LEVERAGING OPTIMAL CONTROL DEMONSTRATIONS IN REINFORCEMENT LEARNING FOR POWERED DESCENT

Callum Wilson[(1)] **and Annalisa Riccardi**[(2)]

[(1)] *Department of Mechanical and Aerospace Engineering, University of Strathclyde, Glasgow, United Kingdom, callum.j.wilson@strath.ac.uk*
[(2)] *Department of Mechanical and Aerospace Engineering, University of Strathclyde, Glasgow, United Kingdom, annalisa.riccardi@strath.ac.uk*

ABSTRACT

This work presents an approach to deriving a controller for spacecraft powered descent using reinforcement learning. To assist in the learning process, our approach uses optimal control demonstrations which provide open-loop control for optimal trajectories. Combining these approaches to use the optimal trajectories as demonstrations helps to overcome issues with convergence on desirable policies in the reinforcement learning problem. We demonstrate the applicability of this approach on a simulated 3-DOF Mars lander. The results show that the learned controller is capable of achieving a pinpoint soft landing from a range of initial conditions. Compared to the open-loop optimal trajectories alone, this controller generalises to more initial conditions and can cope with environmental uncertainties.

1 INTRODUCTION

Future space missions to extra terrestrial bodies such as the Moon and Mars will have more stringent requirements on their subsystems than previous missions. One aspect of this is the control systems which must be able to perform effectively in uncertain environments. Considering the landing phase of planetary missions, this requires effective control systems to overcome many uncertainties in course. Intelligent Control methods are one approach to dealing with such difficult problems as they are capable of handling uncertain environments [1]. Such methods combine theories from artificial intelligence, operations research, and automatic control to develop highly autonomous controllers. From the field of artificial intelligence, Reinforcement Learning (RL) has demonstrated results in a variety of difficult optimal control problems. These approaches learn by interacting with an environment (usually in simulation) and discovering which actions lead to the most desirable outcomes. Recent advances in these algorithms and the increasing availability of computing power has seen these methods become more popular recently for control problems.

In the case we examine here of powered descent, RL has been successfully applied to this environment in other works [2]. However, there remain several challenges in applying RL to this problem. There are two main objectives in the powered descent problem – first to reach the landing point and second to minimise the fuel consumed to achieve this. The former objective gives rise to what is referred to as 'sparse rewards' where most states in the environment give no feedback to the agent except the final state, which indicates its success or failure. These sparse reward settings are difficult for RL agents

[3], since in the initial periods of learning the agent must choose actions randomly to try and find the desired final state. In large state spaces, finding the desired state is unlikely and so agents will often converge on unacceptable policies.

Various methods exist to overcome this problem of exploration in RL. Other approaches to the powered descent problem use a shaped reward function which gives the agent more information on the effectiveness of its policy during training [4]. These are useful for guiding the agent towards the desired final goal, however for this problem they can also result in a loss of optimality with respect to the other requirement of minimising fuel consumption. In addition, improper specification of the reward function can lead to behaviour known as 'reward-hacking' where the agent exploits the reward function to converge on an undesirable policy [5]. Another class of methods uses demonstrations that reach the final goal to be fed to the agent during its training. This concept of using demonstrations to prime a RL agent is well studied and most often applied to robotics problems [3, 6, 7].

To use demonstrations to help train an agent, we must first consider how to find demonstrations which can be used by the agent. Certain real-world problems make use of human demonstrations [8, 9], but this is not as suitable for the powered descent problem. Traditional optimal control methods are also commonly applied to the powered descent problem [10]. It can be computationally expensive to solve these optimal control problems under uncertainties and they will often not generalise well outside the conditions for which they are optimised. However, for the purposes of generating demonstrations, it is sufficient to derive optimal trajectories for nominal conditions and allow the RL agent to deal with any uncertainties.

Here we aim to combine the benefits of RL with optimal control demonstrations to solve the fuel-efficient powered descent problem. The approach consists of first generating optimal trajectories from various initial conditions. The continuous actions output by the optimisation are discretised and used to train a RL agent along with its own experience, with the aim of deriving a controller which generalises well to varying initial conditions and environmental disturbances. We compare this approach to an agent trained without demonstrations to demonstrate its merit.

2 METHODS

This section details the methods used to derive the optimal trajectories and how they are used as demonstrations for the reinforcement learning agent. This includes the approach used to discretise the optimal control actions. The methodology is applied to a 3 degree-of-freedom powered descent problem, where the lander has no rotations and thrusters are mounted along its body-frame axes. Fig. 1 shows this proposed methodology schematically.

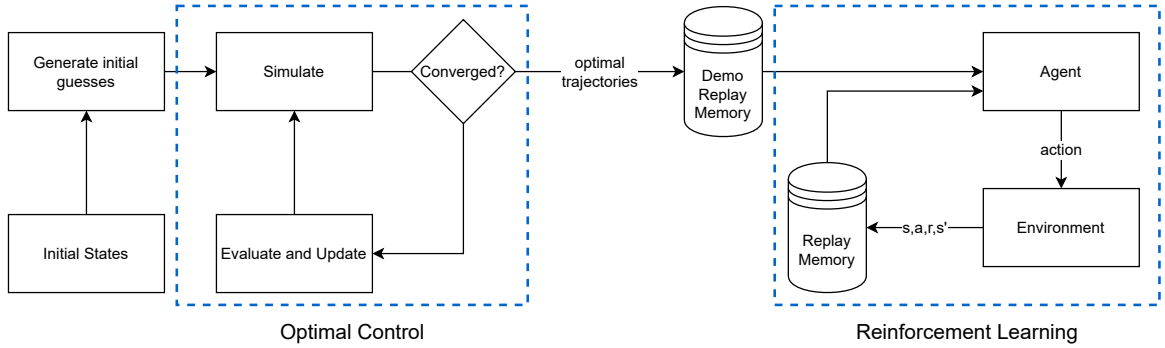


Figure 1: Schematic overview of the proposed methodology consisting of generating trajectories via an optimal control problem and feeding these into a demonstration replay memory for training a reinforcement learning agent.

2.1 Optimal Control Problem

The goal of the optimisation is to minimise the total control effort used to manoeuvre the lander from its initial state to the landing point with a final velocity of 0. The equations of motion of the lander are shown in Eqs. 1-3.

$$\frac{d}{dt}(\mathbf{x}) = \dot{\mathbf{x}} \tag{1}$$

$$\frac{d}{dt}(\dot{\mathbf{x}}) = \frac{\mathbf{T} + \mathbf{F}_{env}}{m} + \mathbf{g} \tag{2}$$

$$\frac{d}{dt}(m) = -\frac{\Sigma_{m=1}^{3} \text{abs}\,(T_m)}{I_{sp} \cdot g_0} \tag{3}$$

where $\mathbf{x} = \{x_1, x_2, x_3\}$ is the lander's position w.r.t. the desired landing point, $\dot{\mathbf{x}} = \{\dot{x}_1, \dot{x}_2, \dot{x}_3\}$ is the lander's velocity, $\mathbf{T} = \{T_1, T_2, T_3\}$ is the body-frame thrust produced by the lander, $\mathbf{F}_{env}$ is the environmental disturbance force, m is the lander's mass, $\mathbf{g} = \{0, 0, -3.72\}$ is the acceleration due to gravity on Mars, $I_{sp} = 210$ is the specific thrust of each thruster, and $g_0 = 9.81$ is the reference acceleration due to gravity. The global frame of reference used has the point $\mathbf{x} = \{0, 0, 0\}$ located at the desired landing location with unit vectors $\vec{i}$, $\vec{j}$, and $\vec{k}$ in the downrange, crossrange, and altitude directions respectively. Note that for generating the optimal trajectories, $\mathbf{F}_{env} = 0$ and the environmental disturbances are only considered when training the RL agent. Then the optimisation problem is defined by Eq. 4.

$$\begin{aligned} \text{minimize} \quad & \sum_{t=0}^{t_f} \left(\sum_{m=1}^{3} T_m^2 \right) \\ \text{subject to} \quad & \mathbf{x}|_{t=t_f} = 0,\ \dot{\mathbf{x}}|_{t=t_f} = 0, \\ & \mathbf{x}|_{t=0} = \mathbf{x}_0,\ \dot{\mathbf{x}}|_{t=0} = \dot{\mathbf{x}}_0,\ m|_{t=0} = m_0, \\ & \mathbf{T}^{min} \leq \mathbf{T} \leq \mathbf{T}^{max}. \end{aligned} \tag{4}$$

The variables to be optimised are the thrusts, $\mathbf{T}$ at each control point and the time-of-flight, t_f. Note that we do not include the additional requirement of the lander's altitude always being greater than zero in the constraints. As shown in the following sections, we still obtain satisfactory trajectories without including this constraint.

2.2 Demonstrations from Optimal Trajectories

Since the RL algorithm implemented here operates over discrete action spaces, we must now convert the continuous thrusts from the optimal trajectories to discrete thrusts. We denote the number of discrete actions in a given direction as n. For example, $n = 3$ denotes a set of possible normalised actions of $[0,\ 0.5,\ 1]$. Let a^* be a normalised continuous action taken by the optimal policy. One simple way to compute the corresponding discrete action is to round the action a^* to the nearest discrete magnitude. Obviously this accumulates errors over the trajectory and generally gives poor solutions.

To remedy this, here we incorporate a heuristic to decide whether to round the optimal action up or down as shown in Eq. 5. This heuristic uses the error between the spacecraft's current velocity and the velocity at the next timestep in the optimal trajectory.

$$a = \frac{\lfloor a^*(n-1) \rfloor}{n-1} + \frac{\text{sgn}(\delta v) + 1}{2(n-1)} \tag{5}$$

$$\delta v = \dot{x}^*_{t+1} - \dot{x}_t \tag{6}$$

where δv is the velocity error and a is the normalised discrete action. At the end of the trajectory where the agent is close to the landing region, $\dot{x}^*_{t+1}$ is held constant at zero in all directions. Although this improves the resulting trajectories compared to simply rounding the actions, the final state may still not be in the desired landing region. To ensure the trajectory with discretised actions reaches a final position closer to 0, we use an approach similar to Hindsight Experience Replay (HER) [11]. This is where the final state is assumed to be the goal and the states over the course of the trajectory are updated accordingly. In practise for this problem, this is achieved by updating the initial position in the trajectory using Eq. 7.

$$\mathbf{x}_0 = \mathbf{x}^*_0 - \mathbf{x}|_{t=t_f} \tag{7}$$

where $\mathbf{x}^*_0$ is the original initial position for the optimal trajectory. Starting from $\mathbf{x}_0$ (with $\dot{\mathbf{x}}_0 = \dot{\mathbf{x}}^*_0$ and $m_0 = m^*_0$) and taking actions as defined by Eq. 5 gives a trajectory which terminates very close to $\mathbf{x} = 0$, as desired.

2.3 Reinforcement Learning Problem

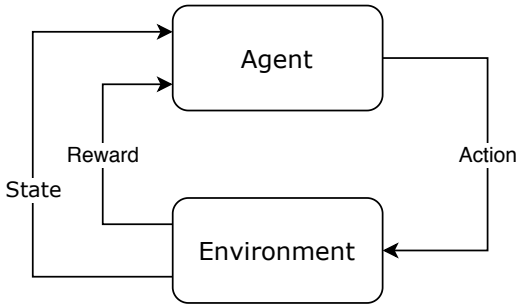


Figure 2: Schematic of agent-environment interaction in reinforcement learning.

A RL problem consists of an agent which interacts with an environment with the goal of maximising its cumulative reward [12]. The agent interacts with the environment via actions and receives a state and reward signal as feedback from the environment, as shown in Fig. 2. A reward indicates the desirability of a certain state-action-state transition. Specifically, the agent aims to maximise its *return*, G_t, as defined in Eq. 8.

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \end{aligned} \tag{8}$$

where r_t denotes the reward at timestep t and γ is the discount factor which controls the extent to which long term rewards are considered.
One subset of approaches to the RL problem uses action-values to determine the best actions to take. For a policy, π which maps states, s to actions, a, Eq. 9 defines the action-value function q_π.

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi \left[G_t | s_t = s, a_t = a \right] \\
&= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right]
\end{aligned} \tag{9}
$$

where $\mathbb{E}_\pi$ denotes the expected value when following policy π. The optimal action-value function is that which has the highest action-value for all state-action pairs as shown in Eq. 10.

$$
q_*(s,a) = \max_\pi q_\pi(s,a) \tag{10}
$$

From this optimal action-value function, the optimal policy which maximises return always performs *greedy* actions by selecting the action with the maximum action-value for the current state. This problem of maximising return can then be formulated as the problem of finding the optimal action-value function.

The Deep Q-Networks (DQN) algorithm [13] is a popular state-of-the-art approach to reinforcement learning based on the Q-learning algorithm [14]. In DQN, action-values are approximated using a Neural Network (NN) with parameters θ which are updated during training. This NN is trained on batches of experience from the agent's *replay memory*, which are sequences of $(s_t, a_t, r_{t+1}, s_{t+1})$ describing state transitions experienced by the agent. Using this experience, Eq. 11 defines the *temporal difference* (TD) error.

$$
e = \begin{cases} Q_\theta(s_t, a_t) - r_{t+1}, & \text{if } s_{t+1} \text{ is terminal} \\ Q_\theta(s_t, a_t) - \left(r_{t+1} + \gamma \max_a Q_{\theta^T}(s_{t+1}, a) \right) & \text{otherwise} \end{cases} \tag{11}
$$

Q_θ is the predicted action-value as parameterised by θ. The loss used to update the NN is then the mean-squared TD-error. To decouple the network updates from action selection, an additional target network is used with the same structure as the action network and parameters denoted θ^T. The target network parameters are initialised equal to θ and then held constant. After n_T timesteps, the parameters θ^T are again set equal to θ and this repeats throughout training. These parameters are used to predict the value $\max_a Q_{\theta^T}(s_{t+1}, a)$ from Eq. 11.

Over the course of training, the agent must explore the state space by not only taking greedy actions, but also taking random actions to potentially find better policies. To do this, DQN uses an ϵ-*greedy* policy, where random actions are selected with probability ϵ at each timestep. Exploration becomes less necessary as the training progresses and the agent seeks to exploit more optimal actions. The value of ϵ is adjusted accordingly by linearly decreasing from ϵ_0 to 0 over n_ϵ episodes, such that towards the end of training the agent takes only greedy actions.

DQN is an *off-policy* RL algorithm which can learn from experiences generated using arbitrary policies. This is why we can use optimal-control demonstrations to train the agent as well as its own epxerience. In the standard DQN algorithm, experiences are stored in the replay memory up to a certain maximum capacity on a first-in-first-out basis. During training, experiences are uniformly random-sampled from the replay memory in batches of size k. Here we add a demonstration replay memory, separate to the agent's own experience replay, which only stores experiences from the optimal-control demonstrations. These are kept and used to update the agent throughout training, again using uniformly random-sampled batches of size k_{demo}.

The reward function can be chosen to guide the agent towards the goal state, which is referred to as *reward shaping*. For the powered descent problem, the simplest reward function would be a *sparse reward* which only rewards the agent upon reaching a terminal state to indicate success or failure in

that episode. However, even when using demonstrations this makes it difficult for the agent to ever reach the goal state. Therefore we use a shaped reward function here which creates a target velocity for the agent to follow. The reward function used here is shown in Eq. 12 which is adapted from the reward used in [2].

$$r = \alpha \left\| \dot{\mathbf{x}} - \dot{\mathbf{x}}_{targ} \right\| + \beta \left\| \frac{\mathbf{T}}{\mathbf{T}^{max}} \right\| + \eta + \kappa \left(r_z < 0.01 \; and \; \|\mathbf{x}\| < x_{lim} \; and \; \|\dot{\mathbf{x}}\| < \dot{x}_{lim} \right) \tag{12}$$

where the following quantities are defined:

$$\dot{\mathbf{x}}_{targ} = -v_0 \left(\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|} \right) \left(1 - exp\left(-\frac{t_{go}}{\tau} \right) \right) \tag{13}$$

$$v_0 = 70 \tag{14}$$

$$t_{go} = \frac{\|\hat{\mathbf{r}}\|}{\|\hat{\mathbf{v}}\|} \tag{15}$$

$$\hat{\mathbf{r}} = \begin{cases} \mathbf{x} - [0 \; 0 \; 15], & \text{if } x_3 > 15 \\ [0 \; 0 \; x_3], & \text{otherwise} \end{cases} \tag{16}$$

$$\hat{\mathbf{v}} = \begin{cases} \dot{\mathbf{x}} - [0 \; 0 \; -2], & \text{if } x_3 > 15 \\ \dot{\mathbf{x}} - [0 \; 0 \; -1], & \text{otherwise} \end{cases} \tag{17}$$

$$\tau = \begin{cases} 20, & \text{if } x_3 > 15 \\ 100, & \text{otherwise} \end{cases} \tag{18}$$

The time-to-go, denoted t_{go}, can be thought of as the remaining time-of-flight starting from position $\hat{\mathbf{r}}$ and travelling at velocity $\hat{\mathbf{v}}$. These quantities $\hat{\mathbf{r}}$ and $\hat{\mathbf{v}}$ depend on the lander's altitude such that the final $15m$ of descent only have movement primarily in the k-direction. The constants α, β, η, and κ weight different terms of the reward function. First, the α term gives a penalty for the difference between the lander's velocity and the target velocity. The target velocity is defined in Eq. 13. Next, the β term gives a penalty for the control effort at each step, since one of the problem goals is to minimise total fuel consumption which is proportional to control effort. In this term, the division indicates elementwise division across each thrust magnitude and its corresponding maximum value. η is a positive constant which can be seen as a means to encourage the agent to continue advancing in the environment. Finally, the κ term rewards a successful landing where the lander's altitude is less than $0.01m$ and its position and velocity are within specified limits of $x_{lim} = 5m$ and $\dot{x}_{lim} = 2m/s$.

3 EXPERIMENTS

This section details the experiments carried out to demonstrate the methods detailed previously. First, we compute optimal trajectories for a range of initial conditions. Then we determine the best discretisation limits to use and convert the optimal trajectories to initialise the demonstration replay buffer. These are used when training the DQN agent, with noise added in the environment. In all cases when simulating the environment, the equations of motion (Eqs. 1-3) are integrated with a constant step size of $dt = 0.2s$.

3.1 Computing Optimal Trajectories

Table 1 shows the maximum and minimum initial positions and velocities of the spacecraft over which optimal trajectories are generated. Its initial mass is fixed as $2000kg$ and the initial altitude is also

fixed as $2.5km$. Optimal trajectories are computed for all combinations of the min. and max. values of each variable, plus the midpoint of all variables. This gives a total of 33 initial conditions from which optimal trajectories were generated.

Table 1: Range of initial conditions for generating optimal trajectories.

State Variable	Min.	Max.
i-position	$0.5km$	$1km$
j-position	$-1km$	$1km$
k-position	$2.5km$	$2.5km$
i-velocity	$-70ms^{-1}$	$-10ms^{-1}$
j-velocity	$-30ms^{-1}$	$30ms^{-1}$
k-velocity	$-90ms^{-1}$	$-70ms^{-1}$
mass	$2000kg$	$2000kg$

The convergence of the optimisation procedure and the quality of the final solution are both heavily dependent on the initial guess passed to the algorithm. Here we use a simple initial guess which does not meet the required constraints, but still manages to manoeuvre the lander close to the desired landing point. Thrusts in each direction are fixed at a constant value with the i- and j-direction thrusts (i.e. T_1 and T_2) being determined using Eq. 19.

$$T = \frac{-2m_0(x_0 + \dot{x}_0 \cdot t_f)}{{t_f}^2} \tag{19}$$

This gives the required thrust starting from x_0 and $\dot{x}_0$ such that the final position $x|_{t=t_f} = 0$, assuming the mass remains fixed at m_0. This assumption does not substantially affect the true final position and is sufficient for the initial guess. Then to determine the time-of-flight and thrust magnitude in the k-direction, we perform a grid search over a range of these parameters to find those which minimises the norm of the final position and velocity. Here we also include the constraint in the initial guess that the altitude (k-position) must never be less than zero.

For all of the optimisations, the number of control points is fixed at 80. This in each direction plus the time-of-flight gives a total size of the vector to be optimised of 241. Given the variable time-of-flight and fixed number of control points, these control points will usually fall outside of the integration points when simulating the environment. Actions on timesteps between control points are linearly interpolated to give a continuous control profile. The minimum and maximum thrusts in each direction are set as $\mathbf{T}^{min} = \{-11,\ -11,\ 0\}kN$ and $\mathbf{T}^{max} = \{11,\ 11,\ 12\}kN$ respectively. Based on the outputs of the optimisation, these limits are adjusted for the RL problem as discussed in Sec. 4.

The optimisation problem, as stated in equation 4, can be solved using various constrained optimisation methods. Here we use the Sequential Least Squares Programming (SLSQP) algorithm [15] as implemented in the SciPy Python library. Then to select the number of discretisation limits, we perform a grid search with the number of actions in the i- and j-directions being equal, i.e. $n_i = n_j$, with the goal of maximising the number of successful episodes.

3.2 DQN with Demonstrations

Using the demonstrations we now train a DQN agent on the environment, including environmental disturbances. The force $\mathbf{F}_{env}$ from Eq. 2 is modelled as gaussian noise (with mean $0N$ and standard deviation $100N$); sampled for every second in an episode and linearly interpolated for timesteps

between samples. The values for the coefficients in the reward function (Eq. 12) are $\alpha = -0.01$, $\beta = -0.05$, $\eta = 0.01$, and $\kappa = 10$.
The state representation used as input to the network contains the position, velocity, and mass of the lander. These are multiplied by a scaling factor $\mathbf{s}_{scale}$ as shown in Eq. 20 to avoid excessive values input to the network. The values for the scaling factor are shown in Eq. 21.

$$s = \{\mathbf{x},\ \dot{\mathbf{x}},\ m\} \cdot \mathbf{s}_{scale} \tag{20}$$

$$\mathbf{s}_{scale} = \{0.01,\ 0.01,\ 0.01,\ 0.1,\ 0.1,\ 0.1,\ 0.005\} \tag{21}$$

The hyperaparameters used in our simulations are shown in Table 2. Both the NNs have 3 hidden layers with numbers of hidden units as shown. The activation function of the NN is $tanh$ and weight updates are performed using RMSProp optimisation with learning rate as shown.

Table 2: Hyperparameters for training the DQN agent.

Parameter	Value
Hidden units	(300,200,300)
learning rate	1×10^{-5}
ϵ_0	0.6
n_ϵ	4000
γ	0.926
k	100
k_{demo}	100
n_T	65

We train two agents - one which uses the demonstrations and one without - to compare their performance. In the case of the agent without demonstrations, the minibatch size is set to $k = 200$. Both agents are trained for 10000 episodes. To test the agents, following training we simulated 1000 episodes using each agent and assessed their performance. For training and testing the agents, we extend the range of initial conditions used for the optimal trajectories to ensure they generalise to points outwith the demonstrations. The new ranges of initial conditions are shown in Table 3.

Table 3: Range of initial conditions for training and testing the DQN agent.

State Variable	Min.	Max.
i-position	$0.4km$	$1.1km$
j-position	$-1.1km$	$1.1km$
k-position	$2.4km$	$2.6km$
i-velocity	$-75ms^{-1}$	$-5ms^{-1}$
j-velocity	$-35ms^{-1}$	$35ms^{-1}$
k-velocity	$-95ms^{-1}$	$-65ms^{-1}$
mass	$2000kg$	$2000kg$

4 RESULTS

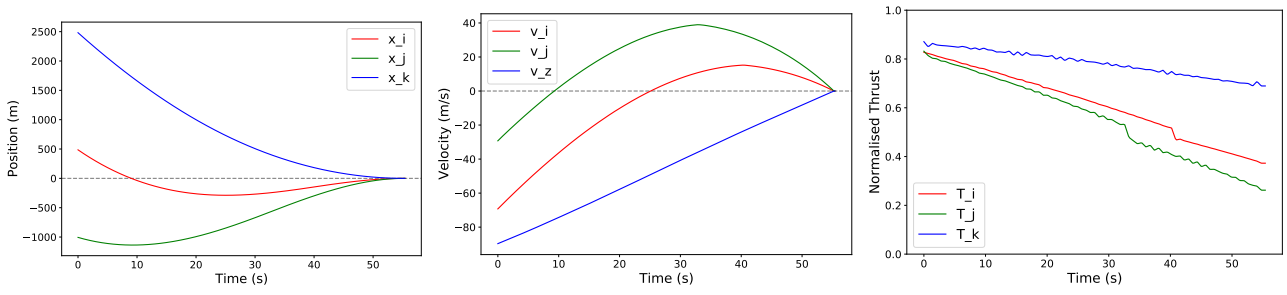
4.1 Optimal trajectories

Table 4 shows the performance of the optimal trajectories in terms of the final state in the trajectory. In all cases they achieve a pinpoint landing with the maximum final position being $0.259m$ amd maximum final velocity of $0.011m/s$. As would be expected for the varying initial conditions, fuel consumption also varies in the range of $291kg$ to $550.5kg$. Considering the thrust magnitudes, these remain well within the limit of $\pm 11kN$ for T_1 and T_2, however for one of the trajectories the thrust T_3 saturates at $12kN$. Based on these observations, the thrust limits were adjusted with the aim of better encompassing the necessary range for these trajectories. The new thrust limits were selected as $\mathbf{T}^{min} = \{-10,\ -10,\ 0\}kN$ and $\mathbf{T}^{max} = \{10,\ 10,\ 13\}kN$. These are the limits used in all further results for action discretisation and training the RL agent.

Table 4: Summary statistics of the optimal trajectories' performance. Positions, velocities, and fuel consumptions are given for the final state of all trajectories. Thrusts are given for the whole trajectories.

Quantity	Median	Min.	Max.
Norm ij-position (m)	0.015	5.6×10^{-4}	0.259
k-position (m)	0.016	-0.043	0.240
Norm ij-velocity (m/s)	3.5×10^{-4}	1.8×10^{-5}	0.011
Velocity-k (m/s)	1.7×10^{-4}	-9.3×10^{-3}	7.8×10^{-3}
Fuel (kg)	415.9	291.0	550.5
T_1 (N)	621	-3331	9320
T_2 (N)	-36	-7415	7557
T_3 (N)	8592	1292	12 000

Fig. 3 shows an example optimised trajectory where the initial state is $\mathbf{x}_0 = \{0.5,\ -1,\ 2.5\}\,km$ and $\dot{\mathbf{x}}_0 = \{-70,\ -30,\ -90\}\,m/s$. The thrust profiles appear mostly smooth and linearly decreasing with only slight ripples. For T_1 and T_2, the thrust jumps from positive to negative indicating a change from accelerating toward the target to reducing the speed to zero for landing, as indicated by the velocity profiles. In the k-direction, the velocity magnitude decreases almost linearly over the course of the trajectory.



(a) Position (b) Velocity (c) Thrust

Figure 3: Example optimised trajectory. Grey lines indicate desired final state of position and velocity equal to zero.

4.2 Discretised Optimal Trajectories

When selecting the number of discrete actions in each direction, the actions are always linearly spaced over the entire range. As a result, the i- and j-directions always use an odd number of actions to include the point where the action $a = 0.5$ such that the thrust is zero. This does not apply in the k-direction where $a = 0$ corresponds to zero thrust, and so even numbers of actions can also be used.
Results of the grid search over number of discrete actions are shown in Table 5. For the purposes of demonstration data, it is desirable to have as many successful episodes, where the agent receives a positive reward, as possible. From the results, it is interesting to note that even values of n_k perform far worse in general than odd values - especially where $n_k = 8$ which had no successful episodes for any value of $n_{i/j}$. It is clear that $n_k = 3$ is the best choice for the number of actions in the k-direction. Selecting the number of actions in the i- and j-directions requires consideration of the problem complexity as well as performance, since a greater number of actions gives a larger action-space size. Both $n_{i/j} = 7$ and $n_{i/j} = 9$ have the best percentage of successful episodes and these result in action space sizes of 147 and 243 respectively. Since $n_{i/j} = 7$ gives a significantly smaller state space size, this is selected as the number of actions for the i- and j-directions.

Table 5: Percentage of successful episodes following action discretisation. Shading indicates quartile with darker shades having more successful episodes.

	n_k	2	3	4	5	6	7	8	9	10
$n_{i/j}$										
3		48.5	78.8	30.3	63.6	24.2	57.6	0.0	48.5	3.0
5		42.4	75.8	24.2	66.7	21.2	54.5	0.0	42.4	3.0
7		39.4	81.8	33.3	66.7	21.2	54.5	0.0	45.5	3.0
9		51.5	81.8	33.3	66.7	24.2	54.5	0.0	42.4	3.0
11		30.3	78.8	30.3	60.6	21.2	51.5	0.0	42.4	3.0

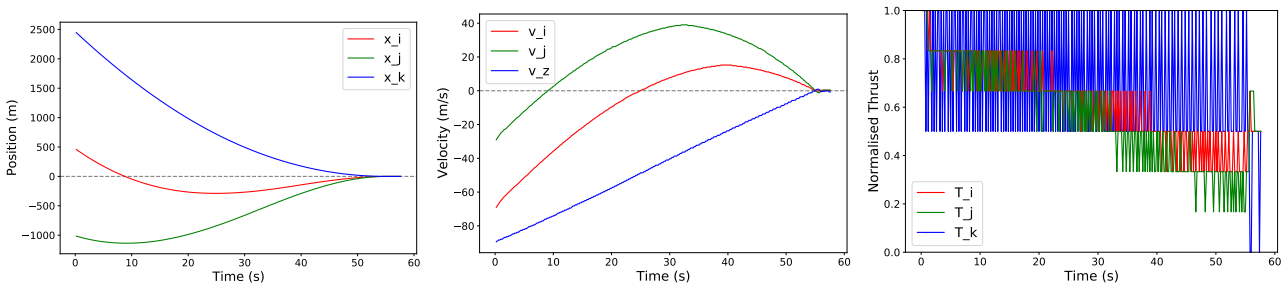
Table 6 shows the performance of the discretised trajectories. Unlike for the optimal trajectories, in this case the episodes terminate when the altitude, x_3 is less than 0.01. Most discretised trajectories perform well, again achieving the pinpoint landing as in the optimal trajectories. However, in some cases the episode terminates early and far from the desired landing state, with the maximum distance being $178.3m$ and maximum downward velocity of $23.7m/s$. Having these episodes included in the replay memory is not an issue provided there are also successful episodes. Finally, we see that on average the fuel consumption is lower for the discretised trajectories than the optimal ones. This can be partly attributed to the unsuccessful episodes which terminate early. In addition, the 'bang-bang' nature of the discrete control inputs is more difficult to converge on using constrained optimisation, but can be more optimal with respect to fuel minimisation.
Fig. 4 shows an example discretised trajectory. This originally started at the same initial state as in Fig. 3, with the initial position adjusted to $\mathbf{x}_0 = \{0.472,\ -1011,\ 2.462\}\ km$ so it reaches the landing area. The effect of action discretisation is clear when looking at the thrust profile, particularly in the k-direction which only has three thrust magnitudes.

4.3 RL-agent training

For the purposes of comparing the agents' performance, here we refer to the agent trained with demonstrations as Agent 1 and the agent trained without demonstrations as Agent 2. Over the course of training, Agent 1 learns to find the desired final state and reaches this state more frequently as training

Table 6: Statistics of the discretised optimal trajectories using $n_i = n_j = 7$ and $n_k = 3$ at their final state.

Quantity	Median	Min.	Max.
Norm ij-position (m)	1.7×10^{-5}	2×10^{-7}	178.3
k-position (m)	-8.4×10^{-5}	-1.3	2.6×10^{-4}
Norm ij-velocity (m/s)	0.384	2.8×10^{-3}	22.2
k-velocity (m/s)	-0.471	-23.7	-0.027
Fuel (kg)	407.9	276.1	503.2



(a) Position (b) Velocity (c) Thrust

Figure 4: Example optimised trajectory following action discretisation, with $n_i = n_j = 7$ and $n_k = 3$. Grey lines indicate desired final state of position and velocity equal to zero.

progresses. This is shown in Fig. 5 where the cumulative successful episodes increase very gradually up until approximately 4000 episodes, after which point it is no longer taking random actions and so it achieves more successes. On the other hand, Agent 2 never achieves a successful episode over the whole of its training. Instead it exhibits 'reward-hacking' behaviour as will be shown in the evaluation episodes.

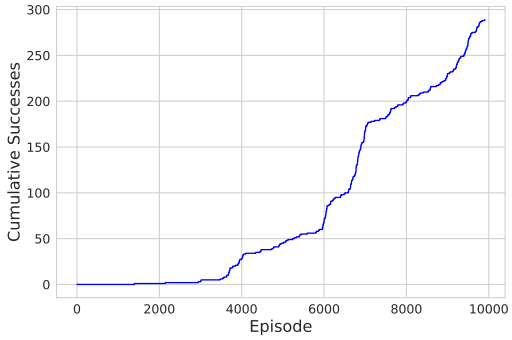
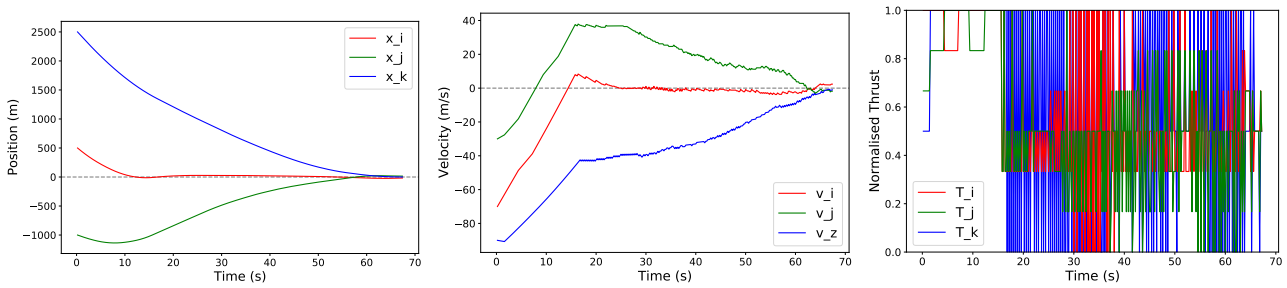


Figure 5: Cumulative successful episodes over training using optimal control demonstrations.

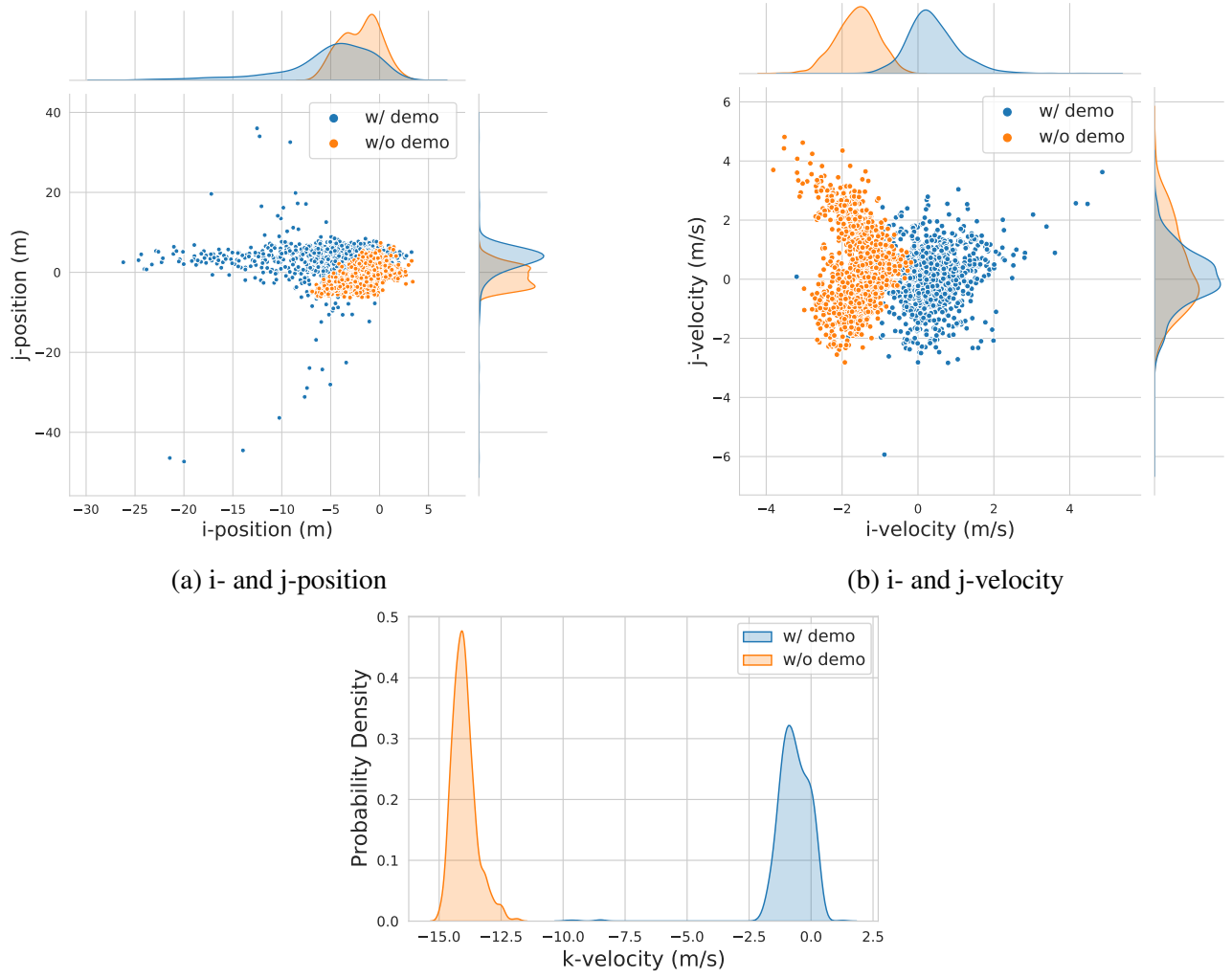
Fig. 6 shows an example trajectory from Agent 1. As a result of the reward function's target velocity, this controller shows a more aggressive output, which quickly accelerates to the desired velocities over the first 20s of the trajectory before gradually decelerating over the remainder of the episode.



(a) Position (b) Velocity (c) Thrust

Figure 6: Example trajectory from the trained agent. Grey lines indicate desired final state of position and velocity equal to zero.

4.4 RL-agent testing



(a) i- and j-position

(b) i- and j-velocity

(c) k-velocity

Figure 7: Distribution of final states for agents trained with and without demonstrations.

Fig. 7 shows the distributions of final states over the evaluation episodes for both agents and Table 7 gives summary statistics of their performance. These results are from simulating 1000 episodes from

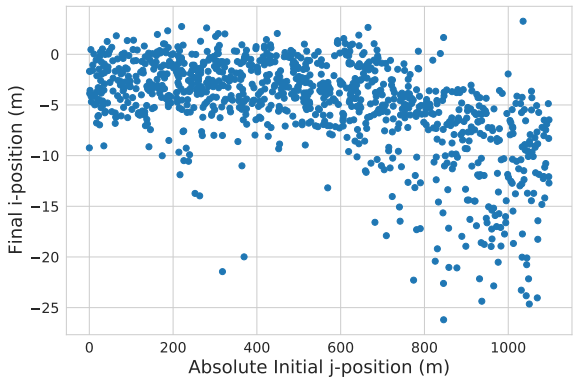


Figure 8: Scatter plot of initial j-position against final i-position for agent trained with demonstrations.

the range of initial conditions and environmental disturbances used in training. The distributions of k-velocities show clearly the undesirable behaviour of Agent 2, since all its velocities in this direction range between $-11.8m/s$ and $-15m/s$. This behaviour can be explained by the discontinuity in the reward function for the final $15m$ of descent. In this region, the target velocity is zero in the i- and j-directions and decreases rapidly in the z-direction. Instead of decelerating, Agent 2 falls through this region as quickly as possible to minimise the negative reward it receives. Including the demonstrations biases the agent towards receiving the positive final reward, which results in better performance with respect to the soft-landing requirement. While Agent 1 performs much better on the whole with respect to its k-velocity, there are also evaluation episodes where it shows poor performance with velocities up to $-9.78m/s$. Across the 1000 evaluation episodes for Agent 1, 5 episodes have final k-velocities less than $-2m/s$ and 60 episodes have final k-velocities less than $-1.5m/s$. These are still superior to the results from Agent 2, however the success rate must be improved before this control system could be deployed in practice.

Considering the distributions of i- and j-velocities in the evaluation episodes, the performance of both agents is far more similar than for the k-velocities. The velocities for Agent 1 are more centred around 0, whereas for Agent 2 they are predominantly negative in the i-direction. This is also indicated by the median values for norm ij-velocities of 0.427 for Agent 1 and 1.58 for Agent 2. In both cases there are large velocities outwith the desired magnitudes, however for Agent 1 the majority of velocities lie close to zero. For the final positions, Agent 2 has more tightly clustered positions close to zero compared to Agent 1. Most of the final positions of Agent 1 fall towards the negative i-direction with several outliers in the j-direction as well. Fig. 8 shows that there tends to be higher variance in the final i-position for higher absolute values of the initial j-position. This could suggest the agent performs more poorly for larger values in the initial j-position - even in the other dimensions.

Finally comparing the fuel consumption of both agents, Agent 2 has lower fuel consumption as can be expected since it does not decelerate towards the end of its trajectory. Therefore, this makes for a poor comparison to Agent 1. Looking instead at the fuel consumption from the discretised optimal trajectories, shown in Table 6, the median fuel consumption of Agent 1 is $88kg$ more than that of the discretised optimal trajectories. More significantly, the maximum fuel consumption of Agent 1 is $186.9kg$ more. The main reasons for this are the larger range of initial conditions for Agent 1 which can cause more fuel consumption and the environmental uncertainties present for Agent 1, which require a more robust and less optimal controller to handle them. Altering the reward function for the RL problem to give greater weight to control effort could improve the agent's performance in this respect.

Table 7: Summary statistics of the trained agents' performance over 1000 evaluation episodes. Agent 1 is trained with demonstrations and Agent 2 is trained without demonstrations.

	Agent 1			Agent 2		
Quantity	Median	Min	Max.	Median	Min	Max.
Norm ij-position (m)	6.04	0.255	51.4	3.37	0.151	8.81
Norm ij-velocity (m/s)	0.427	2.4×10^{-4}	4.86	1.58	0.185	3.81
k-velocity (m/s)	-0.691	-9.78	1.28	-14.1	-15.0	-11.8
Fuel (kg)	495.9	419.4	690.1	335.9	277.6	413.7

5 CONCLUSIONS

Here we have presented a methodology for applying reinforcement learning to the problem of spacecraft powered descent. This uses optimal control demonstrations to overcome issues seen when training an agent to solve this problem. It is clear that using demonstrations helps the agent achieve the desired soft landing with a useful degree of accuracy. Future work should look at improving the performance of this approach, such as by tuning the agent's hyperparameters or adjusting the reward function to better suit the problem. Furthermore, other off-policy RL algorithms which can operate over continuous action spaces could be applied to the problem using this approach with optimal demonstrations.

REFERENCES

[1] C. Wilson, F. Marchetti, M. Di Carlo, A. Riccardi, and E. Minisci, "Classifying intelligence in machines: a taxonomy of intelligent control," *Robotics*, vol. 9, no. 3, p. 64, 2020.

[2] B. Gaudet, R. Linares, and R. Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing," *Advances in Space Research*, vol. 65, no. 7, pp. 1723–1741, 2020.

[3] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6292–6299.

[4] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[5] D. Hadfield-Menell, S. Milli, P. Abbeel, S. Russell, and A. D. Dragan, "Inverse reward design," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6768–6777.

[6] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[7] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *arXiv preprint arXiv:1709.10087*, 2017.

[8] S. Schaal *et al.*, "Learning from demonstration," *Advances in neural information processing systems*, pp. 1040–1046, 1997.

[9] M. E. Taylor, H. B. Suay, and S. Chernova, "Integrating reinforcement learning with human demonstrations of varying ability," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. Citeseer, 2011, pp. 617–624.

[10] P. Lu, "Propellant-optimal powered descent guidance," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 4, pp. 813–826, 2018.

[11] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *arXiv preprint arXiv:1707.01495*, 2017.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[14] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[15] D. Kraft *et al.*, "A software package for sequential quadratic programming," 1988.