1 (Fig. 4 for 3-DOF optimization), we plot statistics for the undiscounted rewards over 120 episodes, which is the number of episodes the agent accumulates before updating the policy and value function. "Steps" refers to the number of interactions between the

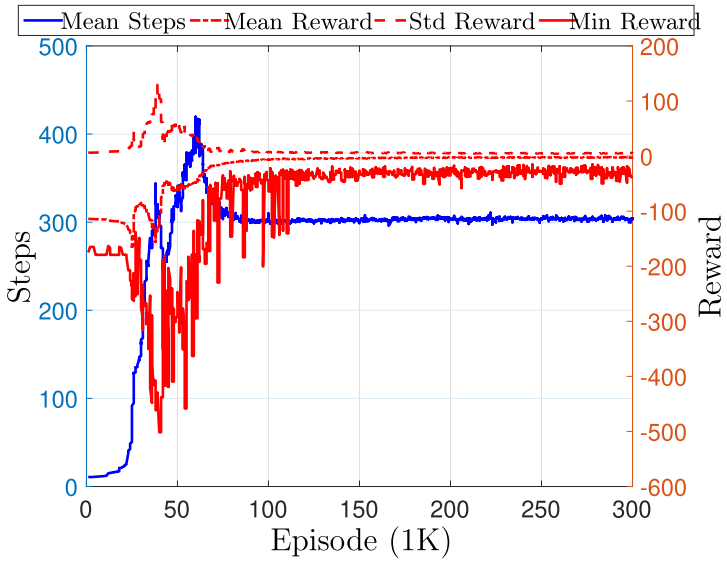


Fig. 1. 6-DOF Optimization: Learning curves of mean number of steps per 1000 episodes (left vertical axis), mean reward (right vertical axis), standard deviation of reward, and minimum reward.

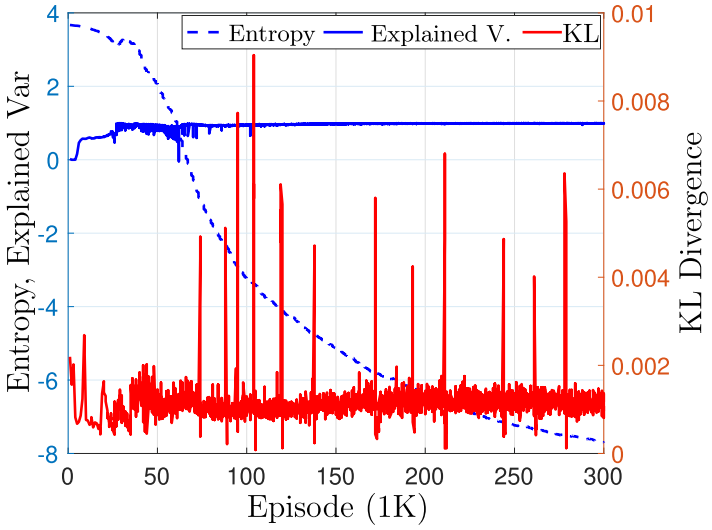


Fig. 2. 6-DOF Optimization: Learning statistics of KL, Entropy, and Explained Variance versus episodes.

agent and environment for the episode; this can be converted to the trajectory duration by multiplying by the navigation period of 0.2s. Fig. 2 (Fig. 5 for 3-DOF optimization) gives the policy entropy and the KL divergence between policy updates. We also plot the explained variance as a measure of how well the value function explains the observed returns; if the explained variance is 1, the value function perfectly explains the observed returns, if it is less than zero, the value functions predictive ability is less than that of predicting constant value for the value of being in any state. Specifically, we calculate the explained variance as shown below in Eq. (32), where y is the actual sum of discounted rewards over the batch of trajectories (the target for the value function approximator) and $\hat{y}$ is the predicted sum of discounted rewards as given by the value function approximator.

$$\exp\_\text{var} = 1 - \frac{\sigma_y^2 - \hat{\sigma}_y^2}{\sigma_y^2} \tag{32}$$

Fig. 3(a) and (b) (Fig. 6(a) and (b) for 3-DOF) plot the lander's end of episode position and velocity magnitudes as learning progresses, where again the statistics are accumulated over the 120 episodes used for the policy and value function updates. Fig. 3 (In Fig. 6 for the 3-DOF optimization), we plot the mean and standard deviation of the lander's position, velocity, attitude, and angular velocity at end of episode computed over the 120 episodes. Note in Fig. 3(c) and (d) that the standard deviation of the attitude and rotational velocity seems inconsistent with a good landing. This is partly due to the parameter uncertainty introduced during learning, and partly due to policy exploration. However, during the execution of the learned policy

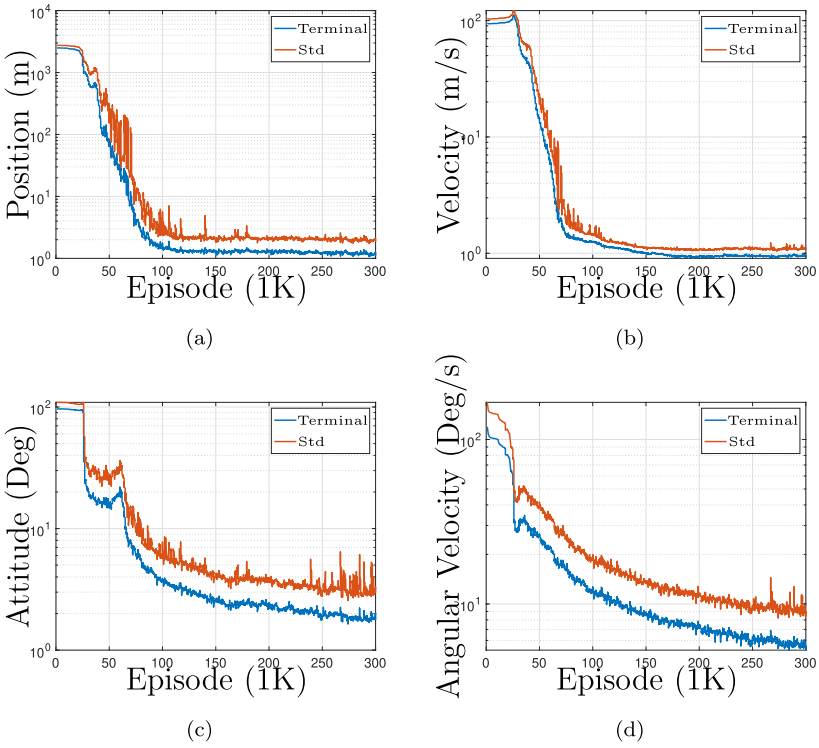


Fig. 3. 6-DOF optimization: lander position, velocity, attitude, and angular velocity at end of episode. (a) Norm position, (b) norm velocity, (c) norm attitude, (d) norm angular velocity.

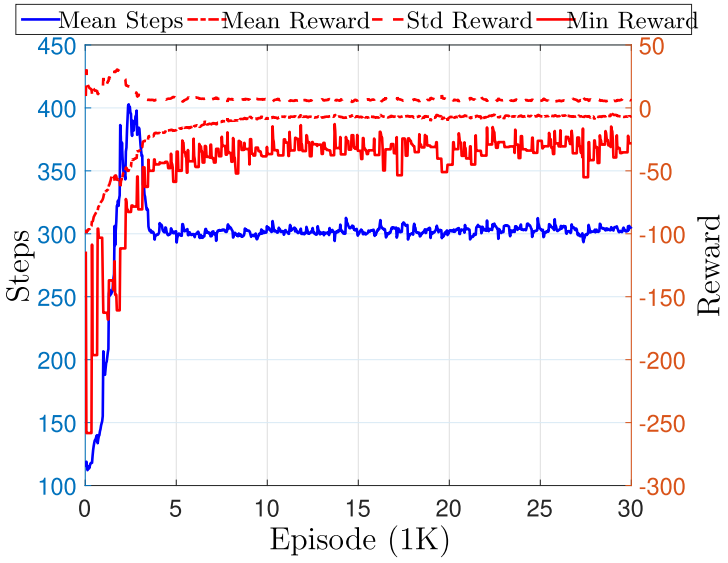


Fig. 4. 3-DOF Optimization: learning curves of mean number of steps per 1000 episode (left vertical axis), mean reward (right vertical axis), standard deviation of reward, and minimum reward.

a good landing solution is achieved. Although we terminated optimization at 300,000 episodes, the performance was still improving, and the standard deviation of the action was still around 4% for exploration. Since this is 4% of the maximum thrust, exploration noise still had a significant impact on the policy. During policy testing, we see that the magnitude of the attitude and rotational velocity is bounded more tightly.

4.2. Policy testing

To test the guidance policy, we simulate the policy for 10,000 episodes using the same initial conditions as used for policy optimization, as given in Table 3. During testing, the dynamics model adds Gaussian noise to the force acting on the lander. The noise has a mean that is computed at the start of each episode uniformly distributed between

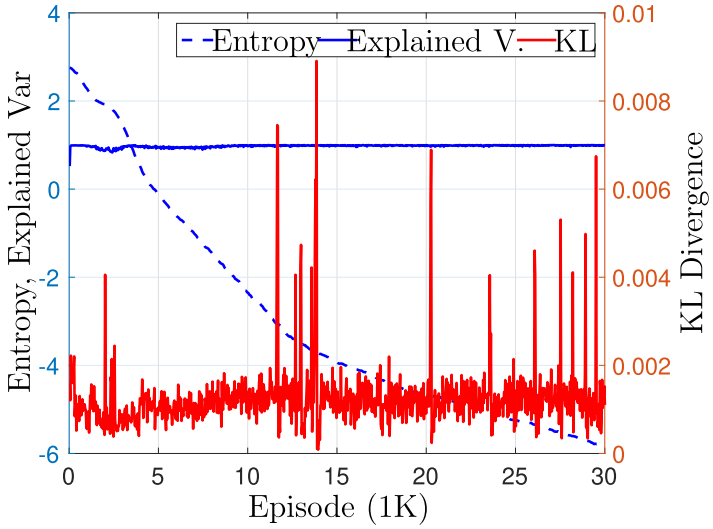
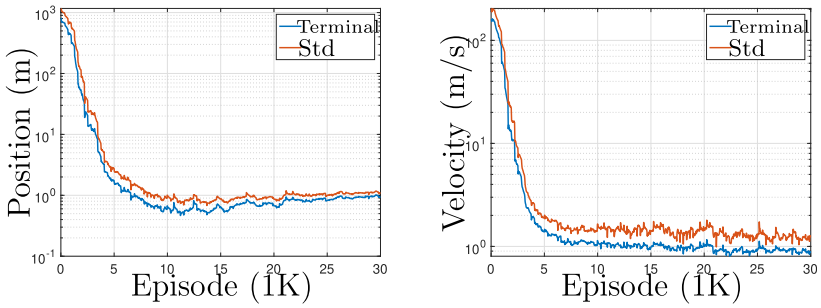


Fig. 5. 3-DOF optimization: learning statistics of KL, entropy, and explained variance versus episodes.



(a) (b)

Fig. 6. 3-DOF Optimization: lander position and velocity at end of episode. (a) Norm position, (b) norm velocity.

−100 and 100 N and held constant during the episode. The noise standard deviation is 100 N. At the start of each episode, the nominal wet mass of 2000 kg is set to a uniformly distributed value between ± 5% of nominal. The distributions for all the uncertain parameters that are randomly sampled during learning are shown in Table 6. Table 7 tabulates the touchdown statistics accumulated over testing. The glideslope statistic is given by the average $\arctan(v_z/\|[v_x, v_y]\|)$ over the final 2 m of descent; here a high value is desirable, as it indicates the lander's velocity is directed primarily in the downward direction. Note that due to symmetry in the lander design, we do not care about the terminal yaw value.

We compared the fuel efficiency of the 6-DOF RL agent to that of a 3-DOF RL agent using the same reward shaping function and to a 3-DOF controller using the energy-optimal closed-loop guidance algorithm as developed by Battin (1999), page 558 and D'Souza (1997) (here referred to as DR/DV algorithm). The fuel efficiency statistics are given at the bottom of Table 7. We use the DR/DV results as a proxy for optimal performance, as the algorithm is energy-optimal for the case of unlimited thrust (although we limit the thrust for the comparison). Because DR/DV has an unacceptable terminal glideslope (less than 45 degrees), we used a piecewise trajectory with a single waypoint 15 m above the landing site, similar to the approach we used for the RL policy, which achieved a minimum glideslope of 82.4 degrees. This added about 20 kg to the DR/DV fuel consumption. We see a 4% increase in fuel consumption for the 6-DOF RL agent as compared to 3-DOF DR/DV. To put this increase in perspective, note that tracking the DR/DV trajectory in a 6-DOF environment would certainly increase fuel consumption. Finally, note that the 6-DOF RL agent achieves fuel efficiency close to that of the 3-DOF RL agent; this tells us that the selected reward shaping function has a critical impact on fuel efficiency. There is certainly room for improvement here, as the exponential decrease in the magnitude of the

Table 6
Parameter uncertainty for policy optimization.

Parameter	min	max
Initial Mass (kg)	1900	2100
Grav. Acc. (m/s^2)	[0.07 0.07 3.64]	[−0.07 −0.07 3.79]
IT Diag (kg m^2)	−100	100
IT Off-Diag (kg m^2)	−10	10

Table 7
Touchdown statistics (same initial condition range as for optimization).

Parameter	Mean	SD	Min	Max
Downrange Position (m)	0.4	0.7	−3.0	4.5
Crossrange Position (m)	−0.1	0.7	−5.9	5.2
Downrange Velocity (m/s)	0.06	0.03	−0.06	0.14
Crossrange Velocity (m/s)	−0.01	0.04	−0.20	0.13
Elevation Velocity (m/s)	−0.93	0.08	−0.36	1.32
Pitch (rad)	−0.016	0.010	−0.063	0.033
Roll (rad)	−0.003	0.011	−0.038	0.066
Rot. Velocity – Roll (rad/s)	−0.000	0.021	−0.129	0.105
Rot. Velocity – Pitch (rad/s)	−0.005	0.013	−0.099	0.066
Rot. Velocity – Yaw (rad/s)	0.000	0.000	0.000	0.000
Glideslope (deg)	87.40	1.12	82.4	89.93
Fuel Consumed – 6-DOF RL (kg)	291	15	257	352
Fuel Consumed – 3-DOF RL (kg)	291	14	260	358
Fuel Consumed – 3-DOF DR/DV (kg)	279	14	233	335

target velocity is probably sub-optimal. Indeed, an optimal trajectory with an initial position far from the target landing site might accelerate towards the target to quickly close the downrange and crossrange distance in order to reduce trajectory time and use less fuel to maintain altitude.

Divert functionality was tested by running 5000 Monte Carlo simulations over the same initial conditions, but triggering a divert of 800 m downrange and 800 m crossrange when the lander reached an altitude of 1500 m. On average, the divert maneuver resulted in a 30 kg average increase in fuel consumption but otherwise did not impact performance.

A sample trajectory is plotted in Fig. 7, where x, y, and z are the downrange, crossrange, and altitude trajectory components in the target-centered reference frame. Thrust is shown in the inertial frame. The left side plot that is second from the top gives the altitude as a function of the norm of the cross range and downrange position. This plot is from initial conditions of 1500 m to −70 m/s downrange, −500 m to −30 m/s crossrange, and 2400 m to −90 m/s elevation. For comparison, we also plot (Fig. 8) a trajectory from the 3-DOF RL policy that starts from the same initial conditions. Note that there are a couple of places where the system exhibits small oscillations in the lander's thrust vector. If this were due to the policy falling into a local cost function minimum, further optimization may have allowed improvement, as the exploration variance was close to 0.2 when optimization was terminated. Also note that depending on the lander aerodynamics, a maximum angular velocity of close to 50 degrees per second may be too large.

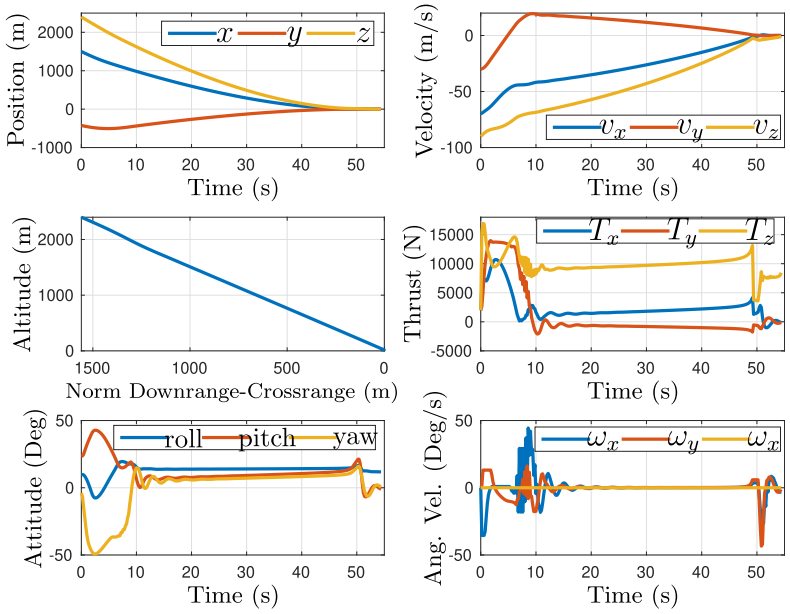


Fig. 7. 6-DOF sample trajectory.

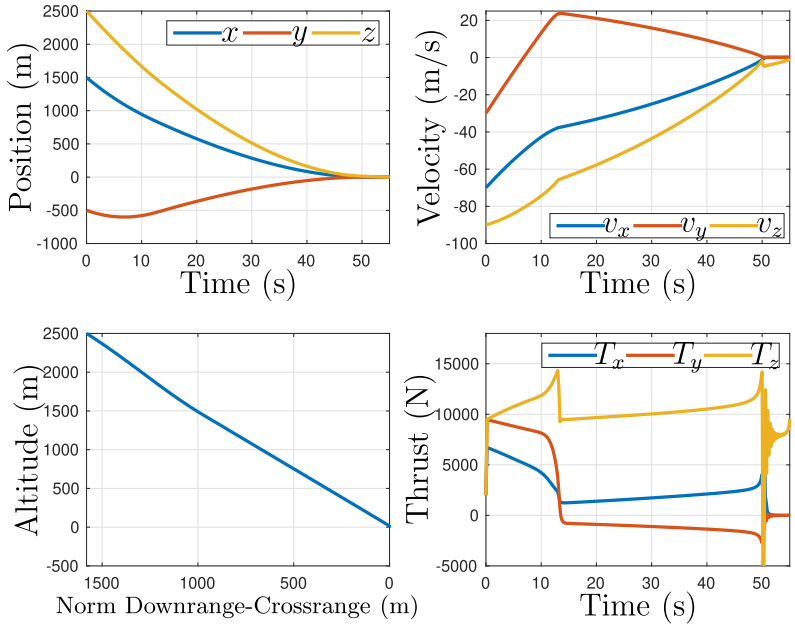


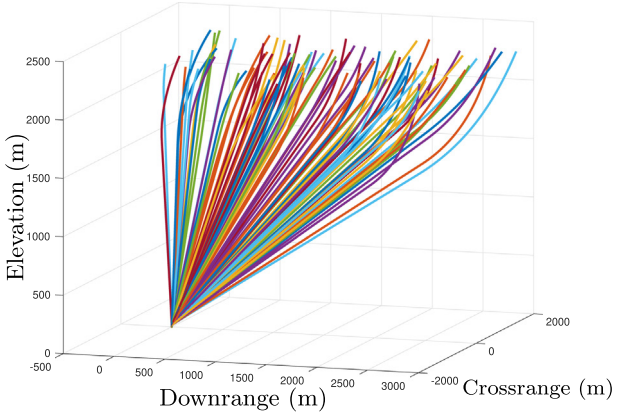
Fig. 8. Sampled 3-DoF trajectory using 3-DOF policy and same position and velocity as 6-DoF case.

Although in this work we only implemented an attitude constraint, it would be a simple matter to add a rotational velocity constraint as well. The apparent discontinuity during the last few seconds of a trajectory is due to the shaping function enforcing a vertical descent over the last 15 m.

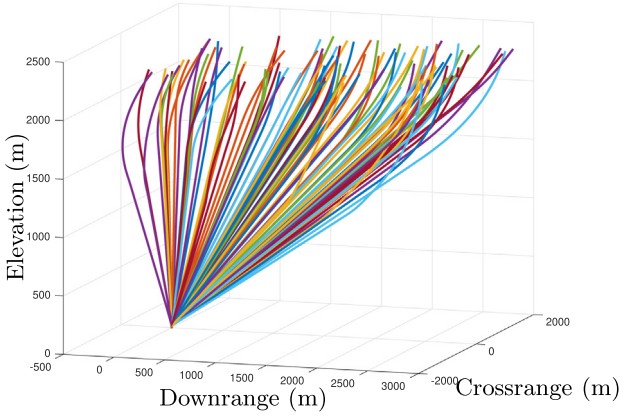
From Figs. 7 and 8, it is apparent that the 3-DOF and 6-DOF position and velocity trajectories are almost identical. The thrust differs toward the start and end of the landing due to the need to adjust attitude in the 6-DOF case in order to change the thrust direction, although overall the thrust trajectories are similar. Specifically, since the simulations begin with an average pitch of 45 degrees, the policy rotates the lander to a smaller pitch angle in order to allow a burn with the thrust vector pointing downwards to reduce the magnitude of the vertical velocity component. In addition, the policy must rotate the lander to allow a burn that reduces the crossrange velocity until the lander

Table 8
Lander extended initial conditions for optimization.

Parameter	min	max
9 km^2 deployment ellipse		
Downrange Position (m)	0	3000
Crossrange Position (m)	−1500	1500
Elevation Position (m)	2400	2500
Downrange Velocity (m/s)	−70	−10
Crossrange Velocity (m/s)	−30	30
Elevation Velocity (m/s)	−90	−70
12 km^2 deployment ellipse		
Downrange Position (m)	0	4000
Crossrange Position (m)	−1500	1500
Elevation Position (m)	2900	3100
Downrange Velocity (m/s)	0	4000
Crossrange Velocity (m/s)	−30	30
Elevation Velocity (m/s)	−90	−70



(a)



(b)

Fig. 9. 100 sample trajectories over 9 km^2 deployment ellipse. (a) 3-DOF trajectories, (b) 6-DOF trajectories.

is on a trajectory pointing towards the target. Towards the end of the trajectory, where the lander must transition to a vertical descent, there is another area where the thrust trajectories differ.

We also re-tested the policy over two extended deployment ellipses, as shown in Table 8, where we extend the deployment ellipse to 9 km^2, and again to 12 km^2. Here we use the same environmental noise and mass uncertainty as in the testing of the policy over the 4 km^2 deployment ellipse. Note that in order to obtain satisfactory performance over the 12 km^2 deployment ellipse, we had to raise the altitude of the deployment ellipse to 3000 m. Importantly, this is the same policy that was optimized over the initial conditions given in Table 3, i.e., a 4 km^2 deployment ellipse. Landing statistics over 20,000 episodes were similar to that given in Table 7, but average and maximum fuel consumption increased. Since the reward shaping function does not require the lander's full translational state, the policy generalizes quite well to regions of state space not experienced during optimization. Fig. 9(a) and (b) illustrate 100 randomly sampled trajectories using the 9 km^2 deployment ellipse for the 3-DOF and 6-DOF policies, respectively.

5. Comparison with GPOPS solution

Fig. 10 shows experimental results comparing the 3-DOF and 6-DOF agents to the solution provided by GPOPS (Rao et al., 2010), an optimal control solver using a 3-DOF problem formulation with the same initial conditions used to generate Figs. 7 and 8. The GPOPS solution, which also required a vertical descent over the final 15 m of altitude loss, had a fuel consumption of 250 kg, as compared to 290 kg for the 6-DOF and 3-DOF policies, making the RL policy fuel consumption 18 percent higher than GPOPS optimal. First, regarding fuel efficiency, note that the GPOPS trajectories are open loop, and when combined with a trajectory tracking controller the fuel consumption will be higher. Second, since the 6-DOF and 3-DOF policies have almost identical fuel consumption, we can attribute the difference in fuel efficiency between the RL policies and optimal to the reward shaping function used during optimization, which although effective, is not optimal. Indeed, we see in Fig. 10 that the RL policy trajectories are on average 20 s longer than the GPOPS trajectories. Moreover, the thrust profile does not conform to the typical "bang-bang" thrust magnitude associated with optimal trajectories. Consequently, future work to improve the fuel efficiency of the RL derived integrated guidance and control system should focus on the reward shaping function. Concretely, it may prove productive to learn an optimal velocity field function $f : \mathbf{r} \mapsto \mathbf{v}$ mapping the agent's position $\mathbf{r}$ to velocity $\mathbf{v}$. Specifically, we would use GPOPS to generate a large number of 3-DOF optimal trajectories over a range of initial conditions spanning the deployment ellipse, and collect a dataset of positions and associated velocities. A neural network would then learn

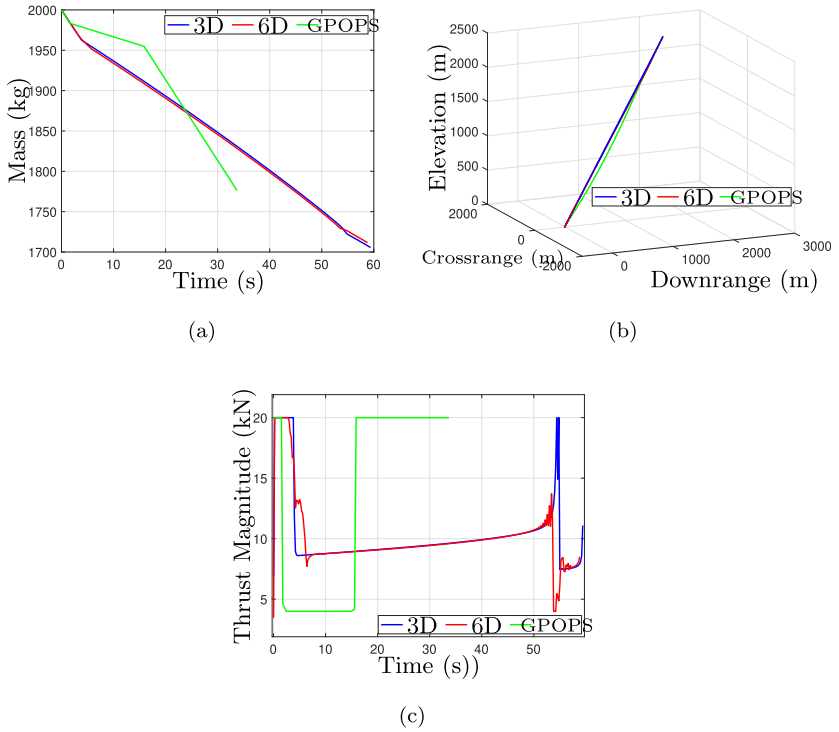


Fig. 10. Comparison of 3-DOF and 6-DOF policy with GPOPS solution. (a) Total mass performance comparison for 3-DOF, 6-DOF, and GPOPS, (b) trajectories for 3-DOF, 6-DOF, and GPOPS starting from same initial conditions, (c) thrust magnitude profiles performance comparison for 3-DOF, 6-DOF, and GPOPS.

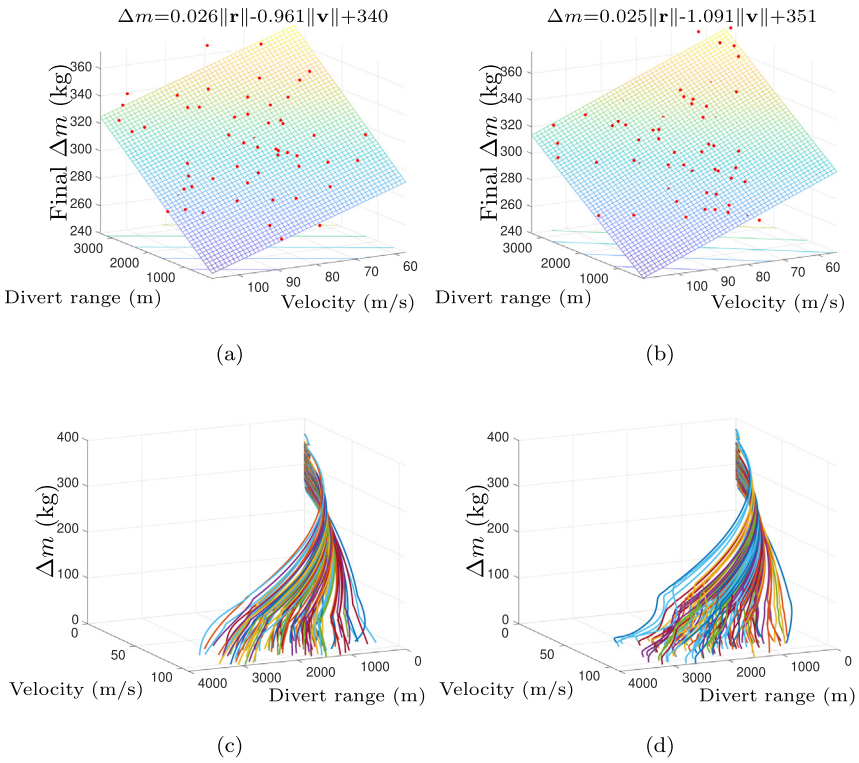


Fig. 11. 6-DOF and 3-DOF fuel-mass performance. (a) 3-DOF fuel-mass performance, (b) 6-DOF fuel-mass performance, (c) 3-DOF fuel-mass along the trajectories, (d) 6-DOF fuel-mass along the trajectories.

the optimal velocity field mapping from this dataset. Finally, during RL optimization, the trained velocity field neural network would output a target velocity vector which would replace $\mathbf{v}_{targ}$ in Eq. (28a).

Additionally, the comparison of the 3-DOF and 6-DOF agents in terms of fuel consumption is further investigated in Fig. 11. Fig. 11(a), (b), (c), and (d) show the fuel performance for the 3-DOF and 6-DOF policies over the 9 square kilometer deployment ellipse. Fig. 11(a), (b), (c), and (d) plot the difference between the initial mass and the final mass, $\Delta m = m(t_f) - m(t_o)$, as a function of the initial velocity and divert range. We expect that for larger divert ranges and larger initial velocities the fuel consumption will be larger, and Figs. 11(a) and (b) show these trends for both the trained 3-DOF and 6-DOF policy. It should be noted that once trained, these policies are able to provide a closed-loop integrated guidance and control solution that safely lands given a 9 square kilometer deployment ellipse and ranges of initial velocities shown in Fig. 11(a) and (b). The main takeaway from Fig. 11(a) and (b) is that the 3-DOF and 6-DOF policy have similar performance in terms of fuel consumption. However, investigating the fuel consumption along the trajectory (shown in Fig. 11(c) and (d) for the 3-DOF and 6-DOF cases, respectively), we see that the 6-DOF and 3-DOF policies initially deviate in terms of fuel consumption but then follow a similar trajectory. This is due to that fact that both policies have the same target velocity profiles. The values for the fuel performance are also summarized in Table 9.

Table 9
Extended range fuel consumption.

Parameter	Mean	SD	Max
9 km^2 deployment ellipse			
Fuel Consumed – 6-DOF RL Policy (kg)	308	25	412
Fuel Consumed – 3-DOF RL (kg)	309	25	400
Fuel Consumed – 3-DOF DR/DV (kg)	297	23	381
12 km^2 deployment ellipse			
Fuel Consumed – 6-DOF RL Policy (kg)	340	31	468
Fuel Consumed – 3-DOF RL (kg)	341	30	437
Fuel Consumed – 3-DOF DR/DV (kg)	319	27	414

On a 2.3Ghz processor, it takes less than 1mS for the 6-DOF policy to map an observation to an action; 3-DOF is faster at less than 0.2mS. On a flight processor running at 100 MHz, we would expect the 6-DOF mapping to increase to around 23mS. In this work, we used a navigation period of 0.2 s, so the policy mapping run time is not an issue. This is using the Tensorflow python API, further speedup would be possible by coding the guidance and control system using C++.

6. Conclusion

The intent of this paper was to demonstrate that reinforcement learning is a viable approach to developing an integrated guidance and control system for aerospace applications, although we have certainly not demonstrated all of

the potential benefits of the reinforcement learning framework (see Table 1) in this work. A second contribution of this work was to introduce the use of different discount rates for shaping and terminal rewards, which we found significantly enhances performance. A policy was optimized in a 6-DOF Mars powered descent phase environment. The trained policy was validated through Monte Carlo simulation, and we demonstrated the ability to achieve pinpoint accuracy and a soft landing with minimal deviation from an ideal landing attitude and rotational velocity, with large divert distance capability. The policy was shown to be robust to noise and parameter uncertainty. The computational requirements to run the policy are modest, with only four matrix multiplications required to map the estimated state from the navigation system to a body frame thrust command, which makes it possible to run the policy on the current generation of flight computers. Although fuel efficiency was not optimal, the approach does have the advantages (as compared to generating and tracking an optimal trajectory) of being global over the theater of operations and computational efficiency. Developing a mission ready guidance and control system would require high fidelity models of peripheral systems, and a promising approach would be to integrate the navigation system into the policy (raw sensor output mapping to actuator commands), providing robustness to sensor noise and miscalibration. Future work will explore a reinforcement learning architecture where the policy can adapt in real time to changing dynamics and internal/ external disturbances, and apply the reinforcement learning framework to other aerospace applications, including hypersonic reentry, asteroid close proximity operations and the homing phase missile interception problem.

Acknowledgment

We borrowed several techniques from the PPO2 implementation at open-AI gym baselines. Our Python code adapted functions from "Analytical Mechanics of Space Systems" by Junkins and Schaub (2009).

References

Acikmese, B., Ploen, S.R., 2007. Convex programming approach to powered descent guidance for mars landing. J. Guid. Control Dyn. 30, 1353–1366. https://doi.org/10.2514/1.27553.

Açkmeşe, B., Carson, J.M., Blackmore, L., 2013. Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. IEEE Trans. Control Syst. Technol. 21, 2104–2113. https://doi.org/10.1109/TCST.2012.2237346.

Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A., 2011. Sequential deep learning for human action recognition. In: International Workshop on Human Behavior Understanding, Springer. pp. 29–39.

Battin, R.H., 1999. An Introduction to the Mathematics and Methods of Astrodynamics, revised ed. American Institute of Aeronautics and Astronautics.

Braun, R.D., Manning, R.M., 2007. Mars exploration entry, descent, and landing challenges. J. Spacecraft Rock. 44, 310–323. https://doi.org/10.2514/1.25116.

D'Souza, C., 1997. An optimal guidance law for planetary landing. In: 1997 AIAA Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics. p. 3709. doi:https://doi.org/10.2514/6.1997-3709.

Gaudet, B., Furfaro, R., 2014. Adaptive pinpoint and fuel efficient mars landing using reinforcement learning. IEEE/CAA J. Automatica Sinica 1, 397–411. https://doi.org/10.1109/JAS.2014.7004667.

Gaudet, B., Furfaro, R., Linares, R., 2019a. A guidance law for terminal phase exo-atmospheric interception against a maneuvering target using angle-only measurements optimized using reinforcement meta-learning. arXiv preprint arXiv:1906.02113.

Gaudet, B., Linares, R., Furfaro, R., 2019b. Adaptive guidance and integrated navigation with reinforcement meta-learning. arXiv preprint arXiv: 1904.09865.

Gaudet, B., Linares, R., Furfaro, R., 2019c. Seeker based adaptive guidance via reinforcement meta-learning applied to asteroid close proximity operations. arXiv preprint arXiv: 1907.06098.

Junkins, J.L., Schaub, H., 2009. Analytical mechanics of space systems. Am. Inst. Aeronaut. Astronaut.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv: 1412.6980.

Kullback, S., Leibler, R.A., 1951. On information and sufficiency. Ann. Math. Stat. 22, 79–86.

Levine, S., Koltun, V., 2013. Guided policy search. In: International Conference on Machine Learning, pp. 1–9.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv: 1509.02971.

Lu, P., 2017. Propellant-optimal powered descent guidance. J. Guid. Control Dyn. 41, 813–826. https://doi.org/10.2514/1.G003243.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529 EP –. doi: https://doi.org/10.1038/nature14236.

Ng, A.Y., 2003. Shaping and policy search in reinforcement learning. Ph. D. thesis. University of California, Berkeley.

Ng, A.Y., Russell, S.J., et al., 2000. Algorithms for inverse reinforcement learning. In: Icml. pp. 663–670.

Rao, A.V., Benson, D.A., Darby, C., Patterson, M.A., Francolin, C., Sanders, I., Huntington, G.T., 2010. Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method. ACM Trans. Math. Softw. (TOMS) 37, 22.

Ross, S., Gordon, G., Bagnell, D., 2011. A reduction of imitation learning and structured prediction to no-regret online learning, in. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 627–635.

Sagliano, M., 2018. Generalized hp pseudospectral convex programming for powered descent and landing. In: 2018 AIAA Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics. p. 0617. doi: https://doi.org/10.2514/6.2018-1870.

Sánchez-Sánchez, C., Izzo, D., 2018. Real-time optimal control via deep neural networks: study on landing problems. J. Guid. Control Dyn. 41, 1122–1135.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015a. Trust region policy optimization. In: International Conference on Machine Learning. pp. 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P., 2015b. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv: 1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv: 1707.06347.

Shotwell, R., 2005. Phoenix—the first mars scout mission. Acta Astronaut. 57, 121–134.

Shuster, M.D., 1993. A survey of attitude representations. J. Astronaut. Sci. 41, 439–517.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms. In: Xing, E.P., Jebara, T. (Eds.), Proceedings of the 31st International Conference on Machine Learning, PMLR, Bejing, China. pp. 387–395. http://proceedings.mlr.press/v32/silver14.html.

Sorensen, D.C., 1982. Newton's method with a model trust region modification. SIAM J. Numer. Anal. 19, 409–426.

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction, vol. 1. MIT press, Cambridge.

Szmuk, M., Acikmese, B., 2018. Successive convexification for 6-dof mars rocket powered landing with free-final-time. In: 2018 AIAA Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics. p. 0617. doi: https://doi.org/10.2514/6.2018-0617.

Tedrake, R., 2015. Lecture notes from underactuated robotics. http://underactuated.csail.mit.edu/underactuated.html.

Wang, J., Cui, N., 2018. A pseudospectral-convex optimization algorithm for rocket landing guidance. In: 2018 AIAA Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics. p. 1871. doi: https://doi.org/10.2514/6.2018-1871.

Waslander, S.L., Hoffmann, G.M., Tomlin, C.J., 2005. Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3712–3717. https://doi.org/10.1109/IROS.2005.1545025.

Watkins, C.J., Dayan, P., 1992. Technical note: Q-learning. Mach. Learn. 8, 279–292. https://doi.org/10.1023/A: 1022676722315.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. 8, 229–256.