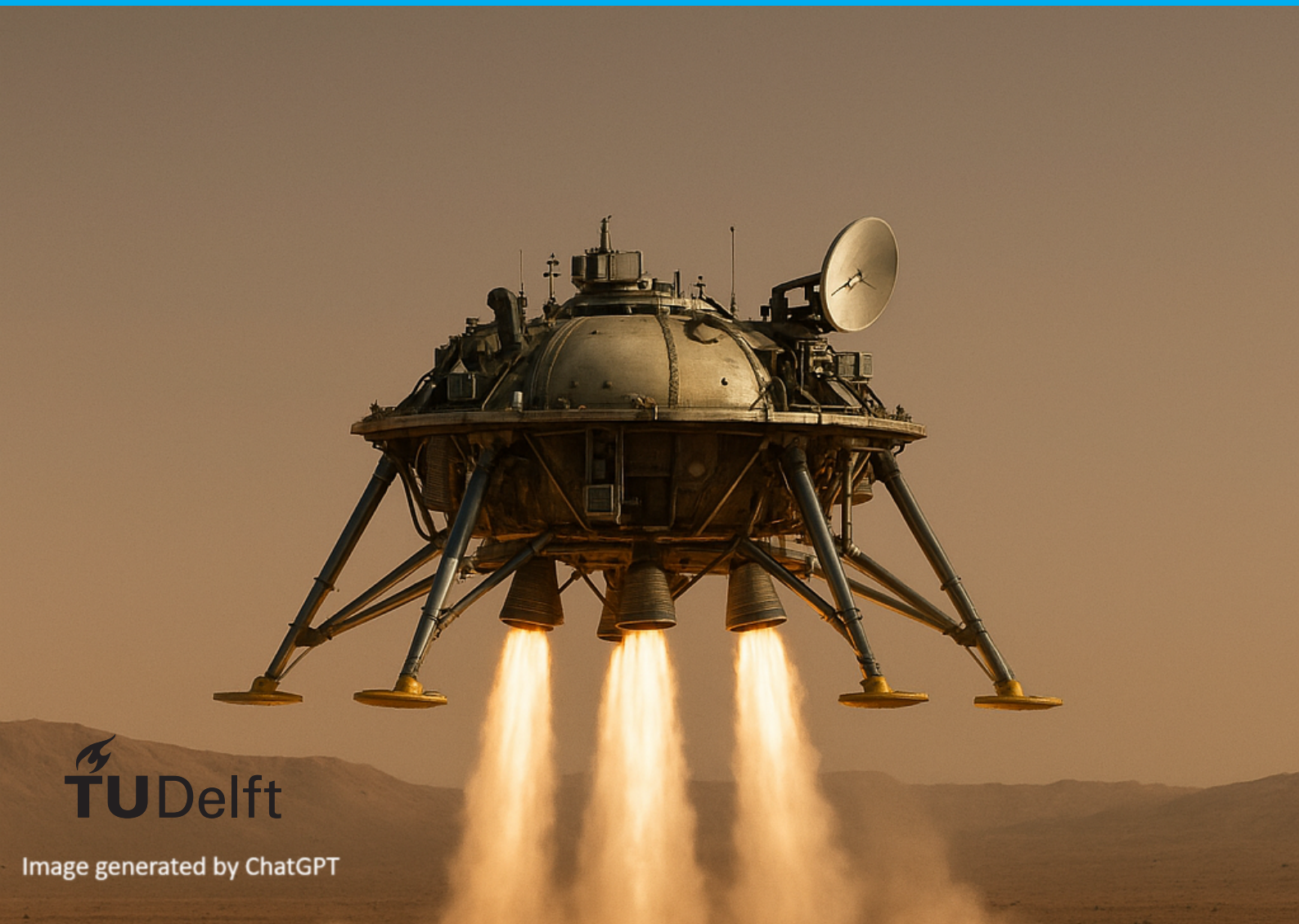
Reinforcement Learning for a Six Degree of Freedom Martian Landing

Ayman el Ghalbzouri

Date: August 24, 2025



Reinforcement Learning for a Six Degree of Freedom Martian Landing

Thesis report

by

Ayman el Ghalbzouri

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on August 28th, 2025 at 14:30

Thesis committee:

Chair:	Dr.ir,E.Smeur
Supervisor:	Dr.ir.E.J.van Kampen
External examiner:	Dr.ir.M.Naeije
Additional member:	ir.I.El-Hajj
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	May, 2021 - July, 2025
Student number:	4381459

Faculty of Aerospace Engineering · Delft University of Technology

Contents

1 Introduction

Landing on a planet is one of the last phases of many space missions. The landing should not be too hard to ensure that the equipment is not damaged. The exact landing position also matters in some cases, especially if the landing has to be made near a hazardous area. Minimizing the position error is even more important for rendezvous/docking problems. This may be done by conventional control methods if the environment is well known and if there are no or small environmental disturbances, but this is not always the case.

Reinforcement learning methods could in some cases learn to cope with these uncertainties. In the last couple of years, reinforcement learning in planetary landing and rendezvous/docking problems became an interesting framework for aerospace engineers to research. Reinforcement learning is one of the paradigms within machine learning, the other two are supervised and unsupervised learning.

In supervised learning, the goal is to learn from a labeled input set which will be used to train the algorithm. This examples inputs are mapped to an output. The learning algorithm is 'told' what the output should be. The goal is to learn patterns that could be used to make predictions. The learned mapping function can be used to label unlabeled data. An example of an supervised learning application is recognizing animals by feeding the learning algorithm with pictures of animals with a label which tells the algorithm which animal it is. Eventually, the algorithm should be able to label images of animal by itself.

Unsupervised learning is about learning from an unlabeled training set. The goal is to find hidden structures and patterns in a training set. Unsupervised learning could be used to organize complex data to make it meaningful. An example of an application is anomaly detection, for example detection of fraud.

Reinforcement learning is about optimizing by collecting rewards in an environment. The reward collected during an episode should be maximized. The reinforcement learning algorithm should learn to map the observable states to an action in such a way that the expected discounted reward is maximized. The expected discounted reward will be defined in the next chapter. The mapping from observable states to an action is called a policy.

Reinforcement learning is applied in this research because it can learn directly from it environment and find an solution for a problem by it own. Different from supervised learning and unsupervised learning, reinforcement learning algorithms have the goal to map observations to actions in an optimal way.

1.1. Motivation

Multiple conventional control methods exist. Examples are full-state feedback, PID controllers and LQR controllers. Reinforcement learning is a promising way to solve control engineering problems. Table 1.1 is from a paper written by Gaudet, Linares and Furfaro [1]. Reinforcement learning is not always a better choice, but from this table it could be concluded that reinforcement learning may be a good choice for some control problems.

Table 1.1: A comparison between optimal control and reinforcement learning [1]

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 (<1ms in this work)
Hybrid dynamics requires special treatment	Hybrid Dynamics handled seamlessly
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	Stochastic, learning does not converge every time, may need to run multiple policy optimizations

A good reason to use reinforcement learning is that the dynamics don't have to be represented as an ordinary differential equation. Infact, there are no constraints at all in the dynamics representation. Because of this, there are less constraints in the model. This means that reinforcement learning agents can learn form higher fidelity models or no model at all for a model free method of reinforcement learning is .

Many conventional control methods require the dynamics to be represented in the following form [2]:

$$\begin{aligned} \dot{\mathbf{x}}(\mathbf{t}) &= \mathbf{A}\mathbf{x}(\mathbf{t}) + \mathbf{B}\mathbf{u}(\mathbf{t}) \\ \mathbf{y}(\mathbf{t}) &= \mathbf{C}\mathbf{x}(\mathbf{t}) + \mathbf{D}\mathbf{u}(\mathbf{t}) \end{aligned} \tag{1.1}$$

In this equations, $\mathbf{x(t)}$ is known as the state vector, $\mathbf{u(t)}$ is the input vector and $\mathbf{y(t)}$ is the output vector. $\mathbf{A}$ is known as the state matrix and $\mathbf{B}$ as the input matrix. $\mathbf{C}$ is the output matrix and $\mathbf{D}$ is called the feed-through matrix. Only linear problems may be written in this form. Non-linear equations have to be linearized. If equation 1.1 is used to describe a nonlinear problem, equation 1.1 will only be an approximation of the problem. This linearized equations are only valid near the trim points. Reinforcement learning agents on the other hand learn a policy without caring if the system dynamics are linear or nonlinear [3].

The output of a system can't be used directly to control the system in most cases. The states have to be estimated from the sensor output because of the fact that sensors are not perfect. For linear systems, such state estimation can be achieved through the application of optimal filtering techniques, such as the Kalman filter. This could also be done for nonlinear problems by linearizing the dynamics first. Reinforcement learning agents however are able to learn directly to compensate for sensor distortion.It may also be a good option to use reinforcement learning algorithms if the environment is not stationary. The more non-stationary the environment is, the higher the learning rate should be. The learning rate is a parameter that will be introduced in section 3.1.5. Optimal control also assumes full knowledge about the environment. Reinforcement

learning does not need full information because it relies on measured data. Even if a virtual agent is trained in a non-perfect computer model, it is able to adapt to the real environment in the real world due to the adaptive nature of reinforcement learning. Some problems present significant challenges in terms of analytical modeling or classical engineering approaches, particularly when system dynamics are complex, high-dimensional, or poorly understood. In such cases, reinforcement learning offers a promising alternative by enabling agents to learn a policy directly from interaction with the environment, without requiring a model of the system.

An example of a hard to engineer problem is an autonomous helicopter flight. Autonomous helicopter flight problems are high dimensional, nonlinear and motions are also coupled. It is possible to train an autonomous helicopter to perform an inverted flight by training it via reinforcement learning [8]. This was done by applying supervised learning to identify the system. The collected flight data was used to make a stochastic and non-linear model. This model was used to train the autonomous helicopter to perform an inverted flight but also more simple tasks like hovering and way point following. An important side benefit mentioned by the authors was that it becomes easier to reconfigure the helicopter or change the payload. The authors did this for example by changing the rotors and mounting or removing sensors. Actions like this also changes the dynamics. The authors were able to train a new controller by using the already existing software without much effort. It could also be considered a problem that the reinforcement learning agent has to be trained again if the dynamics change.

1.2. Challenges in space guidance, navigation and control

Artificial intelligence is developing fast in the last decades. Aerospace engineers are also becoming interested in this area [18]. They have done work in for example planetary landings, spacecraft guidance and hovering and spacecraft attitude control. In this report, applications of reinforcement learning in space guidance, navigation and control will be described with a focus on planetary landings and rendezvous/docking.

Reinforcement learning is a good option if the environment is changing or uncertain. This is because of the adaptiveness of reinforcement learning algorithms. The agent will learn to cope with the unknown dynamics. Gaudet and Furfaro showed that a spacecraft is able to learn non-uniform rotation and a non-uniform gravity field to hover above an asteroid [19]. They have also demonstrated that their reinforcement controller is robust. The controller is made robust to un-modeled dynamics by modeling noise in the environment. In general it could be stated that reinforcement learning should be considered in autonomous navigation and control problems if there are uncertainties and robustness is of importance [19].

If the dynamics are known partially, an agent may be pretrained in a simulation to learn a sub-optimal policy offline. A PID controller, which is not an optimal controller, may also be used as an initial policy. The optimal policy could then be learned online in the real world. This could only be done if it is ensured that the initial policy is good enough to keep the spacecraft operational. It is undesirable to lose a spacecraft due to an accident because of a bad initial policy.

Some problems in space GNC are hard to engineer because of coupling of motion, which was also mentioned earlier in the context of a autonomous helicopter flight. An example is controlling a planetary landing problem with just four actuators [1]. A reinforcement learning algorithm can learn to cope with this coupling of motion without an analytic and explicit mathematical formulation of the problem.

1.3. Challenges in reinforcement learning

There are some challenges in reinforcement learning. Kober, Bagnell and Peters do mention some problems of reinforcement learning in their paper about reinforcement learning in robotics [11].
First, they describe the curse of dimensionality. If the number of dimensions grows, the state space grows exponentially. If we assume that every state is discretized into 10 levels and that there are n states, the state space will have a size of 10^n. For high-dimensional problems, the computation time will be very high for this reason and much data will have to be stored. Determining the value function for every state may become infeasible.

They also describe a problem called the curse of real-world samples. Exploration should be safe if reinforcement

learning is applied online. Switching between controllers may be a way to do this. The dynamics also may change as a result of external factors, for example temperature effects or wear. External factors may also have an influence on the sensors. Light may for example influence the vision system. Reinforcement learning is applied on computers. This means that time should be discretized. There will always be a delay between sensing and actuation. The sensed filtered state may lag behind the real state. In most reinforcement learning algorithms however, the actions are assumed to have an effect instantaneously. Delays would violate the Markov assumption.

The third curse mentioned in the paper is the curse of under-modeling and model uncertainty. Reinforcement learning may learn off-line in a computer model. The learned policy can be transferred to the real world. Creating a good model is important if this approach is chosen but may be challenging. Due to accumulating model errors, a real robot may diverge from the robot in a simulation. Transferring policies from a simulation to the real world works best for self-stabilizing systems. In unstable systems, small model variations may have very big consequences.

The last problem described is the curse of goal specification. The goal is specified by a reward function. Specifying a reward function may be easier than specifying the desired behavior. However, choosing a good reward function is not always simple. Reward may be binary, in that case a reward will only be given if a game is won for example. However, a reward will be given rarely in that case. Intermediate rewards should be given. This reward are called shaping rewards. A trade-off may also be made while rewarding the agent. Hitting a ball very hard for example may lead to a high score but could be bad for the life-span of the hardware. The reward may also be exploited in a way not meant by the designer.

1.4. Research questions

The goal of the thesis is to design a working reinforcement learning controller for a Martian lander. This Martian lander is controlled by thrusters on the bottom of the lander. There are also environmental disturbances in the simulation.

Another end goal is to make the controller fault tolerant. If a thruster fails, the reinforcement learning controller should land the Martian lander safely by using the other controllers. The landing could be harder or less accurate, which could be undesirable in certain cases.

1. What is the state of the art of reinforcement learning in space guidance, navigation and control?
2. What are reasons to apply reinforcement learning instead of conventional control methods?
3. What are the challenges of reinforcement learning in space guidance, navigation and control?
4. How could the dynamics of a planetary lander be modeled?
5. How could a 3-DOF planetary landing be completed successfully by training a reinforcement learning agent to land it?
6. What is a good set of learning rate, discount rate, entropy coefficient and clip range to train a 3-DOF lander?
7. How is a 5-DOF lander performing if the same set of learning rate, discount rate, entropy coefficient and clip range are used to train the lander for a control allocation model and thruster model?
8. How is the 5-DOF reinforcement learning controller, trained by the same set of hyper-parameters, performing in comparison with a PD controller in the nominal situation?
9. How is the 5-DOF reinforcement learning controller performing in comparison with a PD controller for a mass and gravitational acceleration different than the controllers were tuned for?
10. How is the 5-DOF reinforcement learning controller performing if one of the thrusters fail.

1.5. Report overview

In chapter 2, a standalone scientific article can be found.

In chapter 3, the most important outcomes of the literature research will be discusses. This chapter will start by introducing the main principles of reinforcement learning. Then an overview will be given of reinforcement learning applications in space guidance navigation and control. Multiple papers have been written were a reinforcement learning algorithm was used to control a spacecraft in a simulation. The planetary landing problem will be explained in more detail. The dynamics behind the planetary landing will be explained and also the conditions for a successful landing will be explained. Also the rendezvous/docking problem will be explained in more detail because it similarities to the planetary landing problem.

Some results are not directly related to the standalone paper. The most important additional results will be summarized in chapter 4.

Before this thesis was written, a preliminary analysis was done on an easier problem, a 2D lunar landing. The goal was to land a lunar lander on an environment with only three degrees of freedom, one rotation and two translations. The outcomes of the preliminary thesis are shown in chapter 5.
This thesis will end with chapter 6. This section will summarize the most important conclusions and give some recommendations for further research.

2

Scientific Article

Reinforcement Learning for a Six Degree of Freedom Martian Landing

Ayman el Ghalbzouri
Control & Simulation
Delft University of Technology
Delft, Netherlands

***Abstract*—This paper presents a simulation of a Martian lander using the Proximal Policy Optimization (PPO) method. The objective is to train an agent to land on Mars using reinforcement learning. Two reinforcement learning (RL) controllers are made. The first one is for a control allocation model where the translations and rotations are controlled independently. A PD controller is made to compare with the RL controller. The PD controller is found to be more accurate. A RL thruster model controller is also made, where the spacecraft is controlled by five thrusters. The translations and rotations are coupled for this controller which makes it more difficult to control. This lander needs more control effort, and is less accurate in tracking the reference speed compared with the RL control allocation controller. It is also less accurate with landing on the designated landing spot.**

I. Introduction

Landing on a planet is one of the last phases of many space missions. The landing should not be too hard to ensure that the equipment is not damaged. The exact landing position also matters in some cases, especially if the landing has to be made near a hazardous area. This may be done by conventional control methods if the environment is well known and if there are no or small environmental disturbances, but this is not always the case. Reinforcement learning methods could in some cases learn to cope with these uncertainties. In the last couple of years, reinforcement learning in planetary landing and rendezvous/docking problems became an interesting framework for aerospace engineers to research.

Multiple conventional control methods exist. Examples are full-state feedback, PID controllers and LQR controllers [1]. Reinforcement learning is a newer promising way to solve control engineering problems. A good reason to use reinforcement learning is that the dynamics don't have to be represented as an ordinary differential equation. In fact, there are no constraints at all in the dynamics representation. Because of this, there are less constraints in the model. Non-linear problems have to be linearized for most of the conventional control methods. This linearized equations are only valid near the trim points. Reinforcement learning agents on the other hand learn a policy without caring if the system dynamics are linear or nonlinear [2].

In recent years, aerospace engineers have become more interested in reinforcement learning [3]. Aerospace engineers have done research in the context of docking problems [4]–[7], satellite attitude control [8], hovering problems [9] and planetary or lunar landings [10]–[17].

The objective of this paper will be to train an agent to land on Mars in a simulation by the PPO method. The performance will be compared with a PD controller. The agent will be controlled by control allocation. Three translational and two rotational motions are controlled independently. The next objective will be to train the spacecraft to land, where the spacecraft is controlled by a thruster model. This problem is more complex because of the fact that the translational and rotational motions are coupled and cannot be controlled independently.

II. Dynamics

In this section the dynamics of the Martian lander will be derived. The dynamics are inspired by the article of Gaudet, Linares and Furfaro [10]. Consider a lander with a shape of a hemisphere. Also consider a body frame of reference, where the origin of the frame is defined as the center of gravity of the lander. The xy-plane is perpendicular to the flat plane of the hemisphere while the z-axis is perpendicular with the flat plane. This is shown in figure 1. The radius of this hemisphere is R. The mass of the hemisphere is m. The center of mass is located $\frac{3}{8}R$ above the flat plane of the hemisphere. The inertia matrix $\mathbf{I}$ is:

$$\mathbf{I} = \begin{bmatrix} \frac{83}{320}mR^2 & 0 & 0 \\ 0 & \frac{83}{320}mR^2 & 0 \\ 0 & 0 & \frac{2}{5}mR^2 \end{bmatrix} \tag{1}$$

There are five inputs, these are F_x, F_y, F_z, M_x and M_y. These forces and moments are continuous. The minimum and maximum forces in the x-direction and y-direction are -2500 N and 2500 N. There is no negative force in the z-direction, the maximum force in the z-direction is 15000 N. The minimum and maximum torques for M_x and M_y are both -1000 Nm and 1000 Nm. The mass is selected from a

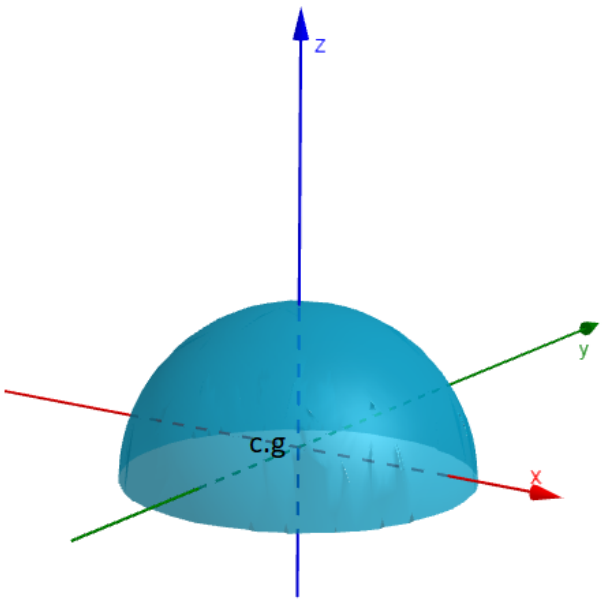


Fig. 1. The coordinate system of the lander. All the axes are drawn in the body frame of the lander and it has a shape of a hemisphere.

uniform distribution between 1900 and 2100 kg.

At the start of each episode, a random wind is generated. The wind is modeled as a force $\boldsymbol{F_{env}}$ with three components. The value of these components are all normally distributed with a mean of 0 N and a standard deviation of 80 N.

The forces F_x, F_y and F_z are all in the body frame. The force in the inertial frame is now defined as:

$$\mathbf{F}^N = \mathbf{C}_B^N \mathbf{F}^B + \mathbf{F}_{env} \tag{2}$$

, with $\mathbf{C}_B^N$ being the body to inertial frame matrix.

The translational dynamics can now be expressed as:

$$\dot{\mathbf{r}} = \mathbf{v} \tag{3}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{F}^N}{m} + \mathbf{g} \tag{4}$$

, with $\mathbf{g}$ being the gravitation acceleration. The x-component and y-component are both 0 m/s^2 and z-component is uniformly distributed between -3.8 and -3.6 m/s^2.
Define a vector $\boldsymbol{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^\intercal$, where p, q and r are the rotational velocities along the body axis. The derivative of this vector is:

$$\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1}\left(\mathbf{M}^B - \omega \times \mathbf{I}\boldsymbol{\omega}\right) \tag{5}$$

The rotational velocities can be calculated by integrating the equation above. The derivatives of the Euler angles are now:

TABLE I
THE INITIAL STATE IS CHOSEN RANDOMLY BY AN UNIFORM DISTRIBUTION

observation	min	max
x(m)	-50	50
y(m)	-50	50
z(m)	500	600
v_x(m/s)	-5	5
v_y(m/s)	-5	5
v_z(m/s)	-45	-35
ϕ(rad)	0	0
θ(rad)	0	0
ψ(rad)	0	0
p(rad/s)	0	0
q(rad/s)	0	0
r(rad/s)	0	0

$$\dot{\varphi} = p + \tan\theta(q\sin\phi + r\cos\varphi) \tag{6}$$

$$\dot{\theta} = q\cos\varphi - r\sin\varphi \tag{7}$$

$$\dot{\psi} = (q\sin\varphi + r\cos\varphi)/\cos\theta \tag{8}$$

The Euler angles can be calculated by integrating equation 6-8.

III. REINFORCEMENT LEARNING MODEL

The observation space is defined as:

$$obs = \begin{bmatrix} x & y & z & v_x & v_y & v_z & \varphi & \theta & \psi & p & q & r \end{bmatrix} \tag{9}$$

The initial conditions for some of these observables were uniformly distributed. The initial conditions are shown in table I.

The agent is trained by making use of the PPO algorithm [18]. This was done within the Stable-Baselines3 framework [19]. The PPO algorithm is an actor-critic method maximizing the following objective function:

$$L^{CLIP}(\theta) = \hat{E}_t\Big[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{10}$$

In this equation, θ stands for a policy parameter, $\hat{E}_t$ is an expectation, ϵ is a hyperparameter, $\hat{A}_t$ is an estimated advantage and r_t is a probability ratio between a new and an old policy. The advantage function A and probability ratio $r_t(\theta)$ are defined as:

$$A = Q(s,a) - V(s) \tag{11}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{12}$$

$Q(s,a)$ and $V(s)$ in equation 11 are both fundamental concepts in reinforcement learning. The action is defined by

the symbol a. The reward per time step is defined as r.

The symbol used for the total discounted return is R. This is the sum of all rewards in the future, where the more further rewards are discounted. This is done by introducing a discount factor γ. This factor is a number between 0 and 1. The value under a policy π for a state is defined as the expected discounted return:

$$V(s) = E[R_t|s_t = s] = E\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\Big|s_t = s\Big] \quad (13)$$

The value function is the expected discounted return given the present state. R_t is defined as the discounted reward on time t while s_t is defined as the state on time t. Because of the fact that the discount factor is a number between 0 and 1, and is raised to the power of k, future rewards are less important. Instead of maximizing the value function, the Q-value shown in equation 14 could be maximized instead. The Q-value is the expected discounted reward given a certain state-action pair:

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R_t \,|\, s_t = s,\, a_t = a] \\ &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s,\, a_t = a\right] \end{aligned} \quad (14)$$

In equation 12, $\pi_\theta(a_t|s_t)$ stands for the probability that action a_t is chosen given being in state s_t.

The first goal was to find a good set of training parameters and reward function. This was done in a 3 degrees of freedom environment. The only inputs were the three translational forces. No moments were present and only translational motion was possible. Four training parameters were varied, the learning rate, discount rate, entropy coefficient and the clip range. A nominal value was set for each training parameter. The nominal learning rate was 0.0003, the nominal discount rate was 0.95, the nominal entropy coefficient was 0.015 and the nominal clip range was 0.20. These nominal values were an educated guess. They were chosen around the values that have been used by other researchers in the field of reinforcement learning. Better nominal values were found by trial and error.

Four or five different values were chosen for each hyperparameter. One of the training parameters were varied while the other three were set equal to the nominal value. Because of the stochastic nature of the training, five training sessions were performed for each set of training parameters. After each training session, 1000 episodes were ran to evaluate the policy. This means that each set of parameters were evaluated by a total of 5000 episodes. The results are shown in table III. The mean win rate, and mean reward are shown. The standard deviation of the win rate and combined standard deviation for the reward after five training sessions is also shown in the table. The best set of hyperparameters is shown in the last row. The conclusion is that the best set of hyperparameters for the learning rate, discount rate, entropy coefficient and clip range are 0.0003, 0.99, 0.005 and 0.20. This is also shown in table III.

An actor and a critic network are used to train the agent. Both networks have three hidden layers. The layers of the actor-network have 240, 150 and 100 nodes. The layers of the critic network have 240, 50 and 10 nodes. The ReLu activation function is used. The number of layers and nodes that had been used were inspired by an article of Gaudet, Linares and Furfaro [10]. They gave formulas that they have used for the number of nodes per layer that were used as a rule of thump This is shown in table II.

The number of nodes per layer were doubled because a ReLu activation function was used instead of a tanh activation function.
The reward function for the three degrees of freedom lander was:

$$r = \begin{cases} -100 & \text{crashing} \\ 400 & \text{landing} \\ 0.0012r_1 + 0.0001r_2 + 0.07r_3 & \text{else} \end{cases} \quad (15)$$

r_1 is a penalty for being off from the landing spot:

$$r_1 = -\sqrt{x^2 + y^2 + (z - z_{c_g}^2)} \quad (16)$$

r_2 is a penalty for control effort:

$$r_2 = -\left(\int |F_x|dt + \int |F_y|dt + \int |F_z|dt\right) \quad (17)$$

r_3 is a penalty for being off from the reference velocity:

$$r_3 = -\sqrt{(v_x - v_{x_{ref}}))^2 + (v_y - v_{y_{ref}}))^2 + (v_z - v_{z_{ref}}))^2} \quad (18)$$

A reference velocity $v_{z_{target}}(z)$ as a function of z is defined. This target velocity is defined to give the spacecraft some feedback about it velocity during the powered descend.

Assume that the agent has to have a constant acceleration in the x-direction, y-direction and z-direction. In reality is does not have to, but if the forces and mass are constant this will be the case. The mass is however not constant because of the fact that fuel is burned during the landing.

It has to land with a velocity of (v_x, v_y, v_z) on position (x, y, z). The equation of motions for each of these directions will be:

$$s_f = s_0 + v_0 \cdot t + \frac{1}{2}at^2 \quad (19)$$

$$v_f = v_0 + at \quad (20)$$

TABLE II
ARCHITECTURE OF THE POLICY- AND VALUE-NETWERK USED BY GAUDET, LINARES AND FURFARO [10].

Layer	**Policy Network**		**Value Network**	
	# units	activation	# units	activation
hidden 1	$10 \cdot$ obs_dim	tanh	$10 \cdot$ obs_dim	tanh
hidden 2	$\sqrt{n_{h1} \cdot n_{h3}}$	tanh	$\sqrt{n_{h1} \cdot n_{h3}}$	tanh
hidden 3	$10 \cdot$ act_dim	tanh	5	tanh
output	act_dim	linear	1	linear

TABLE III
THE MEAN WIN RATE, MEAN REWARD AND COMBINED STANDARD DEVIATION FOR THE WIN RATE ARE CALCULATED. THE RED VALUES INDICATE A VALUE GIVING A MEAN REWARD BETTER THAN THE NOMINAL VALUE.

learning rate	dis rate	ent coef	clip range	mean win rate	win rate std	mean reward	comb std
0.0003	0.95	0.015	0.20	98.0	2.1	247	73
0.000003	0.95	0.015	0.20	0.0	0.0	-246	19
0.00003	0.95	0.015	0.20	80.9	40.1	124	242
0.003	0.95	0.015	0.20	77.6	43.4	32	158
0.03	0.95	0.015	0.20	0.3	0.8	-299	140
0.0003	0.91	0.015	0.20	43.1	49.8	-57	257
0.0003	0.93	0.015	0.20	71.8	43.8	71	225
0.0003	0.97	0.015	0.20	99.1	0.9	266	50
0.0003	0.99	0.015	0.20	99.4	1.0	274	41
0.0003	0.95	0.000	0.20	38.9	53.3	19	249
0.0003	0.95	0.005	0.20	98.2	3.0	257	73
0.0003	0.95	0.010	0.20	99.6	0.2	255	45
0.0003	0.95	0.020	0.20	87.3	20.9	129	208
0.0003	0.95	0.025	0.20	98.8	1.7	207	65
0.0003	0.95	0.025	0.10	96.6	4.8	216	99
0.0003	0.95	0.025	0.15	91.8	17.3	147	225
0.0003	0.95	0.025	0.25	76.3	42.8	125	225
0.0003	0.95	0.025	0.30	76.1	42.8	47	214
0.0003	0.99	0.005	0.20	99.9	0.3	282	24

where s_f is the final desired position for the x, y or z coordinate, v_0 is the initial velocity, t is the time, a is the acceleration, v_f is the desired end velocity and v_0 the initial velocity. The relation between velocity and position will be:

$$v_z = \pm\sqrt{|s_f^2 + 2ax - 2as_f|} \tag{21}$$

Lets assume that the acceleration in the x, y and z direction will be (a_x, a_y, a_z). The desired velocity is $(0, 0, -v_{z_f})$. It is not desirable if the aircraft lands with a non-zero speed in the x-direction or y-direction. The desired end position is $(0, 0, \frac{3}{8}R)$. Keep in mind that the center of gravity of the spacecraft is $\frac{3}{8}R$ m above the bottom of the spacecraft. The reference velocities of the spacecraft will now be:

$$v_X = \begin{cases} \sqrt{2a_x x} & x < 0 \\ -\sqrt{2a_x x} & x \geq 0 \end{cases} \tag{22}$$

$$v_y = \begin{cases} \sqrt{2a_y y} & y < 0 \\ -\sqrt{2a_y y} & y \geq 0 \end{cases} \tag{23}$$

Landing with a lateral speed component is not desired. Because of this, the reference value for v_x and v_y will be zero of the altitude is less than 10 m.

$$v_z = -\sqrt{|v_f^2 + 2az - 2a_z\frac{3}{8}R|} \tag{24}$$

A successful landing and a crash should be defined. A crash is defined as: $|x| > 1500$ OR $|y| > 1500$ OR $|z| > 2500$ OR $|\phi| > \pi/4$ OR $|\theta| > \pi/4$ OR ($z < 0$ AND $||\mathbf{v}|| > 2$) OR ($z < 0$ AND $||\boldsymbol{\omega}|| > 0.3$) OR ($z < \frac{3}{8}R$ AND $\phi > 0.2$) OR ($z < 0$ AND $\theta > 0.2$).

This ensures that the episode is stopped if it is moving to far from the designated landing spot or if either the roll or pitch are larger than 45°. It also ensures that an episode is stopped if the lander lands with a speed or rotational speed that is too high, 2 m/s for the speed and 0.4 rad/s for rotational speed. The last criteria are that ϕ and θ should never be larger than 0.2 rad if the spacecraft lands. The episode also ends if it takes more than 1000 time-steps to land to ensure that it will not hover for a long time.

A successful landing is defined as: $z \leq \frac{3}{8}R$ AND $||\mathbf{v}|| < 2$ AND $||\boldsymbol{\omega}|| < 0.3$ AND $\phi < 0.2$ AND $\theta < 0.2$. This means that the spacecraft had to land within certain speed limits, rotational speed limits and attitude limits.

The same hyper parameters were used to train six degree of freedom control models that will be described later . In these models, the spacecraft is able to translate in three directions and also rotate in three directions, although only pitch and roll are controlled directly. A more detailed description of these models will be given later. The initial conditions are shown in

TABLE IV
THE INITIAL STATE IS CHOSEN RANDOMLY BY AN UNIFORM DISTRIBUTION

observation	min	max
x(m)	-50	50
y(m)	-50	50
z(m)	500	600
v_x(m/s)	-5	5
v_y(m/s)	-5	5
v_z(m/s)	-45	-35
ϕ(rad)	$-\pi/8$	$\pi/8$
θ(rad)	$-\pi/8$	$\pi/8$
ψ(rad)	$-\pi/4$	$\pi/4$
p(rad/s)	0	0
q(rad/s)	0	0
r(rad/s)	0	0

table IV. Also the reward function was almost the same:

$$r = \begin{cases} -100 & \text{crashing} \\ 400 & \text{landing} \\ 0.0012r_1 + 0.0001r_2 + 0.07r_3 + 0.5r_4 + r_5 & \text{else} \end{cases} \tag{25}$$

r_1 is a penalty for being off from the landing spot:

$$r_1 = -\sqrt{x^2 + y^2 + (z - z_{c_g}^2)} \tag{26}$$

r_2 is a penalty for control effort:

$$r_2 = -\int \left(|\mathbf{F}| + |\mathbf{M}|\right) dt \tag{27}$$

r_3 is a penalty for being off from the reference velocity:

$$r_3 = -\sqrt{(v_x - v_{x_{ref}}))^2 + (v_y - v_{y_{ref}}))^2 + (v_z - v_{z_{ref}}))^2} \tag{28}$$

r_4 is a penalty for a nonzero pitch and roll:

$$r_4 = -\sqrt{\varphi^2 + \theta^2} \tag{29}$$

r_5 is a bonus reward for surviving an episode:

$$r_5 = 1 \tag{30}$$

The last term ensures that the lander will not crash at the start of the episode to prevent getting negative rewards.

The Delft Blue was used to train the lander because of the running time of the simulation [20]. The results are shown in figure 2-7. It is observed in figure 2 that the spacecraft is not able to land exactly on the designated landing spot. In figure 3 can be seen that the lateral components of the reference velocity drops to 0 if the altitude is less than 10 m. From figure 4 it could be concluded that the spacecraft lands with pitch and roll, although pitch and roll are not desired. The goal is to land without pitch and roll.

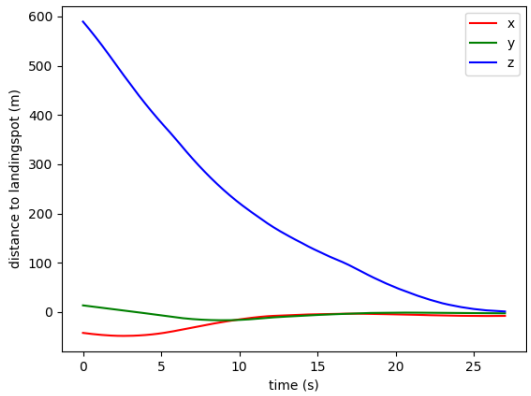


Fig. 2. The position of the lander controlled by the control allocation RL controller. The designated landing spot is (0,0,0).

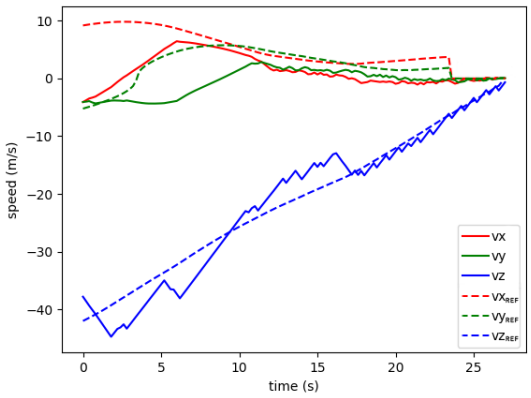


Fig. 3. The speed of the lander controlled by the by the control allocation RL controller. The lateral components of the reference velocity are set to 0 if the attitude is less than 10 m because of the fact that landing with a radial velocity is undesired.

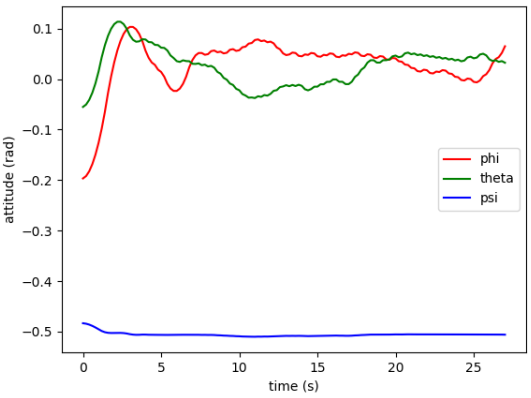


Fig. 4. The attitude of the lander controlled by the control allocation RL controller. The goal is to land without pitch and roll.

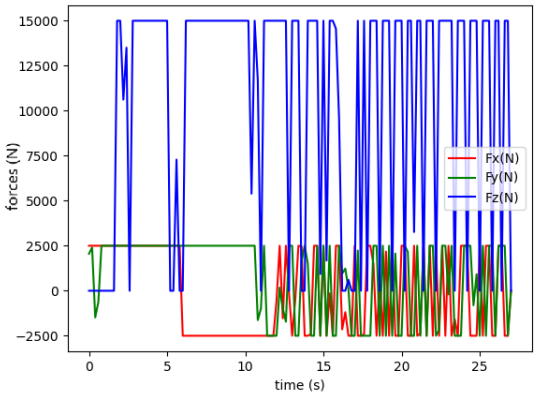


Fig. 5. The forces of the lander controlled by the control allocation RL controller. The maximum thrust is 15000 N in the lateral direction and 2500 N for both radial forces.

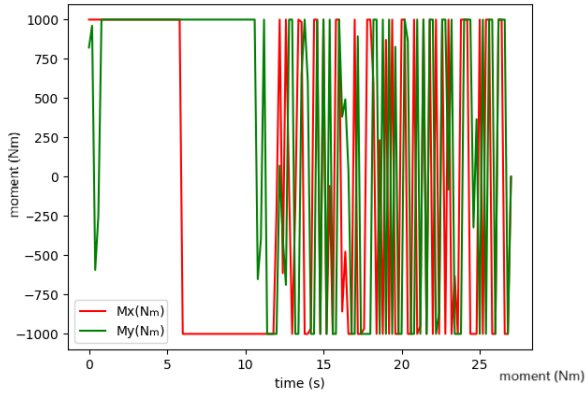


Fig. 6. The moments of the lander controlled by the control allocation RL controller. There is no moment to control the yaw directly.

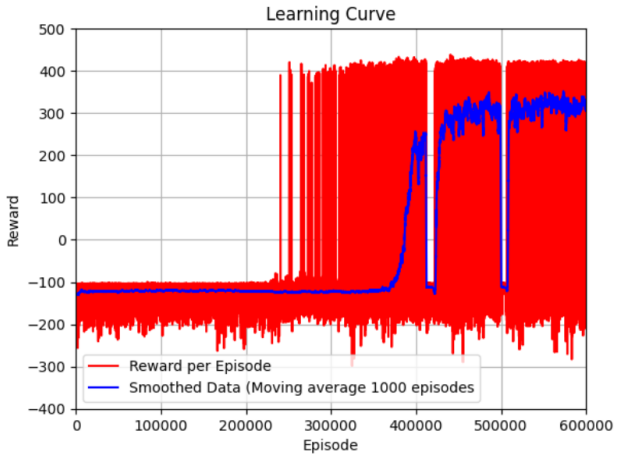


Fig. 7. The learning curve of the lander controlled by the control allocation RL controller. The learning curve stabilizes after about 450 000 episodes.

IV. Tuning of the PD controller

A PD controller was made to land the spacecraft. This controller was made to use as a baseline controller and to eventually compare with a reinforcement learning controller. F_x, F_y, F_z, M_x and M_y were controller by the PD controller. Because of symmetry, the same gains were chosen for F_x and F_y. This is also true for M_x and M_y.

The goal of the PD controller was to track the reference velocity and to have zero roll and pitch. The error in the reference speeds were defined as:

$$e_{v_x} = v_{x_{ref}} - v_x \tag{31}$$

$$e_{v_y} = v_{y_{ref}} - v_y \tag{32}$$

$$e_{v_z} = v_{z_{ref}} - v_z \tag{33}$$

The error in pitch and yaw were defined as:

$$e_\varphi = \varphi_{ref} - \varphi \tag{34}$$

$$e_\theta = \theta_{ref} - \theta \tag{35}$$

Note that φ_{ref} and θ_{ref} are both 0. The error derivatives are calculated numerically:

$$\dot{e}_{v_x} = (e_{v_{x_i}} - e_{v_{x_{i-1}}})/dt \tag{36}$$

$$\dot{e}_{v_y} = (e_{v_{y_i}} - e_{v_{y_{i-1}}})/dt \tag{37}$$

$$\dot{e}_{v_z} = (e_{v_{z_i}} - e_{v_{z_{i-1}}})/dt \tag{38}$$

$$\dot{e}_\varphi = (e_{\varphi_i} - e_{\varphi_{i-1}})/dt \tag{39}$$

$$\dot{e}_\theta = (e_{\theta_i} - e_{\theta_{i-1}})/dt \tag{40}$$

Six gains are defined. These are $K_{p_{F_{xy}}}$, $K_{d_{F_{xy}}}$, $K_{p_{F_z}}$, $K_{d_{F_z}}$, $K_{p_{M_{xy}}}$ and $K_{d_{M_{xy}}}$. The normalized forces in the inertial frame are now defined as:

$$\mathbf{F}_{N_{normalized}} = \begin{bmatrix} K_{p_{F_{xy}}} \cdot e_{v_x} + K_{d_{F_{xy}}} \cdot \dot{e}_{v_x} \\ K_{p_{F_{xy}}} \cdot e_{v_y} + K_{d_{F_{xy}}} \cdot \dot{e}_{v_y} \\ K_{p_{F_z}} \cdot e_{v_z} + K_{d_{F_z}} \cdot \dot{e}_{v_z} \end{bmatrix} \tag{41}$$

The normalized forces in the body frame are:

$$\mathbf{F}_{b_{normalized}} = \mathbf{C}_N^B \mathbf{F}_{N_{normalized}} \tag{42}$$

,where $\mathbf{C}_N^B$ is the inertial to body frame matrix. The outputs the PD controller are now defined as:

$$\mathbf{u} = \begin{bmatrix} clip[\mathbf{F}_{b_{normalized_1}}, -1, 1] \\ clip[\mathbf{F}_{b_{normalized_2}}, -1, 1] \\ clip[\mathbf{F}_{b_{normalized_1}}, 0, 1] \\ clip[(K_{p_{M_{xy}}} \cdot e_\varphi + K_{d_{M_{xy}}} \cdot \dot{e}_\varphi), -1, 1] \\ clip[(K_{p_{M_{xy}}} \cdot e_\theta + K_{d_{M_{xy}}} \cdot \dot{e}_\theta), -1, 1] \end{bmatrix} \tag{43}$$

Note that the upward force in the body frame can't be negative. The maximum force the radial direction is 2500 N, the maximum force in the upward direction is 15000 N and the maximum moments are 1000 Nm. The normalized forces and moments are multiplied by these numbers to convert them to real values for the forces and moments.

The gains of the PD controller have to be tuned. This is done experimentally. Three different values are chosen for each gain. For every possible set of gains, a hundred episodes are ran. New values for the gains are chosen around the previous most optimal set of gains to find a better set of gains. The gains are set to be $K_{p_{F_{xy}}} = 3$, $K_{d_{F_{xy}}} = -1$, $K_{p_{F_z}} = 11$, $K_{d_{F_z}} = -1$, $K_{p_{M_{xy}}} = 14$ and $K_{d_{M_{xy}}} = 14$

V. Comparison of the RL controller with PD controller

The RL controller was compared with the PD controller. A thousand episodes were ran for the PD controller and the RL controller each. The compared variables were the win rate, landing accuracy, force control effort moment control effort and time to land. The results are shown in the table V. The success rate of both controllers seems to be similar. Cohen's d is used to quantify the differences between PD and RL controller [22]. According to the thumb of rules of Cohen's d, the difference in the reward are small. The difference in landing accuracy, control effort and time to land are all large.

VI. Six Degree of Freedom Thruster Model

A six degree of freedom thruster model was also made. In this model, the thee translational and rotational motions are not controller independently. Five thrusters are placed on the bottom of the lander. The rotational and translational motions are now coupled. In this section the dynamics of the lunar lander will be derived. Consider a hemisphere with five thruster on its bottom. This thruster are placed in a circle with radius r. The reference frame of an arbitrary thruster is shown in figure 8.

The top view of the lander is shown in figure 9. The thrusters are separated by $\frac{360}{5} = 72°$. All thruster are positioned $\frac{3}{8}R$ under the center of gravity. The first thruster T_1 lies on the x-axis by definition. The position of the thrusters can be given by polar coordinates. The angle θ_t for each thruster is defined as the angle from the reference position. The Cartesian coordinates of the thrusters are shown in table VI.

Assume that there is an angle of ϕ_t between the unit direction vector of the thrusters and the vertical axis, which was chosen to be 30 °. This is shown in figure 10. From figure 10 it can be concluded that the z-component of the unit direction vector and radial component are:

$$d_z(\theta_t, \phi_t) = \cos\phi_t \tag{44}$$

$$d_r(\theta_t, \phi_t) = \sin\phi_t \tag{45}$$

The radial component can be divided in a x-component and a y-component. The radial component is pointing from the thruster to the center of the half circle. From figure 9 it can be concluded that:

$$d_x(\theta_t, \phi_t) = -a\cos\theta_t \tag{46}$$

$$d_y(\theta_t, \phi_t) = -a\sin\theta_t \tag{47}$$

, where a is a scaling factor to make sure that vector $\mathbf{d}_i$ is an unit vector. The minus sign in front of a ensures that d_r always points to the Center of the lander. From Pythagoras theorem it can be derived that:

$$\begin{aligned} d_x(\theta_t, \phi_t)^2 + d_y(\theta_t, \phi_t)^2 &= d_r(\theta_t, \phi_t)^2 \\ (-a\cos\theta_t)^2 + (-a\sin\theta_t)^2 &= (\sin\phi_t)^2 \\ a^2(\cos^2\theta_t + \sin^2\theta_t) &= \sin^2\phi_t \\ a^2 &= \sin^2\phi_t \\ a &= \sin\phi_t \end{aligned} \tag{48}$$

Now unit vector $\mathbf{d}_i$ may be written as:

$$\mathbf{d}_i = \begin{bmatrix} -\sin\phi_t\cos\theta_t \\ -\sin\phi_t\sin\theta_t \\ \cos\phi_t \end{bmatrix} \tag{49}$$

The force and moment in the lander body frame are:

$$\mathbf{F}^B = \sum_{i=1}^{5} d_i T_i \tag{50}$$

$$\mathbf{M}^B = \sum_{i=1}^{5} \mathbf{r}_i \times \mathbf{F}_i^B \tag{51}$$

Otherwise, the dynamics are the same as described at the beginning of this chapter. The forces and the moments are integrated the same way as in the control allocation model.
The mass is also changing because of the burning of fuel. This can be expressed by the following equation:

$$\dot{m} = -\frac{\sum_{i=1}^{5} ||\mathbf{F}_i^B||}{I_{sp} g_{ref}} \tag{52}$$

, where I_{sp} stands for the specific impulse and g_{ref} for a reference gravitational acceleration, typical 9.8 m/s^2.
The reward function is almost exactly the same as the control allocation model. Only the penalty for the control effort is modified.

TABLE V
THE PD AND RL CONTROLLER COMPARED FOR THE CONTROL ALLOCATION MODEL. THE PD CONTROLLERS SEEMS TO OUTPERFORM THE RL CONTROLLER. A THOUSAND EPISODES WERE RAN TO EVALUATE BOTH MODELS.

	PD		**RL**		**Cohen's d**
	Mean	Std. dev.	Mean	Std. dev.	
Win rate	98.2%	–	98.4%	–	–
Reward	430.9	67.1	403.1	69.5	0.41
Distance (m)	0.45	0.41	7.16	0.95	-9.17
$\int Fdt$ (Ns)	379346	14158	400129	13770	-1.49
$\int Mdt$ (Nms)	6596	12476	48228	2043	-4.66
Time (s)	25.6	0.7	27.2	0.9	-1.98

TABLE VI
THE POSITION OF THE THRUSTERS RELATIVE TO THE LANDER CENTER OF MASS.

	thruster 1	thruster 2	thruster 3	thruster 4	thruster 5
x-coordinate	$r\cos(0°)$	$r\cos(72°)$	$r\cos(144°)$	$r\cos(216°)$	$r\cos(288°)$
y-coordinate	$r\sin(0°)$	$r\sin(72°)$	$r\sin(144°)$	$r\sin(216°)$	$r\sin(288°)$
z-coordinate	$-\frac{3}{8}R$	$-\frac{3}{8}R$	$-\frac{3}{8}R$	$-\frac{3}{8}R$	$-\frac{3}{8}R$

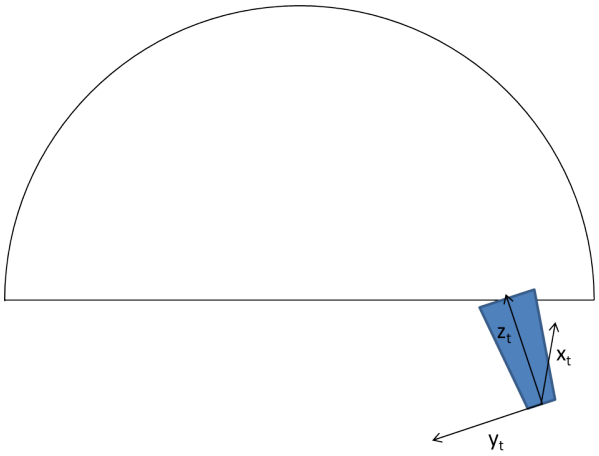


Fig. 8. The reference frame for an arbitrary thruster. The thruster is delivering a force in its z-axis. The y-axis is perpendicular to the z-axis and is pointing to middle of the lander. The x-axis is pointing in tangential direction.

The action space is a continuous action space. The maximum thrust is T_{max} where $T_{max} = 2500$ N. The action space is defined as:

$$a = \mathbf{T} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 & T_5 \end{bmatrix}^\intercal \tag{53}$$

, where $T_i \in \begin{bmatrix} 0 & T_{max} \end{bmatrix}$

The penalty for the control effort is:

$$r_2 = -\Big(\int |T_1|\, dt + \int |T_2|\, dt + \int |T_3|\, dt + \int |T_4|\, dt + \int |T_5|\, dt \Big) \tag{54}$$

VII. TRAINING OF THE THRUSTER MODEL AGENT

The agent was trained by the PPO method. A total of 45 million time steps were ran to train the agent. The win rate, total reward, control effort and time to land were all calculated. This is shown on table VII. The agent controlled by the thruster model seems to be less consistent in comparison with the agent controlled by the control allocation modem. This can be concluded by the fact that the win rate is lower and the large standard deviation of the reward. The landing accuracy and time to land are also larger.

TABLE VII
THE WIN RATE, TOTAL REWARD, CONTROL EFFORT AND TIME TO LAND FOR THE LANDER CONTROLLED BY THE THRUSTER MODEL. THE EVALUATION WAS DONE BY RUNNING A THOUSAND EPISODES.

	mean	standard deviation
win rate	57.6%	
reward	159.9	255.8
distance (m)	19.27	5.62
$\int Fdt$ (Ns)	374250	34013
time (s)	35.4	4.8

The position, speed, attitude, and forces were plotted for a successful landing. Also the learning curve was plotted This is all shown in figure 11-15. From figure 11 could be concluded that the agent controlled by the thruster model is

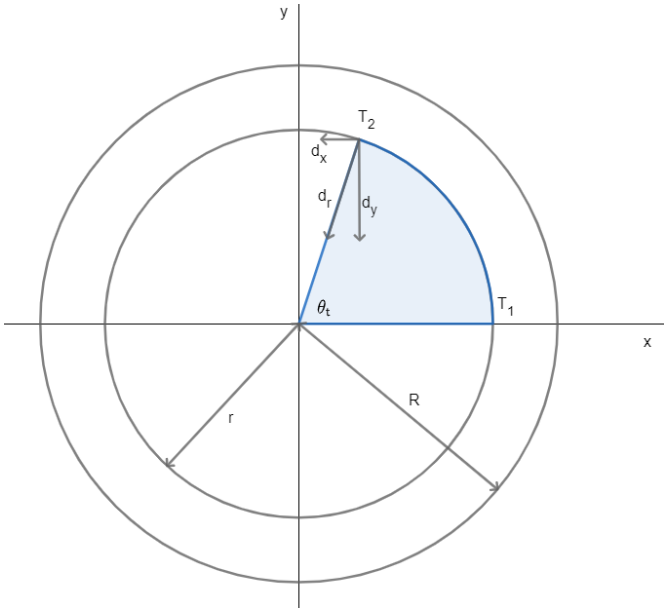


Fig. 9. Top view of the lander. The thrusters are separated by 72°. The angle θ_t for each thruster is defined as the angle from the reference position. The first thruster T_1 lies on the x-axis by definition. The lander has a radius of R and the thrusters all are placed in a circle with radius r. The thrusters are delivering a force in the direction of a unit vector d which has a radial component and a component in the z-direction. The radial component can be decomposed in a component in the x-direction and a component in the y-direction.

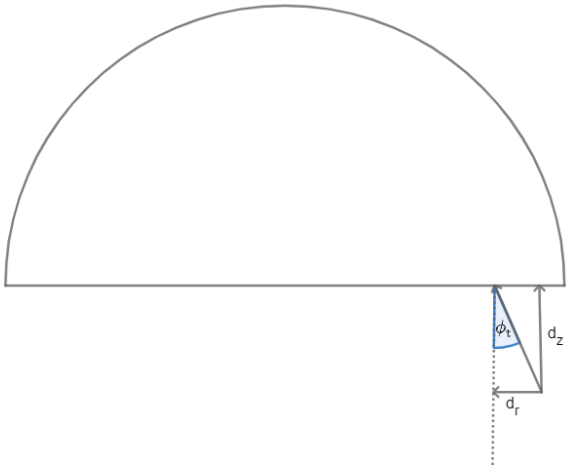


Fig. 10. The side view of the lander with unit vector **d**. This unit gives the direction of the force that is delivered by the thruster. There is an angle of ϕ_t between the unit direction vector of the thrusters and the vertical axis.

less accurate. From figure 12 and 13 could be concluded that this model is also less accurate in following the reference velocity and reference attitude. In figure 15 can be seen that the thruster model agent needs more time before the reward stabilizes.

The win rate for the thruster model was lower than that of the control allocation model, which was also trained by RL. This could be due to the increased complexity of the problem. It was also suspected that the chance of a successful landing depends on the initial conditions. This was tested by making the width of the ranges of the initial conditions smaller and reevaluating the win rate. The win rate converges to around 80% if the win rate is made smaller. This is shown in figure 16.

It was also attempted to train the agent to cope with failure. One of the thrusters was turned off. The goal was to land the spacecraft with the remaining five engines. This training was done with 45 million time steps. The evaluation was done with 1000 episodes. Not one of the 1000 attempts to land

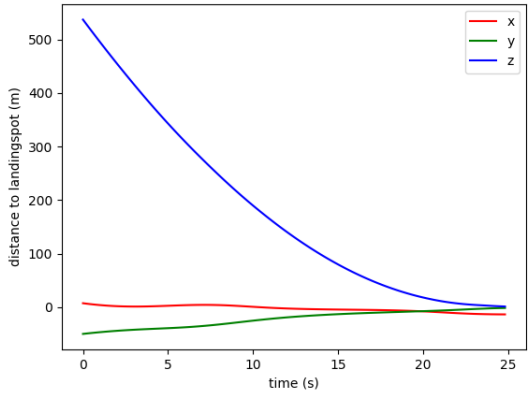


Fig. 11. The position of the lander controlled by the thruster model controller. The designated landing spot is (0,0,0). The thruster model controller is less accurate compared with the control allocation controller.

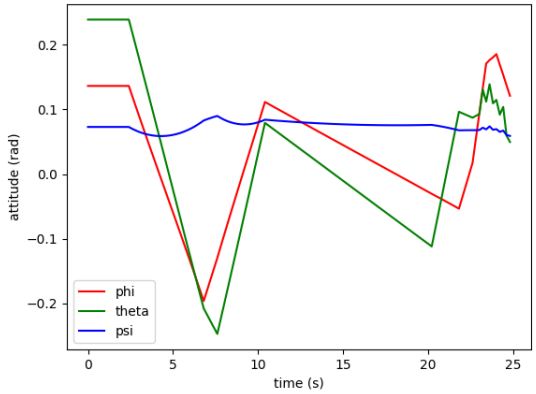


Fig. 13. The attitude of the lander controlled by the thruster model controller. It is harder to train the thruster model agent to land without pitch and roll.

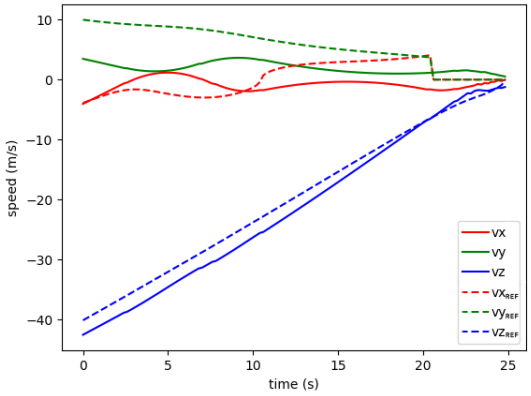


Fig. 12. The speed of the lander controlled by the thruster model controller. Landing with a radial velocity is undesired again.

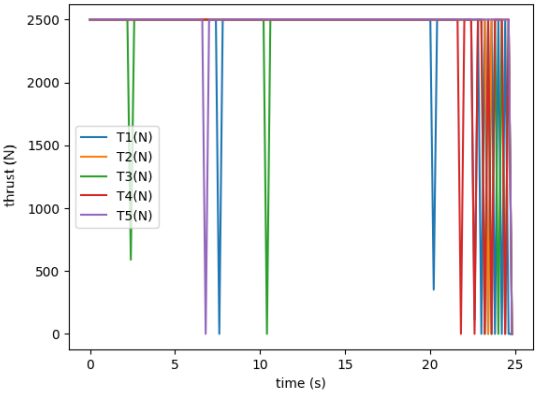


Fig. 14. The forces of the lander controlled by the thruster model controller. All thrusters have a maximum thrust of 2500 N.

was successful.

The agent was trained already with five fully working thrusters. If a thruster is set to deliver a maximum of 75% thrust controlled by the existing RL controller, the win rate was 0.2% for a thousand episodes. The agent was also trained to cope with the situation that one of the engines is able to deliver 75% thrust. This training was also done with 45 million time steps and evaluated with 1000 episodes. The outcome was that 13.3% of the landings was successful.

VIII. Conclusions and Recommendations

The objective of this research was to make a robust and fault-tolerant reinforcement learning controller. The problem is similar to a tracking problem, because a position and reference velocity have to be tracked. Tracking problems could also be solved by a PD controller. A reinforcement learning controller and a PD controller have been made to solve a control allocation planetary landing problem. The PD controller was found to perform much better in landing accuracy, control effort and time to land if Cohen's d was considered.

A PD controller is easy to design for a control allocation problem. All translations and moments could be controlled separately. This is not the case for a thruster model problem. Translational and rotational motions are coupled.

A reinforcement learning controller is made to train an agent to land an lander controlled by a thruster model. It turned out to cost more time to train a lander controlled by a thruster model than a control allocation model. The thruster model agent was less successful in landing the spacecraft. The chance of a successful landing turns out to depend on the initial conditions of the lander.

A spacecraft able to cope with thruster failure was attempted

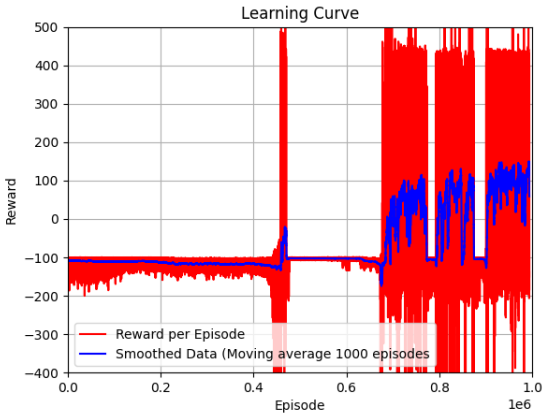


Fig. 15. The learning curve for the thruster model. The learning curve stabilized after about 700 000 episodes.

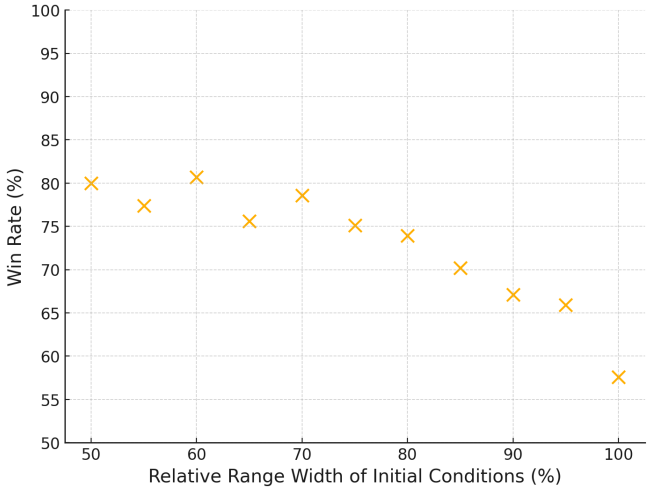


Fig. 16. The win rate for the thruster model increases if the ranges for the initial conditions are made smaller. This proves that the win rate depends on the initial condition..

to be made. One of the thrusters was turned off. Gaudet, Linares and Furfaro showed that four thrusters are enough to control a lander [10]. The spacecraft was however not able to land safely with four thrusters. This may be because a lack of symmetry in the problem unlike the problem of Gaudet, Linares and Furfaro.

The reference velocity was derived by the assumption that the acceleration has to be constant. This is the case if the forces and lander mass are constant. The lander mass is however not constant because fuel is burned. In the future, a reference velocity that depends on the lander mass could be used instead.

Further research have to be done to create a fault-tolerant controller and to improve the landing success rate of the thruster model reinforcement learning controller. It was however shown that a RL controller is better able to cope with 75% if it is trained to do so, compared with a RL controller were the agent is trained by using fully working controllers.

A possible way to create a more fault tolerant controller could be to make a thruster model with six thrusters instead of five. In the case that one of the controllers fail, five controllers all left. However, a thruster addition is more time-consuming during the training, because of the curse of dimensionality. Faster computers should be used or the training should be done on parallel environments by using multiple CPU's.

Another option to create a fault tolerant controller would be the introduction of curriculum learning [21]. The agent is trained in an adapting environment. At the start of the training session there is no failed thruster. The failure is introduced step by step by decreasing the maximum thrust for one of the thrusters.

The success rate of the lander depends on the initial conditions. The success may be improved if it would be further researched for which initial conditions the controller performs the worst. These initial conditions could be chosen with a higher probability during the training.

REFERENCES

[1] Ogata, K. (2010). Modern Control Engineering. Upper Saddle River, NJ, USA: Prentice Hall.

[2] Yang, X., Liu, D., & Huang, Y. (2013). Neural-network-based online optimal control for uncertain non-linear continuous-time systems with control constraints. IET Control Theory & Applications, 7(17), 2037–2047. https://doi.org/10.1049/iet-cta.2013.0472

[3] Izzo, D., Märtens, M. & Pan, B. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. Astrodyn 3, 287–299 (2019). https://doi.org/10.1007/s42064-018-0053-6

[4] Oestreich, Charles E., e.a. 'Autonomous Six-Degree-of-Freedom Spacecraft Docking Maneuvers via Reinforcement Learning'. arXiv:2008.03215 [cs, eess], augustus 2020. arXiv.org, http://arxiv.org/abs/2008.03215.

[5] Oestreich, C. E., Linares, R., & Gondhalekar, R. (2021). Autonomous Six-Degree-of-Freedom Spacecraft Docking with Rotating Targets via Reinforcement Learning. Journal of Aerospace Information Systems, 1–12. https://doi.org/10.2514/1.i010914

[6] Broida, J., & Linares, R. (2019). Spacecraft Rendezvous Guidance in Cluttered Environments Via Reinforcement Learning. In 29th AAS/AIAA Space Flight Mechanics Meeting January. Ka'anapali, HI: American Astronautical Society, Univelt.

[7] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning, second edition: An Introduction (Adaptive Computation and Machine Learning series) (second edition). Bradford Books.

[8] W. M. van Buijtenen, G. Schram, R. Babuska and H. B. Verbruggen, "Adaptive fuzzy control of satellite attitude by reinforcement learning," in IEEE Transactions on Fuzzy Systems, vol. 6, no. 2, pp. 185-194, May 1998, doi: 10.1109/91.669012

[9] Gaudet, B., Linares, R., & Furfaro, R. (2020). Six degree-of-freedom body-fixed hovering over unmapped asteroids via LIDAR altimetry and reinforcement meta-learning. Acta Astronautica, 172, 90-99.

[10] Gaudet, B., Linares, R., & Furfaro, R. (2020). Deep reinforcement learning for six degree-of-freedom planetary landing. Advances in Space Research, 65(7), 1723–1741. https://doi.org/10.1016/j.asr.2019.12.030

[11] Acikmese, B., & Ploen, S. R. (2007). Convex Programming Approach to Powered Descent Guidance for Mars Landing. Journal of Guidance, Control, and Dynamics, 30(5), 1353–1366. https://doi.org/10.2514/1.27553

[12] D'Souza, C., & D'Souza, C. (1997). An optimal guidance law for planetary landing. In Guidance, Navigation, and Control Conference (p. 3709).
[13] Dunlap, K., Mote, M., Delsing, K., & Hobbs, K. L. (2022). Run time assured reinforcement learning for safe satellite docking. In AIAA SCITECH 2022 Forum (p. 1853).
[14] Scorsoglio, A., D'Ambrosio, A., Ghilardi, L., Furfaro, R., Gaudet, B., Linares, R., & Curti, F. (2020, August). Safe Lunar landing via images: A Reinforcement Meta-Learning application to autonomous hazard avoidance and landing. In Proceedings of the 2020 AAS/AIAA Astrodynamics Specialist Conference, Virtual (pp. 9-12).
[15] Scorsoglio, A., D'Ambrosio, A., Ghilardi, L., Gaudet, B., Curti, F., & Furfaro, R. (2022). Image-Based Deep Reinforcement Meta-Learning for Autonomous Lunar Landing. Journal of Spacecraft and Rockets, 59(1), 153-165.
[16] Gaudet, B., & Furfaro, R. (2022). Integrated Guidance and Control for Lunar Landing using a Stabilized Seeker. In AIAA SCITECH 2022 Forum (p. 1838).
[17] G. Ciabatti, S. Daftry and R. Capobianco, "Autonomous Planetary Landing via Deep Reinforcement Learning and Transfer Learning," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2021, pp. 2031-2038, doi: 10.1109/CVPRW53098.2021.00231.
[18] Schulman, John, e.a. 'Proximal Policy Optimization Algorithms'. arXiv:1707.06347 [cs], augustus 2017. arXiv.org, http://arxiv.org/abs/1707.06347
[19] Stable-Baselines3 team. *PPO — Stable Baselines3 Documentation*. Stable-Baselines3. https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html
[20] Delft High Performance Computing Centre (DHPC), *DelftBlue Supercomputer (Phase 2)*, 2024. [Online]. Available: https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2
[21] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009, June). Curriculum learning. In Proceedings of the 26th annual international conference on machine learning (pp. 41-48).
[22] Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

3

Literature Research [1]

3.1. Reinforcement Learning

3.1.1. Introduction

Reinforcement learning is an area within machine learning. The goal is to learn how to map a state to an action in order to optimize a problem. The reinforcement learner is not told directly which action it had to perform. The reinforcement learning agent gets a reward instead for a certain action. The goal is to map every possible state to an action in order to maximize the expected reward in the future.

In this chapter a short introduction will be given in reinforcement learning. This chapter will be theoretical and is used to introduce the general ideas and concepts of reinforcement learning algorithms.

3.1.2. Elements of reinforcement learning

Sutton and Barto define four elements of reinforcement learning [22]. These elements are a policy, a reward signal, a value function and a model. The last element is optional.

A policy is a mapping from a state to an action. It tells how an agent should respond for every situation it could encounter. A policy could be deterministic but it could also be stochastic. In a deterministic policy, a certain state is always mapped to the same action with a probability of 100%. In a stochastic policy a certain state is mapped to an action according to a probability distribution.

A reward signal is used to give an agent some feedback for a certain action. It is used to define a goal for an agent. The policy may be changed for a state-action pair if an action turns out to lead to a low or high total future reward. The actions that will lead to a high expected future reward will become more likely. The interaction between an agent an it environment in reinforcement learning is shown in figure 3.1[22].

A reward could be given at the end of an episode but in many cases intermediate reward are given every time step. This makes learning faster in most cases. The reward function only tells what a good action is in the short-term. However, the goal is to maximize the total future reward. The value function defines what a good choice is in the long term. The value function is the expected return if the agent is in a certain state. The expected return will be explained mathematically later in this chapter. By making use of the value function, an action that gives a high reward immediately but a relatively low reward in the future, is considered a "bad" action.

The final element is a model. Not all reinforcement learning algorithms make use of a model. A model can be used to predict the next state and reward, given a current state and an action. Model-free algorithms do learn the optimal behavior by trial-and-error. Model-based algorithms on the other hand, do learn by planning. In the case of planning, possible future situations are considered before they are experienced.

[1] This chapter has been graded as part of the literature study course

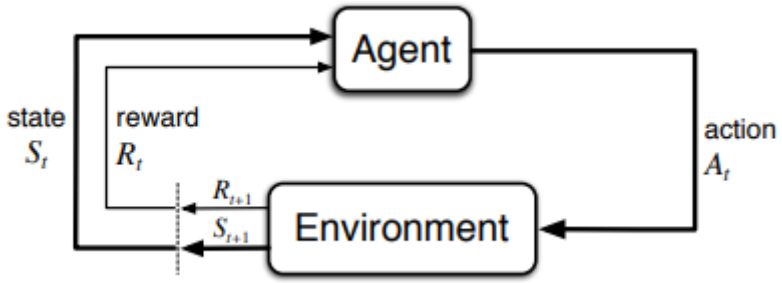


Figure 3.1: The interaction between an agent and it environment in reinforcement learning [22]. The action of the agent depends on the current state and policy. The state is then updated and a reward is given to the agent. The policy is then updated.

3.1.3. Parameters in reinforcement learning

In this section two parameters used in reinforcement learning will be introduced, the parameters ϵ which is used in the ϵ-greedy algorithm and the discount factor γ [22].

Before the introduction of this parameters, some variables will be defined. The state is defined by the symbol s. The set of all possible states is S. The state could also be multidimensional. In that case, the state is described by a vector. The number of possible states grows exponentially with the number of dimensions. This is called "the curse of dimensionality".

The action is defined by the symbol a. Also the action can be multidimensional. This is for example the case for a robot with multiple actuators. Each actuator adds a dimension in the action space. The set of all possible actions at state s is $A(s)$.

The reward per time step is defined as r. Like explained earlier, the goal of reinforcement learning is not to maximize the immediate reward but the total discounted return. The symbol used for the total discounted return is R. This is the sum of all rewards in the future, where the more further rewards are discounted. This is done by introducing a discount factor γ. This factor is a number between 0 and 1. The value under a policy π for a state is defined as the expected discounted return:

$$V_\pi(s) = E_\pi[R_t|s_t = s] = E_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\Big|s_t = s\Big] \tag{3.1}$$

The value function is the expected discounted return given the present state. R_t is defined as the discounted reward on time t while s_t is defined as the state on time t. Because of the fact that the discount factor is a number between 0 and 1, and is raised to the power of k, future rewards are less important. Instead of maximizing the value function, the Q-value shown in equation 3.2 could be maximized instead. The Q-value is the expected discounted reward given a certain state-action pair:

$$Q_\pi(s,a) = E_\pi[R_t|s_t = s, a_t = a] = E_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\Big|s_t = s, a_t = a\Big] \tag{3.2}$$

The agent has to make a trade-off between exploration and exploitation while learning. While exploring, the agent takes the best known action, meaning that this action gives the best expected discounted reward. There may also be an unknown better actions. The agent has to explore to learn about these actions. A policy where the agent only chooses the best known action is called a greedy policy. A policy where actions are random is called a random policy. A hybrid form of these types of policy is an ϵ-greedy policy. This policy is a common to train a reinforcement learning agent.

A value for ϵ has to be chosen. This number has to be between 0 and 1. At every time step, a random number between 0 and 1 is generated. If the random number is lower than ϵ, a random action is chosen. Otherwise, the best known action is chosen. The lower the value for ϵ, the greedier the agent.

3.1.4. Discrete reinforcement learning methods

Multiple reinforcement learning methods do exist. In this section, some discrete reinforcement learning methods will be explained. These methods are only suitable if the state-space and action-space are discrete. If the state-space or action-space is continuous, other methods have to be used. These methods will be introduced later in this chapter.

Examples of discrete reinforcement learning methods are dynamic programming methods, Monte Carlo methods and temporal difference (TD) learning methods. The applications of dynamic programming methods are limited. The reason for this is that dynamic programming methods do require a model. The other methods on the other hand do not require a model. In this section, a short introduction will be given about discrete reinforcement learning methods.

3.1.4.1 Dynamic Programming

Dynamic programming (DP) methods do require a model. This means that the probabilities and rewards for all state transitions should be known. A Markov Decision Process (MDP) is assumed. A MDP satisfies the Markov Property which means that the future states depend on the present state only [22].

First, the state value function V_π for the current policy π should be calculated. The value function given in equation 3.1 can be rewritten in the following way:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma V_\pi(s') \Big] \tag{3.3}$$

In the equation above, s' means the transition state, $\pi(a|s)$ means the probability to take action a in s under policy π. The initial value function $v_0(s)$ can be chosen arbitrary and the successive approximations are calculated by the following equation:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma V_k(s') \Big] \tag{3.4}$$

If k goes to infinity, v_k will converge to v_π. A pseudo code to evaluate the value function is also given by Sutton and Barto:

Algorithm 1: A model-based method to approximate the value function

Initialize π, the policy to be evaluated:
Initialize an array $V(s) = 0$, for all $s \in S$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in S$
    $v \leftarrow V(s)$
    $V(s) \leftarrow V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma V_\pi(s') \Big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

The next step is to improve the policy. The goal is to find a better policy than the existing one. For a state s, we would like to know if an action $a \neq \pi(s)$ is better or not than the existing policy. We should look at the Q-value to know this. The Q-value in equation 3.2 can be rewritten to:

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big] \tag{3.5}$$

Let π and π' be a pair of two deterministic policies such that for all $s \in S$, $q_\pi(s, \pi'(s)) \leq v_\pi(s)$, than we know for sure that $v_{\pi'}(s) \leq v_\pi(s)$. This is known as the policy improvement theorem. If $q_\pi(s, a) \leq v_\pi(s)$, then the changed policy is better than the old policy π. The new greedy policy is found by choosing the action such that:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] \tag{3.6}$$

A pseudo-code to find the best policy will be:

Algorithm 2: A model-based method to approximate the value function

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$
2. Policy evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in S$
   $v \leftarrow V(s)$
   $V(s) \leftarrow V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma V_\pi(s')\Big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)
3. Policy Improvement
   $policy - stable \leftarrow true$
   For each $s \in S$
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big]$
   If $a \neq \pi(s)$, then $policy - stable \leftarrow false$
   If $policy - stable$, then stop and return V and π; else go to 2

3.1.4.2 Monte Carlo

In the case of Monte Carlo (MC) methods, no information about the environment is required initially. The goal is to estimate the value function by running multiple runs. If a high number of runs are performed, the estimated value function will converge to the real value function.

At every run, state s may be visited multiple times. For this reason, a distinction has to be made between the First-Visit MC method and the Multiple-Visit MC method. The First-Visit MC method estimates $V_\pi(s)$ by taking the average of all returns considering the first visit to s in a run only. To optimize a problem, the first step is to learn the value function. Sutton and Barto described the following pseudo-code to do this by using the first-visit MC method:

Algorithm 3: First-visit MC method to estimate the value function

Initialize: $\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
$Returns(s)$ an empty list, for all $s \in S$
Repeat forever:
Generate an episode using π
For each state s appearing in the episode:
$G \leftarrow$ return following the first occurrence of s
Append G to $Returns(s)$
$V(s) \leftarrow$ average($Returns(s)$)

If no model exists, the Q-value should be used to find the optimal policy π_*. The policy is evaluated and improved alternately. Policy evaluation is the estimation of the value function. Policy improvement is accomplished by making the policy greedy with respect to the current known value function. The policy will converge to the optimal policy π^* while the the value function also converges to the optimal value function $v_*(s)$.

After each episode, the observed returns should be used for policy evaluation. The policy is improved at all observed states. An algorithm doing this is the Monte Carlo with Exploring Starts (Monte Carlo ES) is described by Sutton and Barto:

Algorithm 4: The Monte Carlo ES algorithm as described by Sutton and Barto

Initialize, for all $s \in S$, $a \in A(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list
Repeat forever:
  Choose $s_0 \in S$ and $a_0 \in A(s_0)$ s.t. all pairs have probability > 0
  Generate an episode starting from s_0, a_0, following π
  For each pair s, a appearing in the episode:
    $G \leftarrow$ return following the first occurrence of s, a
    Append G to $Returns(s,a)$
    $Q(s,a) \leftarrow average(Returns(s,a))$
  For each s in the episode:
    $\pi(s) \leftarrow arg\max_a Q(s,a)$

Multiple other Monte Carlo method variations exist. These variations will not be described in this report.

3.1.4.3 On-policy and Off-policy methods

A distinction between off-policy and on-policy learning methods can be made. On-policy learning methods are improving and evaluating the same policy as the policy used to make decisions. Off-policy learning methods are making use of two different policies. This policies are the target policy π and the behavior policy μ. The behavior policy is the one that is used to make choices. This policy is usually stochastic, for example an ϵ-greedy policy. This is to ensure exploration. The target policy π converges to the optimal policy and is a deterministic greedy policy in many cases. The Monte Carlo ES algorithm is an example of an on-policy method [22].

3.1.5. Temporal Difference Learning

Monte Carlo methods update the value function or Q-value at the end of an episode. It is also possible to make an estimation after every time step while the episode is running. The reinforcement learning methods that are doing this are called Temporal Difference methods (TD methods). TD methods are making use of an existing estimate for the value function or Q-value to update it. This is called bootstrapping. A example of an first-visit MC update rule for a non-stationary environment is:

$$V(s_t) \leftarrow V(s_t) + \alpha\big[G_t - V(s_t)\big] \tag{3.7}$$

This is an example of a constant-α MC method. In this equation, G_t is the actual return following time t. G_t is not known until the end of the episode. G_t is known at the target. The letter α stands for a learning rate. The learning rate is a number between 0 and 1. Increasing the learning rate makes the more recently learned values for G_t more important compared with the older values.

The value function has a recursive property:

$$\begin{aligned} V_\pi(s) &= E_\pi\left[G_t \mid s_t = s\right] \\ &= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right] \\ &= E_\pi\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,\middle|\, s_t = s\right] \\ &= E_\pi\left[r_{t+1} + \gamma V_\pi(s_{t+1}) \,\middle|\, s_t = s\right] \end{aligned} \tag{3.8}$$

An update rule for a simple TD method is:

$$V(s_t) \leftarrow V(s_t) + \alpha\big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big] \tag{3.9}$$

In this case, G_t is estimated to be equal to $r_{t+1} + \gamma V(s_{t+1})$ which is the target in this case. The return is now estimated after every time step instead of after every episode. For this reason, the value function could also

be updated after every time step.

A pseudo-code to evaluate a policy by a TD(0) algorithm has the following form:

Algorithm 5: A pseudo-code used to evaluate a policy using a TD(0) algorithm.

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily
Repeat for each episode:
  Initialize s
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for s
    Take action A; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s) - V(s)\big]$
    $s \leftarrow s'$
  until s is terminal

It is not mathematically proven that TD methods converge faster than constant-α MC methods. However, in practice TD methods do converge faster.

Multiple TD learning algorithms have been developed in the past. Sarsa and Q-learning are two well known examples of TD methods. The former one is an on-policy method while Q-learning is an off-policy method [22].

3.1.6. Eligibility traces

In the previous section, the value function is updated after every time step. This methods are called TD(0) methods. There are also methods that are in between TD(0) methods and Monte Carlo methods. This methods do not wait until the end of an episode to update the value function, but are also not updating the value function every time step. This methods make use of eligibility traces [22].

TD algorithms are using $r_{t+1} + \gamma V(s_{t+1})$ as target. It is also possible to update the value function every time step. The target will be $r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2})$. This idea could be expanded to update the value function every n steps. This is called a n-step backup. It is also possible to define a target that is calculated by weighting multiple TD targets, for example by taking a weighted average of a two-step backup and a three-step backup. Eligibility traces are used to estimate the return by weighting multiple TD targets in an episode. This weighting is done for a one-step backup until a n-step backup. This weights add up to 1. Examples of algorithms that make use of eligibility traces are TD(λ), Sarsa(λ) and Watkin's Q(λ). The Greek letter λ is a number between 0 and 1. If λ is set to 0, the target is calculated each time step. If λ is set to 1, the target is calculated only at the end of every episode and becomes a MC method.

3.1.7. Continuous reinforcement learning

The reinforcement learning methods explained are suitable for problems with discrete state-spaces and action-spaces. The state space may also be continuous or discrete but large. The reinforcement learning algorithm will not be able to visit every possible state a multiple number of times in a reasonable amount of time, especially if the problem is high dimensional because of the curse of dimensionality. In that case, the algorithm has to estimate the value function or Q-value [22].

Multiple methods exist to estimate the value function or Q-value. Examples are gradient descent methods, coarse coding, tile coding and multivariate splines. One of the most common gradient-descent methods to estimate a function is by making use of neural networks.

3.1.7.1 Neural networks

An artificial neural network could be described as an interconnected network of nodes and are used to map a set of inputs to a set of outputs [10]. Neural networks are constructed from multiple layers. The first layer

is called the input layer. The number of nodes in the first layer is the same as the number of inputs. The last layer is the outputs layer and the number of nodes in this layer is the number of outputs. There are also hidden layers between the input and output layer. An example of a neural network topology is shown in figure 3.2.

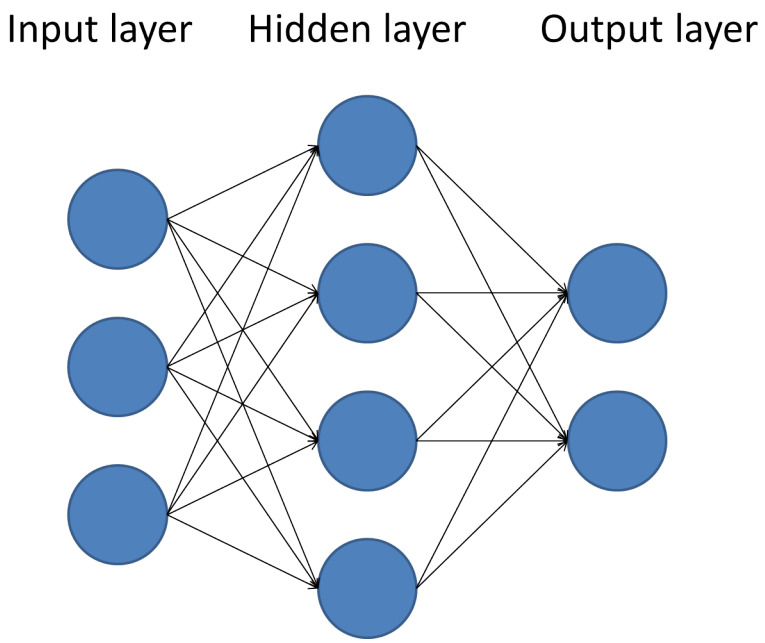


Figure 3.2: An example of a neural network with an inputs layer, a hidden layer and an output layer. There are three inputs and two outputs.

Consider the case where there are D inputs. The inputs are $x_0, x_1, ..., x_D$. It is assumed that the first hidden layer has M nodes. M linear combinations of the input variables can now be constructed:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{3.10}$$

The superscript (1) is indicating that the parameters are in the first layer of the neural network, $w_{ji}^{(1)}$ are the weights of the network, $w_{j0}^{(1)}$ are the biases and the numbers a_j are known as the activations. These activations are mapped to a number between 0 and 1 by a differentiable nonlinear activation function $h(a)$. Examples of possible activation function are the sigmoid and the 'tanh' function. This mapping gives:

$$z_j = h(a_j) \tag{3.11}$$

Now consider the case that the next layer has M nodes. Now z_j may be linearly combined to get a_k:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{3.12}$$

If we assume that there are just three layers, a_k is mapped to y_k. This mapping depends on the nature of the problem. For standard regression problems, y_k is just equal to a_k. If the output should be binary, then a_k may be mapped to y_k by using the standard sigmoid function $\sigma(a)$. This function has the following form:

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{3.13}$$

It is also possible to define an additional input variable $x_0 = 1$. In that case equation 3.10 will have the following form:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{3.14}$$

If we also define $z_0 = 1$, also the form of equation 3.12 will change:

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{3.15}$$

The neural network structure shown in figure 3.2 is called a Deep Feed Forward (DFF) neural network. This type is one of the most common one but there are also other types of neural networks.
Neural networks are used to approximate a function. Any function may be approximated by summing up sigmoids. The weights and biases may be determined by a technique called backpropagation.

The goal of backpropagation is to find and minimize an error function $E(\mathbf{w})$. The first step is to derive the error function with respect to the weights. The second step is to make a step toward the local minimum. This is the downside of backpropagation, the backpropagation algorithm will not always find a global optimum.

Consider N data points to train the neural network. Each data point will contribute to the total error function:

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{3.16}$$

For every training input, there is also a target output. There is also an output from the neural network. The errors are summed up quadratically to calculate E_n and has usually the following form:

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{3.17}$$

, where the neural network output $y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$. The target output is t_{nk}. If the activation functions are differentiable, the gradient of $E(\mathbf{w})$ with respect to $\mathbf{w}$ can be calculated. If the gradient $\nabla E(\mathbf{w})$ is calculated, we can now move to a minimum by using the following equation:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{3.18}$$

This method of finding the minimum is called gradient descent. The choice could also be made to update the weight based on one data point only:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}) \tag{3.19}$$

This is called on-line gradient descent, sequential gradient descent or stochastic gradient descent. The letter η stands for a learning rate and should always be a positive number.

3.1.7.2 Policy-based learning

Some reinforcement learning methods are estimating value functions. These methods are called value-based methods. If the value function is known, a global optimal solution may be found by choosing greedy actions. Error propagation could be a problem in value-based learning. A small change in policy, could lead to a large change in the value-function, which may cause again a large change in policy. For this reason, value-based learning methods may be unstable if the policy is approximated. This may cause dangerous situations in real systems [11].

Policy-based methods on the other hand, are attempting to map observations directly to an action. This methods are usually only considering the current policy and it neighborhood. Policy-based methods are for this reason more likely to stick into a local optimum. This methods however tend to work better for continuous problems [11]. Policy-gradient methods are also more suitable if the optimal policy is stochastic. Value-based methods are only able to find deterministic policies. Policy-based methods also have better convergence properties [12].

Two formulations exist to measure the performance of a policy [12]. The first one is the average reward formulation in which the policy is rated according to the average long-term expected reward per time step:

$$\rho(\theta) = \lim_{n \to \infty} E[r_1 + r_2 + ... + r_n | \pi] = \sum_s d^{\pi}(s) \sum_a \pi(s, a, \theta) \mathscr{R}_s^a \tag{3.20}$$

In this formulation $\pi(s,a,\theta) = Pr(a_t = a|s_t = s,\theta)$ and $\mathscr{R}_s^a = E[r_{t+1}|s_t = s, a_t = a]$. The letter θ is used as a parameter vector to parameterize the policy. The variable $d^\pi(s)$ is the stationary distribution and is defined as $\lim_{t\to\infty} Pr(s_t = s|s_0,\pi)$. If the formulation in equation 3.20 is used, the value of a state-action is defined as:

$$Q^\pi(s,a) = \sum_{t=1}^{\infty} E[r_t - \rho(\theta)|s_0 = s, a_0 = a, \pi] \tag{3.21}$$

If a designated start state exists, the following formulation for $\rho(\theta)$ is used:

$$\rho(\theta) = E\Big[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \Big| s_0, \pi\Big] \tag{3.22}$$

If this formulation is used, $Q^\pi(s,a)$ is defined as:

$$Q^\pi(s,a) = E\Big[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \Big| s_t = s, a_t = a, \pi\Big] \tag{3.23}$$

The variable $d^\pi(s)$ is now a discounted weighting and has a slightly different definition in this formulation: $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s|s_0,\pi)$.

For both formulations, the derivative of ρ with respect to θ will be:

$$\frac{\partial \rho}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s,a,\theta)}{\partial \theta} Q^\pi(s,a) \tag{3.24}$$

In the equation above $Q^\pi(s,a)$ is not known and must be estimated. The goal is to find a θ such that ρ is maximized. This is usually done by a technique called gradient ascend. The gradient of the policy objective function ρ is ascended by taking the gradient with respect to θ. Gradient descent was discussed earlier in this chapter to minimize an error function. In the case of gradient ascent, a maximum is found by updating the weights and taking the following update rule for θ with learning rate α:

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta \rho(\theta) \tag{3.25}$$

Different algorithms exist to estimate the gradient $\nabla_\theta \rho(\theta)$. The gradient $\nabla_\theta \rho(\theta)$ is defined as:

$$\nabla_\theta \rho(\theta) = \begin{pmatrix} \frac{\partial \rho(\theta)}{\theta_1} \\ \vdots \\ \frac{\partial \rho(\theta)}{\theta_n} \end{pmatrix} \tag{3.26}$$

3.1.7.3 Actors and Critics

A mapping from an observation to a value-function is called a critic, while a mapping from an observation to an action is called an actor. Methods called actor-critic methods, are using an actor and a critic to find an optimal policy [22].

An action is selected by the actor. The critic is used to evaluate the action to determine if the action is better or worse than expected. The evaluation is done by making use of the following TD error:

$$\delta_t = r_{t+1} + \gamma v_t(s_{t+1}) - v(S_t) \tag{3.27}$$

where v_t is the value function calculated by the critic at time t and is used to evaluate action a_t taken in state s_t. If the TD error is positive, the tendency to select the same action should grow.

There are multiple ways to select actions. A possible way to select an action is by making use of the Gibbs distribution:

$$\pi_t(a|s) = Pr(a_t = a|s_t = s) = \frac{e^{H_t(s,a)}}{\sum_b e^{H_t(s,b)}} \tag{3.28}$$

$H_t(s,a)$ are the values at time t of the policy parameters of the actor. The likelihood of an action to be chosen can be strengthened or weakened by updating $H_t(s,a)$. Several update rules are possible. Two examples mentioned by Sutton and Barto are:

$$H_{t+1}(s_t,a_t) = H_t(s_t,a_t) + \beta\delta_t \tag{3.29}$$

and

$$H_{t+1}(s_t,a_t) = H_t(s_t,a_t) + \beta\delta_t\Big[1-\pi_t(a_t|s_t)\Big] \tag{3.30}$$

where β is a positive step-size parameter.

3.2. Applications of Reinforcement learning in Space Guidance, Navigation and Control

3.2.1. Introduction

This section will introduce some applications of reinforcement learning in space guidance navigation and control. Two trajectory problems, the planetary landing and the rendezvous/docking problem will be worked out further in the coming chapters. Some of the papers discussed it this section, do also compare the result of a reinforcement learning controller with the performance of a conventional controller.

Reinforcement learning is not widely applied in space guidance, navigation and control today. However, many research is going on in this topic. Simulations are useful to do this. It is possible to use reinforcement learning for spacecraft attitude control, planetary landings, rendezvous/docking, satellite clustering and other applications. Actuators may fail and reinforcement learning could be used to train an agent to make the best out of such a situation. In this chapter, an overview in reinforcement learning applications in space guidance, navigation and control will be given.

3.2.2. Planetary landing

A successful planetary landing requires a navigation system to estimate the state and a control system. This control system should map a state to an action. The action is a command thrust in this particular case. The control system may be trained in a simulation by a reinforcement learning algorithm as is shown by Gaudet, Linares and Furfaro [1]. They do this by making use of the PPO algorithm [4]. The problem has six degrees of freedom. The goal is to land the planetary lander while the fuel consumption is minimized. Four thrusters were used in a thruster model. This paper is worked out in section 3.3.

In 2020 Scorsoglio, D'Ambrosio, Ghilardi, Furfaro, Gaudet, Linares and Curti published a paper about autonomous hazard avoidance and landing [24]. A supervised learning approach is used to train the agent to recognize unsafe landing spots.

The training set is composed of 18000 images with a resolution of 64×64 pixels. Unsafe landing spots are labeled by black pixel while safe landing spots are labeled by white pixels. A pixel is unsafe if the local slope is less than 8 degrees, the roughness is more than 5% or if the area is in shadow. A median filter is then used to reduce the number of unsafe pixels. For every pixel, the distance to the closest unsafe pixel is calculated. The pixel with the largest distance to an unsafe pixel will be the intended landing spot. A camera is then centered to this spot to prefer the target pixel to change position.

The PPO method is used to train the GNC policy. Three degrees of freedom are considered, translation only. The equations of motion are:

$$\ddot{\mathbf{r}} = \mathbf{g} + \frac{\mathbf{T}}{m} \tag{3.31}$$

$$\dot{m} = -\frac{||\mathbf{T}||}{I_{sp}g_0} \tag{3.32}$$

where $\mathbf{r} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}$ is the position vector, $\mathbf{T} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}$ is the thrust vector, m is the mass, $g = \begin{bmatrix} 0 & 0 & -1.62 \end{bmatrix} m/s^2$ is the Moon gravity vector, I_{sp} is the specific impulse and g_0 is 9.81 m/s^2.

The navigation sensors are a radar altimeter and optical sensors to make images of the ground. Two networks are used to train the GNC policy, an actor network and a critic network. The networks have different observation spaces. The observation space of the actor networks is:

$$obs_{\pi_\theta} = \begin{bmatrix} r_z & \dot{r}_z & LP_x & LP_y \end{bmatrix} \tag{3.33}$$

where r_z is the vertical position and LP_x and LP_y are the coordinates of the landing pixel. The landing pixel is selected randomly to train the GNC policy. The observation space of the critic network is:

$$obs_{VF} = \begin{bmatrix} \mathbf{r} & \dot{\mathbf{r}} & LP_x & LP_y \end{bmatrix} \tag{3.34}$$

The target point is 200 meters above the ground. Under this attitude, the camera is not able to distinguish ground features. The initial position and velocity are uniformly distributed. The reward function is defined such that a difference between the velocity and a target velocity is minimized and that the distance between the projection of the landing spot and the center of the central pixel are minimized.

In 2022, a follow-up paper was written by the same authors [25]. Also in this problem, a radar altimeter and ground images were used the land a lander autonomously. The value of a pixel indicated the elevation of at that pixel. The GNC policy is learned by the PPO method. The dynamics is modeled in the same way as the previous paper. The goal is to land on a landing target but the coordinates of the lander are hidden. The observation space of the critic network is defined as:

$$obs_{VF} = \begin{bmatrix} \mathbf{V}_{err} & t_{go} & r_z \end{bmatrix} \tag{3.35}$$

where $\mathbf{V}_{err}$ is defined as the difference between the real velocity and target velocity, t_{go} as the time-to-go and r_z as the altitude. The observations available to the actor are:

$$obs_{\pi_\theta} = \begin{bmatrix} \mathbf{I} & r_z & \dot{r}_z \end{bmatrix} \tag{3.36}$$

where $\mathbf{I}$ is a 16×16 grayscale image. The reward function minimizes $\mathbf{V}_err$ and the control effort. A bonus is given if the lander makes it to the target position with a tolerable position error and if the speed is within a defined speed limit.

The gravitational acceleration and mass are random variables chosen from a uniform distribution. The same is true for the initial position and velocity. The target position is chosen to be 1000 meters above the ground for the same reason as mentioned before. Actuator failure is simulated by reducing the thrust in one of the three directions randomly. The authors conclude that the performance is fairly good but not good enough for a real landing.

In 2022 a paper about the use of a seeker in a guidance and control problem in the context of a lunar landing was published by Gaudet and Furfaro [26]. This seeker has a LIDAR sensor and a camera in it. This seeker is directed to a landing point.

The origin of the coordinate system is chosen as the landing spot. During the powered descent, a new landing spot is designated and the lander has to divert. The initial position, velocity and attitude are chosen randomly. There is also a random perturbation in the inertia matrix.

Four thrusters are used to land the spacecraft. At the start of every session, the thrust of one actuator is reduced to model failure. Actuator delay is modeled by defining an actuator time constant τ_{ctrl}. The seeker attached to the lander has its own dynamics. The center of mass of the lander is also shifting while fuel is burnt. The dynamics of this seeker are not worked out in this report because it is not relevant for this thesis. The details about the seeker dynamics can be found in the paper.

The PPO method is used to train the lunar lander. Two different policies are trained, a guidance segment and a landing segment. The reward functions are slightly different. The authors are claiming that this leads to

the best performance. The lunar lander reward function is designed to minimize the difference between the velocity and reference velocity and to minimize the control effort. A bonus reward is given for a successful landing and a penalty for exceeding a roll or pitch limit. The episode is also terminated in that case. The observables of the actor and critic are:

$$obs_{VF} = \begin{bmatrix} r_z & \mathbf{v} & \mathbf{q} & \omega \end{bmatrix} \tag{3.37}$$

$$obs_{\pi_\theta} = \begin{bmatrix} h & \dot{h} & \mathbf{q} & \omega \end{bmatrix} \tag{3.38}$$

where r_z stands for the position of the lander relative to the designated landing point, $\mathbf{v}$ for the velocity, $\mathbf{q}$ for the attitude expressed by quaternions, ω for the angular velocity and h for the altitude measured by the seeker.

The evaluation of the optimized policy shows that 98% of the landings are safe. The 2% of the episodes that did not meet the conditions of a safe landing where close however. The controller is also able to cope with conditions not modeled during training like actuator degradation and a variation in the mass and inertia.

In 2021, Ciabatti, Daftry and Capobianco published a paper about deep reinforcement learning and transfer learning in the context of a planetary landing [27]. An agent is trained to land a spacecraft on Mars and the Moon by the Deep Deterministic Policy Gradient (DDPG) algorithm. Terrain models made by NASA and the OpenAI gym interface were used. The Bullet/PyBullet engine was used as physics engine.

The lander has four propulsive forces as input. Two types of sensors are used to observe the environment, RBG camera's and LIDAR sensors. The input is an observation vector with 262251 observables. The input are a 256×256 image with four RGB channels including the alpha channel, a three dimensional vector four the position and a four dimensional quaternion vector.

Mars and Moon terrain mesh models are published my NASA on Github. Two Moon terrains are used to train the agent. The first terrain is the near side of the Moon that is facing to the Earth. This side has a relatively smooth terrain. Also the far side of the Moon is used to train the agent. This side has more craters and is more rough. The performance of the controller is tested on Mars. The Victoria Crater is chosen as terrain to land on. This crater is rough similar to the Moon's far side.

The researchers also designed a model of the Polar Lake district on Titan, one of Saturn's moons. The policy learned on the Moon was transferred and used to land the spacecraft on Mars and Titan. The initial state is chosen randomly with the exception of the altitude. The lander has to land on four legs. The reinforcement learning controller is able to land the spacecraft with four legs on Mars and Titan.

3.2.3. Docking

The PPO algorithm is also used by Oestreich, Linares and Gondhalekar [5]. They use this algorithm to learn an agent to dock a spacecraft. This problem is close to the planetary landing problem. They want to minimize the control cost and the conditions for a successful landing is similar. No thruster model is used. The model has six inputs, three forces in the three translational directions and three torques in the three rotational directions. The problem is not solved in the relative orbit reference frame.

Broida and Linares attempt to solve a three degrees of freedom docking problem by making use of the PPO algorithm [17]. They solve this problem in the relative orbit reference frame but only take translational motion into consideration which reduces the problem to three degrees of freedom. They use the PPO algorithm to solve the problem. The inputs are the forces in the three translational directions. This paper will be worked out in more detail in section 3.4.
An interesting option is Run Time Assured (RTA) Reinforcement learning. The idea is to train an agent by a reinforcement learning and to make use of a reinforcement learning policy if the agent is in a save situation. If the situation becomes unsafe, the agent switches to a backup controller and is penalized. Dunlap, Mote, Delsing and Hobbs researched RTA reinforcement learning in satellite docking [23].

3.2.4. Satellite Attitude Control

Reinforcement learning may also be used to control the attitude of an aerospace vehicle. Delft Engineers designed a reinforcement learning control system for example in 1998 [9]. The authors noted that the attitude controller of a satellite is often characterized by a limit cycle due to measurement inaccuracies and sensor noise. The engineers hoped to reduce the limit cycle by implying the reinforcement learning controller.

The reinforcement learning problem was continuous. A critic network was used to estimate the value function. The inputs of this network were a filtered attitude error and the attitude error rate. This networks was a Takagi–Sugeno (T-S) fuzzy-network.

Also a T-S fuzzy controller was used. This controllers maps the attitude error and the attitude error rate to an action and could be considered to be an actor. The fuzzy controller was initialized as a PD controller. The temporal difference was used as an operator to reward the algorithm. If the value function of an action is better than the predicted value function, a reward was given. If it was worse, a penalty was given. This way, the controllers learns what the best actions are. From figure 3.3, it could be concluded that the learned policy performs better than the initial linear PD controller. This is not a surprise however because of the fact that a PID controller is not an optimal controller. The PID controller was an initial policy and the learned policy could only become better than the initial policy.

3.2.5. Hovering

Gaudet and Furfaro used reinforcement learning to learn an agent to hover above an asteroid with partially unknown dynamics. The rotation and gravity above this asteroid was even non-uniform. The authors intended to create a controller that is able to cope with uncertain environments and is also robust. With robust they mean that the steady-state error is small, the transient-state is living short and the overshoot is small. The problem only considers translation and is a three degrees of freedom problem.

There are multiple uncertainties about asteroids from a distance. The surface properties, gravity, shape and rotational velocity are all unknown. For this reason, asteroids have to be observed from a close distance before a landing maneuver could be planned. The authors designed a reinforcement learning agent that is able to fly to a specified location in the body-centered rotating reference frame of the asteroid, with the use of an optical navigation system. The spacecraft is only limited by its thrust capabilities.

The authors do mention multiple applications of robust and accurate hovering. The first one is drilling while the controller ensures that the opposing force as a result of drilling does not push the spacecraft off the surface. The second application is a executing a soft landing on an asteroid. The gravity on an asteroid is very weak. The consequence of this is that a spacecraft may bounce away from the asteroid surface if the landing is to hard. Another application is the deployment of a sky-crane. The last mentioned application is landing on unstable terrains.

It is decided to model sensor noise, actuator noise and sensor delay in the problem. This noise has a Gaussian nature. Also environmental noise is modeled to include solar radiation and other types of noise. This environmental noise is also Gaussian.

The chosen algorithm is direct policy optimization. This means that only an actor network is used. In this problem, three neural networks were used, one for each translational direction. The reward function is negative quadratic with respect to the desired position:

$$R = -||\mathbf{p} - \mathbf{p}_0|| \tag{3.39}$$

The policy is tested on 1000 random generated ellipsoid asteroids. The worst steady-state error is a deviation of 34 centimeter.

The reinforcement learning controller is also compared to a linear PD controller optimized by reinforcement learning. The PD controller has a larger overshoot and a longer transient state. The steady state error is also higher. The non-linear reinforcement learning controller is better in all of these aspect.

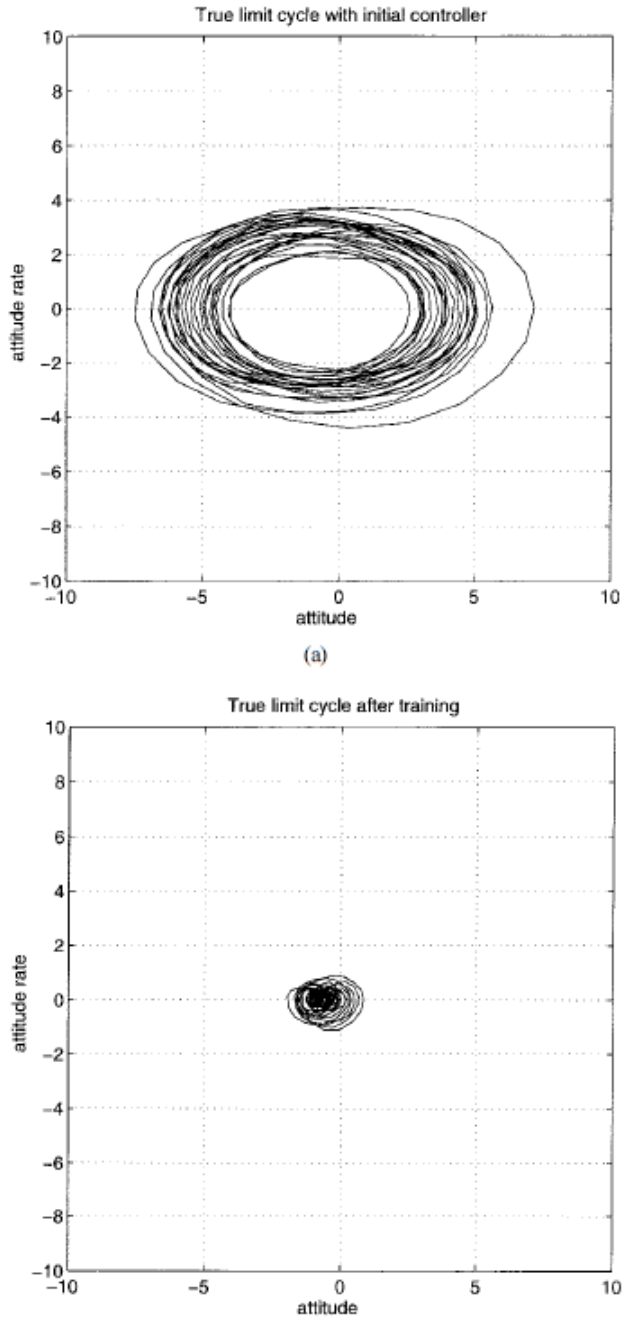


Figure 3.3: The limit cycle of the initial PD policy and the limit cycle after reinforcement learning[9]

The non-linear reinforcement learning controller is also compared with a LQR controller. The steady-state errors were similar but the transient response of the reinforcement learning controller had less oscillations. The explanation for this is that the system matrix used to design the LQR controller had no information about the environmental noise. The reinforcement learning controller on the other hand is trained in a noisy environment, making it robust to unmodeled dynamics.

Gaudet, Furfaro and Linares wrote a follow up paper where the same problem is solved with six degrees of freedom [20]. The spacecraft is modeled as a cube. The chosen algorithm to train the reinforcement learning agent is PPO [4]. The reinforcement learning controller is made adaptive by meta reinforcement learning. The agent is trained on different asteroid shapes and environmental dynamics. For each episode, the agent

is trained on a random shape, density, rotational speed, and nutation angle. Actuator noise and sensor noise is modeled again in the problem. The controller is trained to cope with failure by reducing the thrust of a random thruster by 90% with a probability of 50%. The controller is able to generalize to new situations but performed slightly worse on new situations compared with the situations used for training.

3.3. The planetary landing problem

3.3.1. Introduction

The planetary landing problem will be explained in more detail in this chapter. A paper of Gaudet, Linares and Furfaro is used as reference [1]. In this chapter, the question how the planetary landing could be modeled will be answered. At the end of the chapter, the fuel consumption of the reinforcement learning controller is compared with a fuel optimal controller. The problem is solved with six degrees of freedom.

3.3.2. PPO algorithm

3.3.2.1 Equations of motion

Gaudet, Linares and Furfaro trained a spacecraft to land on Mars in a simulation [1]. The force $\mathbf{F}^B$ and torque $\mathbf{L}^B$ in the lander body frame depend on the place of the thrusters. The placement of the vector can be described by direction vector $\mathbf{d}$ and position vector $\mathbf{r} \in \mathbb{R}^3$. If it is assumed that the lander has n mounted thrusters, the force and torque in the body frame are:

$$\mathbf{F}^B = \sum_{i=1}^{n} d_i T_i \tag{3.40}$$

$$\mathbf{L}^B = \sum_{i=1}^{n} \mathbf{r}_i \times \mathbf{F}_i^B \tag{3.41}$$

$T_i \in [T_{min}, T_{max}]$ is the commanded thrust for thruster i, T_{min} is the minimum thrust and T_{max} is the maximum thrust, $\mathbf{d}_i$ is the direction vector for thruster i and $\mathbf{r}_i$ is the position vector of thruster i. The force in the body frame is converted to the force in the inertial frame by the following equation:

$$\mathbf{F}^N = A_B^N(\mathbf{q})^T \mathbf{F}^B \tag{3.42}$$

$A_B^N(\mathbf{q})$ is the direction cosine matrix and $\mathbf{q}$ is the attitude expressed as a quaternion divided in the vector component $\boldsymbol{\varrho}$ and scalar component q_4 such that $\mathbf{q}^T = [\boldsymbol{\varrho} \quad q_4]$. $A_B^N(\mathbf{q})$ is related to $\mathbf{q}$ by:

$$A_B^N(\mathbf{q}) = \Xi^T(\mathbf{q})\Psi(\mathbf{q}) \tag{3.43}$$

$\Xi(\mathbf{q})$ and $\Psi^T(\mathbf{q})$ are defined as:

$$\Xi(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times 3} + [\boldsymbol{\varrho}\times] \\ -\boldsymbol{\varrho}^T \end{bmatrix} \tag{3.44}$$

$$\Psi(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times 3} - [\boldsymbol{\varrho}\times] \\ -\boldsymbol{\varrho}^T \end{bmatrix} \tag{3.45}$$

The definition of $[\mathbf{a}\times]$ is:

$$[\mathbf{a}\times] \equiv \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{3.46}$$

The rotational velocities $\boldsymbol{\omega}_{B/N}$ are determined by integrating the Euler rotational equations of motion:

$$\mathbf{J}\dot{\boldsymbol{\omega}}_{B/N} = -[\boldsymbol{\omega}_{B/N}\times]\mathbf{J}\boldsymbol{\omega}_{B/N} + \mathbf{L}^B + \mathbf{L}_{env}^B \tag{3.47}$$

$\mathbf{L}_{env}^B$ is the external body frame torque from external disturbances form the environment and J is the inertia matrix. The attitude of the lander is obtained by integration:

$$\dot{\mathbf{q}} = \frac{1}{2}\Xi(\mathbf{q})\dot{\boldsymbol{\omega}}_{B/N} \tag{3.48}$$

The translation is modeled by using the following equations:

$$\dot{\mathbf{r}} = \mathbf{v} \tag{3.49}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{F}^N + \mathbf{F}^N_{env}}{m} + \mathbf{g} \tag{3.50}$$

$$\dot{m} = -\frac{\sum_{i=1}^{n} ||\mathbf{F}_i^B||}{I_{sp} g_{ref}} \tag{3.51}$$

A value of 9.8 m/s^2 was used for g_{ref}, $\mathbf{g} = \begin{bmatrix} 0 & 0 & -3.7114 \end{bmatrix}$ m/s^2 and $I_{sp} = 225$ s. F^N_{env} is a normally distributed random variable. The moment of inertia of the lander is modeled as an ellipsoid with a uniform density:

$$\mathbf{J} = \frac{m}{5} \begin{bmatrix} b^2 + c^2 & 0 & 0 \\ 0 & a^2 + c^2 & 0 \\ 0 & 0 & a^2 + b^2 \end{bmatrix} \tag{3.52}$$

The half lengths of the principal axes in the x, y and z direction are given by a, b and c and the mass is given by m.

3.3.2.2 Constraints

The reinforcement learning problem is defined as [1]:

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: 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 summation is over a trajectory and $\mathbf{T}$ is the total thrust.

Subject to:

1. Terminal glideslope: $\frac{||v_z||}{||v_{x,y}||} > 5$ over the final 2 m of the descent (soft constraint)
2. Attitude constraints: Pitch ϕ and roll θ are less than 80 degrees (hard constraint)
3. Equations of motion

Soft constraints and hard constraints are both penalized by a negative reward, but the episode is only terminated after breaking a hard constraint.

3.3.2.3 Training

The problem is solved by using the PPO algorithm. Two separate actor and critic networks are used. Three hidden layers were used for each network.

During the agent's training, the initial conditions were chosen randomly by an uniform distribution. The initial conditions are given in table 3.1 [1]:

Table 3.1: The initial conditions to train the agent were chosen from an uniform distribution [1].

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

A target velocity was defined to speed up the learning process. This target velocity was used to define the shaping reward:

$$\mathbf{v}_{targ} = -v_0\left(\frac{\hat{\mathbf{r}}}{||\hat{\mathbf{r}}||}\right)\left(1-\exp\left(-\frac{t_{go}}{\tau}\right)\right) \tag{3.53}$$

$$t_{go} = \frac{||\hat{\mathbf{r}}||}{||\hat{\mathbf{v}}||} \tag{3.54}$$

$$\hat{\mathbf{r}} = \begin{cases} \mathbf{r} - \begin{bmatrix} 0 & 0 & 15 \end{bmatrix} & \text{if } r_z > 15 \\ \begin{bmatrix} 0 & 0 & r_z \end{bmatrix} & \text{otherwise} \end{cases} \tag{3.55}$$

$$\hat{\mathbf{v}} = \begin{cases} \mathbf{v} - \begin{bmatrix} 0 & 0 & -2 \end{bmatrix} & \text{if } r_z > 15 \\ \mathbf{v} - \begin{bmatrix} 0 & 0 & -1 \end{bmatrix} & \text{otherwise} \end{cases} \tag{3.56}$$

$$\tau = \begin{cases} \tau_1 & \text{if } r_z > 15 \\ \tau_2 & \text{otherwise} \end{cases} \tag{3.57}$$

The value for $\tau_1 = 20$ and $\tau_2 = 100$. A reward function is defined. A reward bonus is given if the lander lands with the right conditions. The reward function is [1]:

$$\begin{aligned} r = \alpha||\mathbf{v}-\mathbf{v}_{targ}|| + \beta||\mathbf{F}_B|| + \gamma \text{any}(\mathbf{q}(t) > \mathbf{q}_{lim}) + \delta \sum_{i=1}^{3}\Big[-\max(0, \mathbf{q}_i - \mathbf{q}_{mgn_i})\Big] + \eta \\ +\kappa\Big(r_z < 0 \text{ and } ||\mathbf{r}|| < r_{lim} \text{ and } ||\mathbf{v}|| < v_{lim} \text{ and all } (\mathbf{q} < q_{lim}) \text{ and all } (\boldsymbol{\omega} < \omega_{lim} \text{ and } gs < gs_{lim})\Big) \end{aligned} \tag{3.58}$$

There are six parameters in equation 3.58. The first parameter α is used to penalize the agent for deviating from the target velocity. The second parameter β penalizes control effort. If yaw, pitch or roll limits are exceeded, this is penalized by the term γ. The parameter δ is used to penalize the agent if the attitude of the lander passes a set threshold $\mathbf{q}_{mgn_i}$ and is used to prevent that the lander further approaches the attitude limit. The episode is then also terminated. The constant term η encourages progress and is is used to prevent that the agent violates the attitude limits immediately just to stop getting more negative reward. The rewards mentioned before are all negative while η is positive. The last parameter κ is a win bonus given if the lander is able to land successfully. The values for this parameters chosen by the writers were $\alpha = -0.01$, $\beta = -0.05$, $\gamma = -100$, $\delta = -20$, $\eta = 0.01$ and $\kappa = 10$. The value for $\mathbf{q}_{lim} = \begin{bmatrix} 360 & 80 & 80 \end{bmatrix}$ degrees and $\mathbf{q}_{mgn_i} = \begin{bmatrix} 360 & 67 & 67 \end{bmatrix}$ degrees.

The landing is successful if the distance the distance to the goal is less than 5 m, the maximum landing speed is 2 m/s, the maximum allowed attitude angle except for yaw is 0.2 rad. The yaw is not limited at all. The limit

in the angular velocity 0.2 rad/s and the glideslope limit is 79 degrees. This means that the limits are $r_{lim} = 5$ m, $v_{lim} = 2$ m/s, $q_{lim} = 0.2$ rad, $gs_{lim} = 79$ degrees and $\omega_{lim} = 0.2$ rad/s. The observations made by the agent are:

$$\text{obs} = \begin{bmatrix} \mathbf{v}_{error} & \mathbf{q} & \boldsymbol{\omega} & r_z & t_{go} \end{bmatrix} \tag{3.59}$$

In this observation matrix, $\mathbf{v}_{error} = \mathbf{v} - \mathbf{v}_{targ}$, $\mathbf{q}$ and $\boldsymbol{\omega}$ are an estimation of the attitude and angular velocity of the spacecraft. The translation coordinates r_x and r_y are not observed by the agent. This is done to find a policy that is able extend it behavior to areas that are not experienced while learning. Uncertainties are modeled in the observations to make the found policy more robust.

3.3.2.4 Results

300,000 episodes were run to train the agent. The mean reward, minimum reward and standard deviation for the reward were calculated per 120 episodes. The policy and value function were also updated after 120 episodes. This resulted to the graph in figure 3.5.

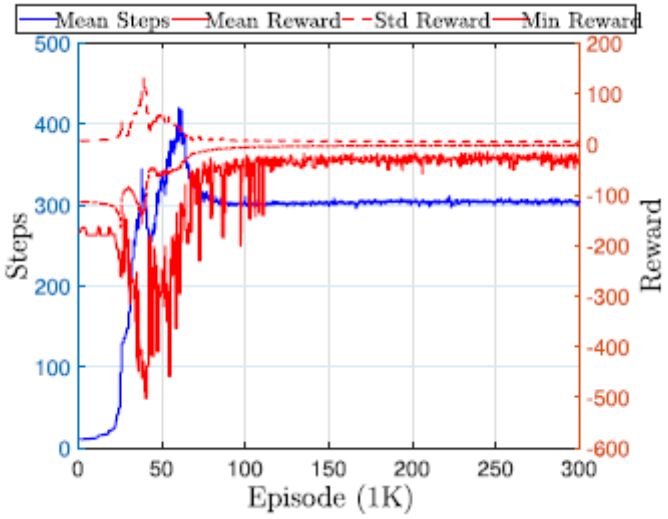


Figure 3.4: The policy is converging to an optimum [1]. The mean reward, minimum reward and standard deviation for the reward were calculated per 120 episodes. The mean number of steps were calculated per 1000 episodes.

Steps is referring to the number of time steps per episode. The integration period is 0.2 seconds. The authors made a similar graph for a simplified but similar problem with three degrees of freedom and is shown in figure 3.5.

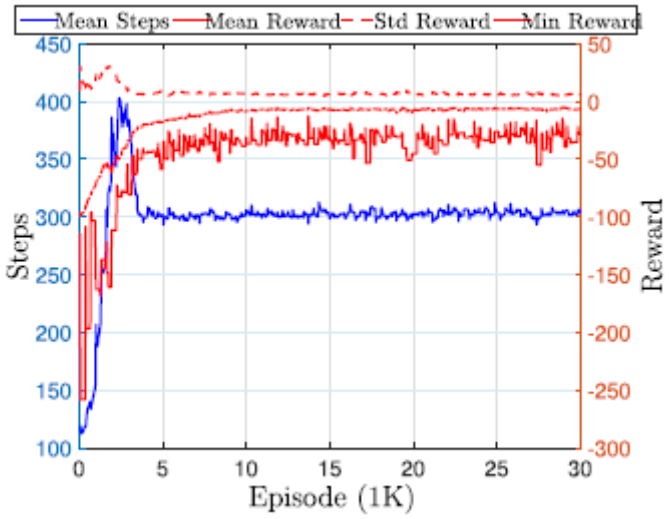


Figure 3.5: The mean reward, minimum reward and standard deviation for the reward were calculated per 120 episodes for a three degrees of freedom mars lander. The mean number of steps were calculated per 1000 episodes also [1].

To test the policy, 10,000 episodes were run. Gaussian noise was acting on the lander during this test with a standard deviation of 100 N. This noise had a mean between -100 N and 100 N. This mean was chosen by an uniform distribution. This mean was kept constant during an episode. Also the initial wet mass and gravity were chosen randomly. The results for the touch down are shown in table 3.2.

Table 3.2: The touchdown statistics. The glideslope angle is an average over the last 2 m of the trajectory [1].

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

DR/DV refers to an energy-optimal guidance algorithm used by the authors to compare the reinforcement learning result with [14] [15], although this algorithm assumes an unlimited thrust and leads to an unacceptable high glideslope.

The authors also compared their reinforcement learning results with the solution of the GPOPS optimal controller for a three degrees of freedom problem [16]. From this comparison it can be concluded that the reinforcement learning problem is not fuel optimal. They are arguing that the reinforcement learning learned policy may do better if the reward function is modified.

3.3.3. DDPG, TD3 and SAC

3.3.3.1 Equations of motion

Xu, Chen and Bai did something similar but they used the DDPG, TD3 and SAC algorithm [28]. Six thrusters were used and are deployed in the shape of a regular hexagon. This is shown in figure 3.12 [28]. An angle ϕ is also defined. This is the angle between the thruster and the vector perpendicular to the hexagon.

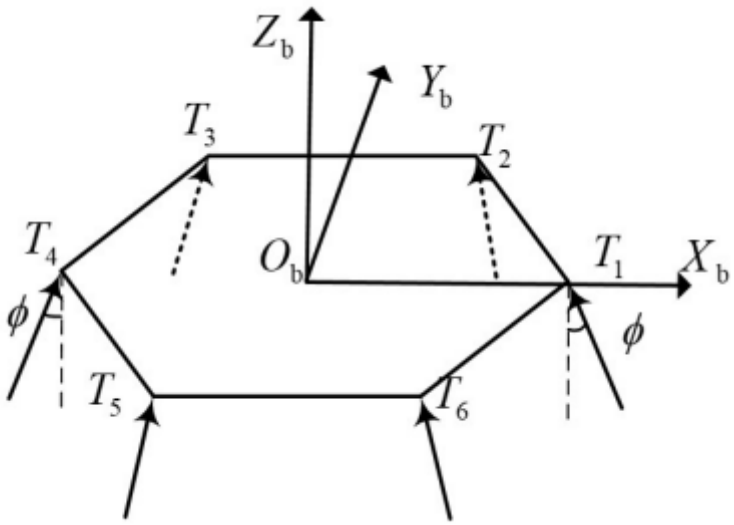


Figure 3.6: The thrusters are deployed in the shape of a regular hexagon. The angle ϕ is the angle between the thruster and the vector perpendicular to the hexagon [28].

The thrust of each engine is limited by a minimum and a maximum:

$$T_{min} \leq T_i \leq T_{max} \tag{3.60}$$

Lets also define a thrust vector $\mathbf{T} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \end{bmatrix}$. From geometry a force and moment on the body frame of the spacecraft can be determined:

$$\mathbf{T}_b = \begin{bmatrix} -\sin\phi & -\frac{1}{2}\sin\phi & \frac{1}{2}\sin\phi & \sin\phi & \frac{1}{2}\sin\phi & -\frac{1}{2}\sin\phi \\ 0 & -\frac{\sqrt{3}}{2}\sin\phi & -\frac{\sqrt{3}}{2}\sin\phi & 0 & \frac{\sqrt{3}}{2}\sin\phi & \frac{\sqrt{3}}{2}\sin\phi \\ -\cos\phi & -\cos\phi & -\cos\phi & -\cos\phi & -\cos\phi & -\cos\phi \end{bmatrix} \mathbf{T} \tag{3.61}$$

$$\mathbf{M}_b = L \begin{bmatrix} 0 & -\frac{\sqrt{3}}{2}\cos\phi & -\frac{\sqrt{3}}{2}\cos\phi & 0 & \frac{\sqrt{3}}{2}\cos\phi & \frac{\sqrt{3}}{2}\cos\phi \\ \cos\phi & \frac{1}{2}\cos\phi & -\frac{1}{2}\cos\phi & 0 & -\cos\phi & \frac{1}{2}\cos\phi \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{T} \tag{3.62}$$

, where L is the distance between the thruster and the center of the hexagon shown in figure 3.6. The equations of motion of the translational dynamics are:

$$\dot{\mathbf{v}}_b = \frac{\mathbf{T}_b + \mathbf{C}_e^b g}{m} + \mathbf{v}_b \times \boldsymbol{\omega}^\intercal \tag{3.63}$$

$$\dot{\mathbf{r}}_b = \mathbf{C}_b^e \mathbf{v}_b \tag{3.64}$$

, where $\mathbf{r} = \begin{bmatrix} x & y & z \end{bmatrix}^\intercal$ is the position of the lander, $\mathbf{v}_b = \begin{bmatrix} u & v & w \end{bmatrix}^\intercal$ is the velocity vector in the body frame of the lander, $\boldsymbol{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^\intercal$ is the attitude vector, and g is the gravitational acceleration and m the mass of the spacecraft. $\mathbf{C}_b^e = \left(\mathbf{C}_b^e\right)^\intercal$ is the cosine direction matrix from the body frame to fixed frame on the planet. The article referred to defines the direction cosine matrix as:

$$\mathbf{C}_b^e = \mathbf{C}_x(\varphi)\mathbf{C}_y(\theta)\mathbf{C}_z(\psi) \tag{3.65}$$

Let $Q = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}$ be the quaternion and $\mathbf{I}$ the inertia matrix. The attitude dynamics can now be expressed as:

$$\dot{\boldsymbol{\omega}}^\top = \boldsymbol{I}^{-1} \cdot \boldsymbol{\omega}^\top \times (\boldsymbol{I} \cdot \boldsymbol{\omega}^\top) \tag{3.66}$$

$$\dot{\varphi} = p + \tan\theta(q\sin\phi + r\cos\varphi) \tag{3.67}$$

$$\dot{\theta} = q\cos\varphi - r\sin\varphi \tag{3.68}$$

$$\dot{\psi} = (q\sin\varphi + r\cos\varphi)/\cos\theta \tag{3.69}$$

The time derivative of the mass is:

$$\dot{m} = -\frac{1}{I_{sp}} \sum_i T_i(t) \tag{3.70}$$

The authors assumed a lander of a cuboid shape with the dimensions $a \times b \times c$. The inertia matrix is now:

$$\mathbf{I} = \begin{bmatrix} \frac{m}{12}(b^2+c^2) & 0 & 0 \\ 0 & \frac{m}{12}(a^2+c^2) & 0 \\ 0 & 0 & \frac{m}{12}(a^2+b^2) \end{bmatrix} \tag{3.71}$$

3.3.3.2 Training

The reward function has multiple components. The agents gets a reward for a successful landing:

$$r_{goal} = \lambda(h < 0 \text{ and } v_z < 0 \text{ and } ||v|| < v_{lim} \text{ and } \varphi < \varphi_{lim} \text{ and } \theta < \theta_{lim} \text{ and } \psi < \psi_{lim} \text{ and } ||\omega|| < \omega_{lim}) \tag{3.72}$$

A negative reward if given for deviating from a reference velocity. The reference velocity is defined as:

$$v_{ref} = \begin{cases} -\frac{\mathbf{r}-\mathbf{r}_1}{k_{v1}} & h \geq h_1 \\ -\frac{\mathbf{r}-\mathbf{r}_2}{k_{v2}} & 0 \leq h \leq h_1 \end{cases} \tag{3.73}$$

The agent gets a negative reward for deviating from the reference velocity:

$$r_{vel} = \beta\left|\left| v - v_{ref} \right|\right| \tag{3.74}$$

The next term in the reward function is a penalty for crashing:

$$r_{crash} = \eta(\varphi > \varphi_{lim} \text{ and } \theta > \theta_{lim} \text{ and } \psi > \psi_{lim}) \tag{3.75}$$

The goal is to make a fuel efficient controller. A negative reward is given for consuming fuel:

$$r_{fuel} = \alpha \frac{1}{I_{sp}} \sum_{i=1}^{6} T_i \tag{3.76}$$

The rewards r_{vel}, r_{crash} and r_{fuel} are all negative. To prevent that the agent crashes at the begin of the episode, a constant reward is given at the begin of the episode:

$$r_{constant} = \kappa \tag{3.77}$$

The total reward is now:

$$r = r_{fuel} + r_{vel} + r_{crash} + r_{constant} + r_{goal} \tag{3.78}$$

To improve the generalization ability of the agent, the tracking velocity deviation is put in the observation matrix instead of the position:

$$\delta v_b = \mathbf{C}_e^b(v - v_{ref}) \tag{3.79}$$

The observation vector used by the authors is:

$$s = \left[\delta v_{bx}, \delta v_{by}, \delta v_{bz}, \sin\varphi, \cos\varphi, \sin\theta, \cos\theta, \sin\psi, \cos\psi, p, q, r, q_0, q_1, q_2, q_3\right] \tag{3.80}$$

The action space has six dimensions, one for each thruster. The elements in the output vector are bounded by a number between -1 and 1 if it goes through a tanh activation function, thus $a_i \in [-1,1]$. The thrust is then:

$$T_i = \frac{T_{max} - T_{min}}{2} a_i + \frac{T_{max} + T_{min}}{2} \tag{3.81}$$

The network architecture is shown in figure 3.7 [28]. All of the three training methods use the same value function network. The DDPG and SAC algorithm use the same policy network while the SAC algorithm uses a different network. There are three hidden layers in all networks with 200 units with the ReLU activation function.

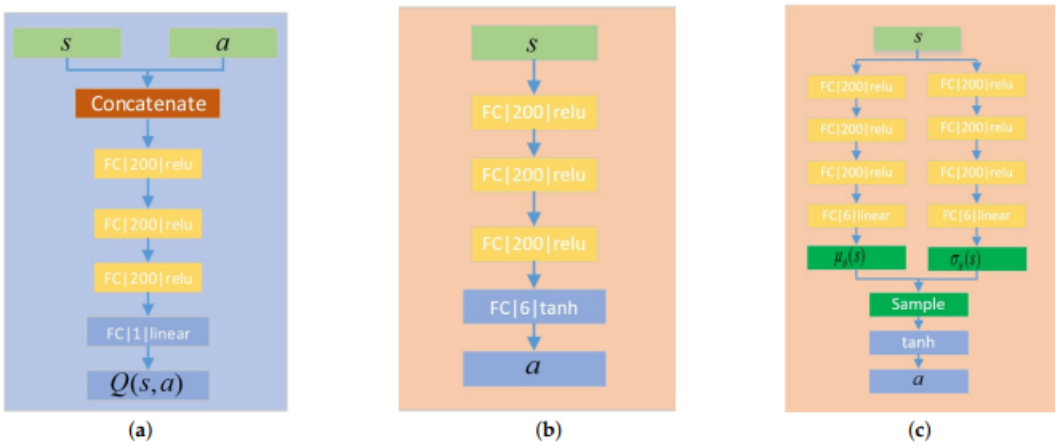


Figure 3.7: The value function network is shown on the left. The image in the middle shows the policy network during the DDPG and TD3 training session. The image on the right illustrates the policy network during the SAC training session [28].

3.3.3.3 Results

The parameters settings of the environment and the training sessions may be found in the paper. The initial value for δv_b were chosen randomly. The values for $\delta v_b x0$, $\delta v_b y0$ and $\delta v_b z0$ were all set between -3 and 3 m/s. The writers of the paper found out that the training with the DDPG method is unstable. For this reason, they set the learning rate for the value function network lower than for the TD3 and SAC method.

The DDPG method is slower in learning than the other two methods and the performance is the worst one. All of the reinforcement learning controllers are able to keep the velocity deviation within a certain range but the DDPG controller is the worst in this task. The deviation is oscillating around the reference velocity in the case of the DDPG controller while there is no oscillation if the velocity is tracked by the TD3 and SAC controller. The SAC controller track the velocity the best. The velocity deviation is even converging to 0. This can be seen in figure 3.8 [28].

The DDPG controller has a success rate of 74% which is much less than the success rate of the TD3 method with 92% and the SAC method with 96%. All the controllers were tested with 100 experiments. The writers did not analyse the landing accuracy in a mathematical way but they showed the landing points in a two dimensional graph. This can be seen in figure 3.9 [28].

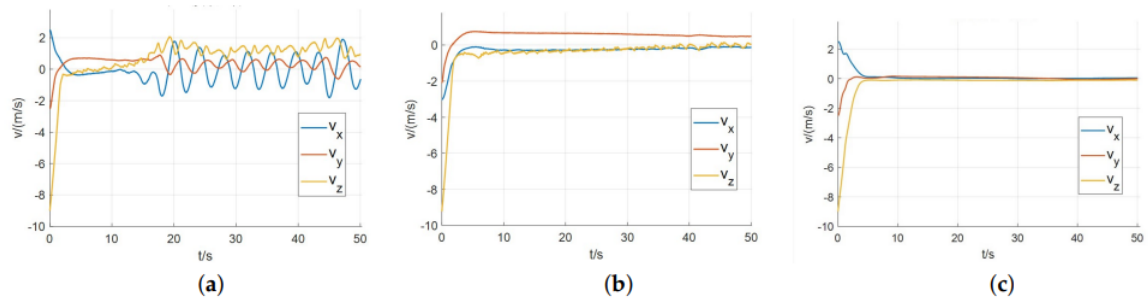


Figure 3.8: The velocity tracking error in time [28]. (a) DDPG (b) TD3 (c) SAC

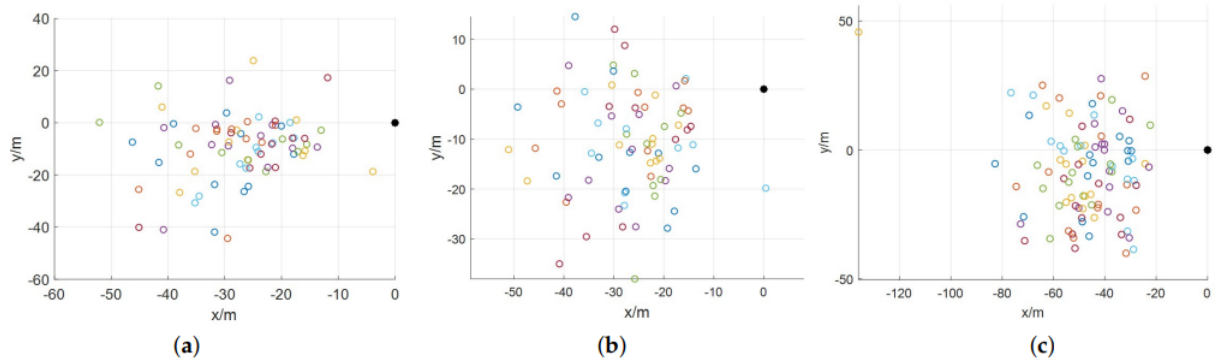


Figure 3.9: The landing accuracy of the three reinforcement learning controllers. [28]. (a) DDPG (b) TD3 (c) SAC

3.4. The rendezvous/docking guidance problem

3.4.1. Introduction

The planetary landing problem will be explained in more detail in this chapter. A paper of Gaudet, Linares and Furfaro is used as reference [1]. In this chapter, the question how the planetary landing could be modeled will be answered. At the end of the chapter, the fuel consumption of the reinforcement learning controller is compared with a fuel optimal controller. The problem is solved with six degrees of freedom.

Broida and Linares attempted to solve a three degrees of freedom docking problem by making use of the PPO algorithm [17]. They only took translational motion into consideration and ignored rotation. In this chapter, the question how the dynamics of a rendezvous/docking problem should be modeled will be answered partially. Only the modeling of translational dynamics by making use of the Clohessy–Wiltshire equations will be explained. Rotational motion was not taken into consideration by Broida and Linares.

3.4.2. Modeling of the problem

A keep-out zone was modeled to avoid collisions. This keep-out zone represented the docking target. The authors defined the reinforcement learning problem as follows:
Minimize:

1. Terminal position error: $||\mathbf{r}||$ at $t = t_f$
2. Total docking distance: $\sum_t ||\mathbf{r}||$

Subject to:

1. A 50 meter cubic keep-out zone
2. Clohessy-Wiltshire Equations of Motion

The keep-out zone was implemented by defining a soft constraint. A negative reward was given for being in the keep-out zone without terminating the episode.

To model a docking spacecraft, a relative orbit reference frame is used. The origin in this frame is in a stable orbit around the earth. The spacecraft in this origin is called the chief. A second spacecraft called the deputy and has to make it to the origin of the relative orbit reference frame. The relative orbit position vector has three components, $\boldsymbol{\rho} = \begin{bmatrix} x & y & z \end{bmatrix}^T$. If the assumption is made that the chief is orbiting in a circular orbit, the Clohessy–Wiltshire equations do apply:

$$\ddot{x} = 3n^2 x + 2n\dot{y} \tag{3.82}$$

$$\ddot{y} = -2n\dot{x} \tag{3.83}$$

$$\ddot{z} = -n^2 z \tag{3.84}$$

In the three equations above, n represents the angular velocity of the chief satellite:

$$n = \dot{\theta} = \sqrt{\frac{\mu}{a^3}} \tag{3.85}$$

where μ stands for the standard gravitational parameter and a for the orbit radius. Broida and Linares modeled a controlled deputy my modifying the Clohessy–Wiltshire (CW) equations:

$$\ddot{x} - 3n^2 x - 2n\dot{y} = u_x \tag{3.86}$$

$$\ddot{y} + 2n\dot{x} = u_y \tag{3.87}$$

$$\ddot{z} + n^2 \dot{z} = u_z \tag{3.88}$$

The state vector is given by $\mathbf{x} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T$ and the input matrix by $\mathbf{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}$. If the control is absent, the CW equations do have an analytic solution:

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} 4-3\cos(nt) & 0 & 0 & \frac{1}{n}\sin(nt) & \frac{2}{n}(1-\cos(nt)) & 0 \\ 6(\sin(nt)-nt) & 1 & 0 & -\frac{2}{n}(1-\cos(nt)) & \frac{1}{n}(4\sin(nt)-3nt) & 0 \\ 0 & 0 & \cos(nt) & 0 & 0 & \frac{1}{n}\sin(nt) \\ 3n\sin(nt) & 0 & 0 & \cos(nt) & 2\sin(nt) & 0 \\ -6n(1-\cos(nt)) & 0 & 0 & -2\sin(nt) & 4\cos(nt)-3 & 0 \\ 0 & 0 & -n\sin(nt) & 0 & 0 & \cos(nt) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ \dot{x}_0 \\ \dot{y}_0 \\ \dot{z}_0 \end{bmatrix} \tag{3.89}$$

The control thrust is modeled as an acceleration at the beginning of an integration step. The thrust is allowed to be positive and negative in all three directions.

3.4.3. Setup of the training

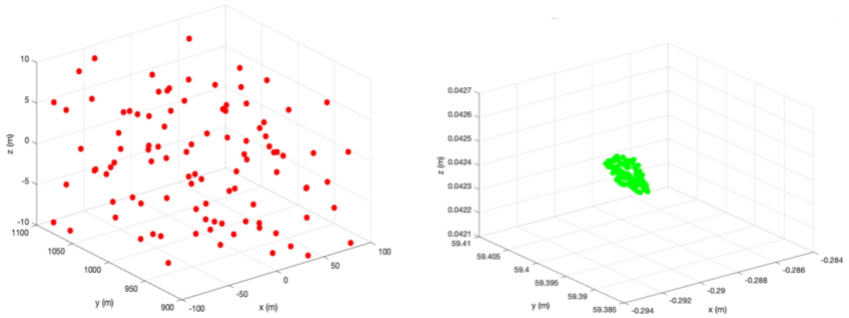
Two different approaches were considered in this experiment, the V-bar and the R-bar approach. The deputy had to dock on the point $\begin{bmatrix} 0 & 60 & 0 \end{bmatrix}$ m. The starting point for the V-bar approach was $\begin{bmatrix} 0 & 1000 & 0 \end{bmatrix} \pm \begin{bmatrix} 100 & 100 & 5 \end{bmatrix}$ m and for the R-bar approach it was $\begin{bmatrix} 1000 & 0 & 0 \end{bmatrix} \pm \begin{bmatrix} 100 & 100 & 5 \end{bmatrix}$ m.

A negative reward was given to the agent equal to the norm of the state-error vector. If this norm was less than 0.05 a bonus reward of 1 was given and a bonus reward of 2 if the norm was less than 0.01. A 50 meter keep-out zone centered at the origin was defined as a soft constraint. The agent was given a reward of -2 for being in the keep-out zone. This was done to avoid collisions. The control input was in a range between -0.005 N and 0.005 N for each direction. The authors did not define a control cost, which means that no penalty is given for controlling the spacecraft. The mass of the spacecraft was set to 1 kg.

The neural networks had three hidden layers of size 50. The ReLU function was used as activation function. The actor had six inputs and three outputs. The state-vector was the input while the output was an action chosen from a random distribution. The critic also had the state-vector as output. The output was the expected discounted reward given the current policy. The learning rate was initially halved every 4,000 episodes for both neural networks. This increment length was doubled if the learning rate was halved. The maximum increment length was 20,000 episodes. This means that the adaptiveness is declining over time. The training was running for 200,000 episodes.

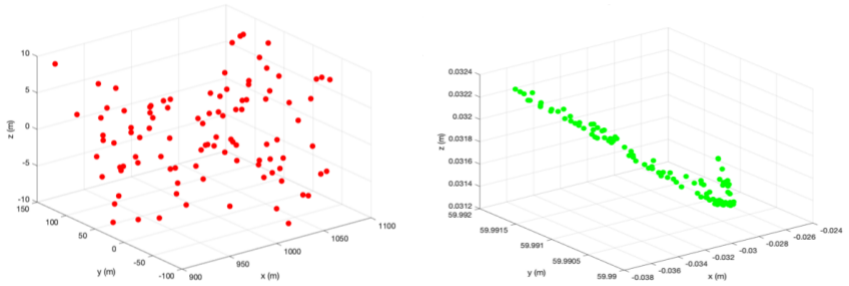
3.4.4. Results of the training

The learned policy was evaluated by saving the neural networks and running the simulation by making use of these networks. The actions were not chosen from a normal distribution, but the best known action was chosen instead which is the mean of the sampled normal distributions. The final position errors are on the order of centimeters. The distribution of the initial and final position of the deputy are illustrated in figure 3.10 and figure 3.11[17]. In figure 3.12, 100 trajectories are shown for the V-bar and R-bar approach. The cubic keep-out zone is also plotted in this figure. It can be concluded that the deputy is able to avoid the keep-out zone.



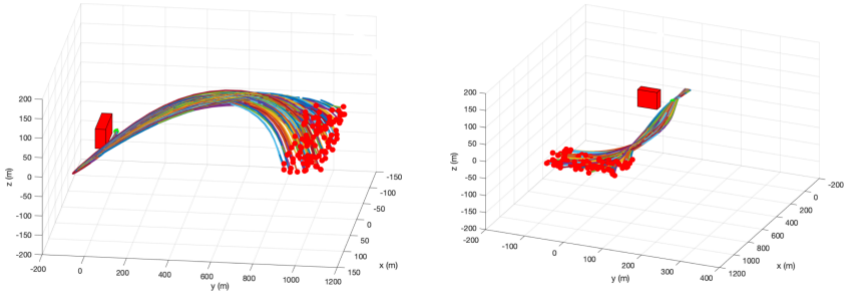
(a) V-bar Initial Position Distribution (b) V-bar Final Position Distribution

Figure 3.10: The distributions of the initial and final position given the V-bar approach [17].



(a) R-bar Initial Position Distribution (b) R-bar Final Position Distribution

Figure 3.11: The distributions of the initial and final position given the R-bar approach[17].



(a) 100 V-bar Trajectories (b) 100 R-bar Trajectories

Figure 3.12: A 100 trajectories are shown for the V-bar and R-bar approach making use of the learned policy [17].

3.5. Conclusions

The goal of this chapter is to explain the general ideas of reinforcement learning and to give an overview of applications of reinforcement learning in space guidance, navigation and control.

The chapter started with explaining four elements of reinforcement learning problems. Only model-free methods will be considered in this research. Model-based methods are only suitable if the next state and reward is known given the current state and an action. In the real world, the states are not known exactly because of sensor noise. The goal of this thesis is to find a policy robust to uncertainties.

The concept of discrete reinforcement learning methods and continuous methods was also explained in this chapter. The plan is to consider the planetary landing problem and for this thesis.

Continuous reinforcement learning methods may be divided in actor-only methods, critic-only methods and actor-critic methods. Value-based problems make use of only a critic while policy-based methods only use an actor. Policy-based methods converge better and are more effective in the case of continuous problems but are more likely to stick into a global optimum. There is also a relatively high variance in the policy in the case of policy-based methods. This variance can be reduced by using an critic. For this reason, actor-critic methods are prioritized in this thesis. Actor-critic methods do combine advantages from both worlds.

The actor and critic are typically neural networks. It is also possible to use a different estimator but these estimators are rarely found in literature about continuous reinforcement learning problems. For this reason, only neural networks will be considered as estimator.

Aerospace engineers have experimented with reinforcement learning in simulations. In literature, the PPO algorithm have proven to work well to solve trajectory problems. For this reason, the PPO algorithm will be prioritized in this research.

The reinforcement learning controllers have been proven to be more robust to unmodeled dynamics compared with the conventional controllers if noise is modeled in the training scenario.

4

Preliminary Analysis [2]

4.1. Introduction

In this chapter, a lunar lander will be considered. This lander is able to move in a two dimensional world. There are three degrees of freedom, two translations and one rotation.

In this chapter two controllers will be made, a PD controller and a reinforcement learning controller. The goal is to land the planetary lander on the center of the two dimensional world. This methods will be compared with each other.

The environment is created by OpenAI [21]. They made two lunar lander environments, one with a discrete action space and one with a continuous action space. Only the environment with the continuous action space will be considered. The training will be done by using the Stable-Baselines3 framework [30].

The goal of this chapter is to answer the question how reinforcement learning algorithms are performing in comparison with conventional control methods.

4.2. Reinforcement Learning

4.2.1. Environment

The goal is two land a lunar lander on the position (0,0). The lander has three engines. There is a main engine, a left engine and a right engine. There are eight observables in the model and two action-inputs.

States:

- Horizontal position x
- Vertical position y
- Horizontal speed v_x
- Vertical speed v_y
- Angle θ
- Angular velocity ω
- Left leg of the lander touches ground ($m = 1$ if True and $m = 0$ if False)
- Right leg of the lander touches ground ($n = 1$ if True and $n = 0$ if False)

[2] This chapter has been graded as part of the literature study course

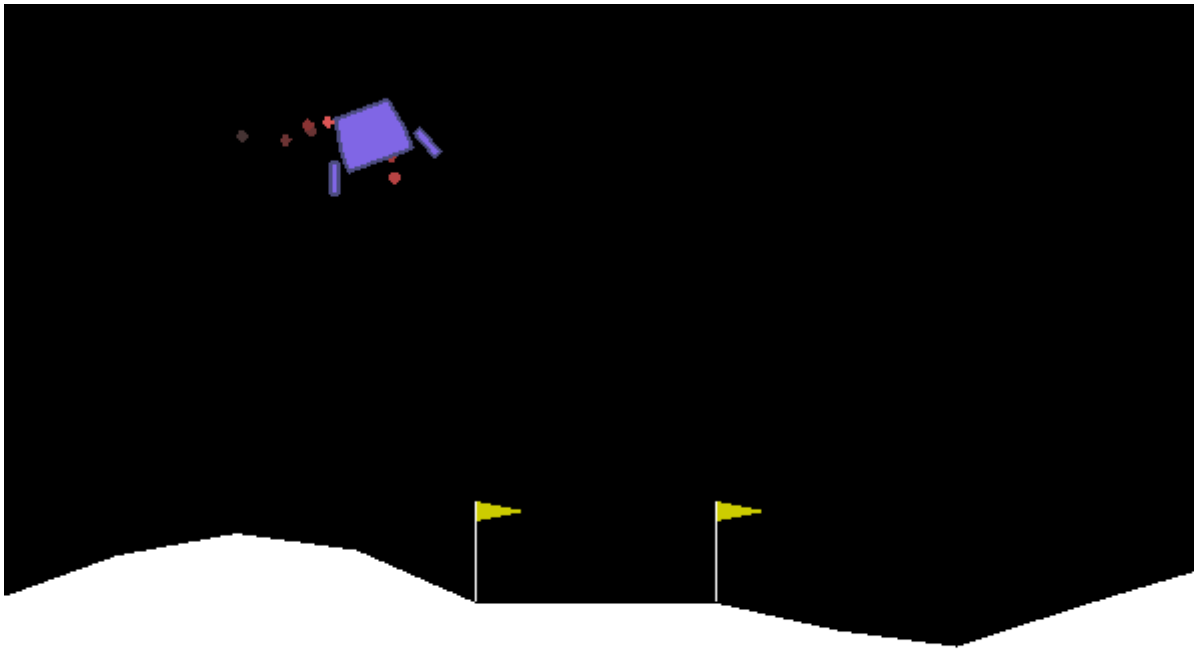


Figure 4.1: The environment of the lunar lander. The lander has to land with both legs on the ground.

The action space is a two dimensional vector. Both elements in this vector are a number between -1 and 1. The first input in this action-space vector controls the main engine. The engine is off if this number is between -1 and 0 and is on if this number is between 0 and 1. The throttle is between 50% power and 100% if it is on, for all of the engines.

The second element in the action-space vector controls the left and right engine. If this number is between -1 and -0.5 the left engine will be on, if it is between -0.5 and 0.5 the engine will be on and the right engine will be on if it is between 0.5 and 1. The main engine has more power than the side engines, 13 and 0.6.

A shaping reward is defined:

$$r_s = -100\sqrt{x^2+y^2} - 100\sqrt{v_x^2+v_y^2} - |\theta| + 10m + 10n \tag{4.1}$$

A difference is defined in the shaping reward between the current time step and previous time step:

$$\Delta r_s = r_{s_i} - r_{s_{i-1}} \tag{4.2}$$

Δr_s will be positive if the current state is better than the previous state. The reward per time step is now defined as:

$$r = \Delta r_s - 0.30 m_{power} - 0.03 s_{power} + r_{bonus} \tag{4.3}$$

In this equation, m_{power} is the power of the main engine while s_{power} is the power of the left engine or right engine. The problem is solved if the agent is able to get a score of 200.

$$r_{bonus} = \begin{cases} -100 & \text{crashing} \\ 100 & \text{landing} \\ 0 & \text{all other cases} \end{cases} \tag{4.4}$$

Table 4.1: The mean reward and standard deviation of the reward for different configurations of α, γ and n_{steps}. The sixth configuration was the best one and is chosen to be compared with a PD controller.

α	γ	n_{steps}	r_{mean}	σ_r
0.0003	0.99	2048	206	81
0.003	0.99	2048	188	70
0.00003	0.99	2048	118	89
0.0003	0.95	2048	-80	19
0.0003	1.00	2048	226	58
0.0003	0.99	1024	257	52
0.0003	0.99	4096	193	90

The exploration is done by introducing an Gaussian action noise. The agent is trained by making use of the PPO algorithm. The PPO algorithm is an actor-critic method maximizing the following objective function:

$$L^{CLIP}(\theta) = \hat{E}_t\Big[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{4.5}$$

In this equation, θ stands for a policy parameter, $\hat{E}_t$ is an expectation, ϵ is a hyperparameter, $\hat{A}_t$ is an estimated advantage and r_t is a probability ratio between a new and an old policy. The advantage function A and probability ratio $r_t(\theta)$ are defined as:

$$A = Q(s,a) - V(s) \tag{4.6}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{4.7}$$

4.2.2. Sensitivity analysis

For the sensitivity analysis, the learning rate α, the discount factor γ and the number of steps to run before each update n_{steps} are taken into consideration. The training is performed seven times for different values of α, γ and n_{steps}. A million time steps are performed for each training. The learned policies are evaluated by running a thousand episodes with the learned policy. The mean of the reward and the standard deviation in the reward for each configuration. Two hidden layers of 64 networks per layer are used for both of the networks. The result is shown in table 4.1.

It seems that a learning rate of 0.0003 is the best one to choose from the learning rates taken in the sensitivity analysis. Taking a discount rate of 1 seems to be the best option while lowering n_{steps} seems to be the best option. The learning curves are shown in figure 4.2-4.8. The policy learned with $\alpha = 0.0003$ is, $\gamma = 0.99$ and $n_{steps} = 1024$ is chosen to compare with a PD controller.

4.3. PD controller

The main engine can be used to control the altitude, while the side engines may be used to control the orientation of the lander. Before a PD controller could be made, a reference altitude and reference attitude should be defined [3] . Define an angle of descent of α radians. A cone will be the result if a landing angle of α radians is defined. The lander should ascend if it is outside the cone and descend if it is inside. This means that the power of the main engine should be increased if the lander is outside the cone and decreased if inside. The first control law will be,

$$y_{ref} = \Big|\tan\Big(\alpha\Big)x\Big| \tag{4.8}$$

The error in y may know be defined as,

$$y_{error} = y_{ref} - y \tag{4.9}$$

[3] PD controller inspired by William Fleshman, `https://github.com/wfleshman/PID_Control`

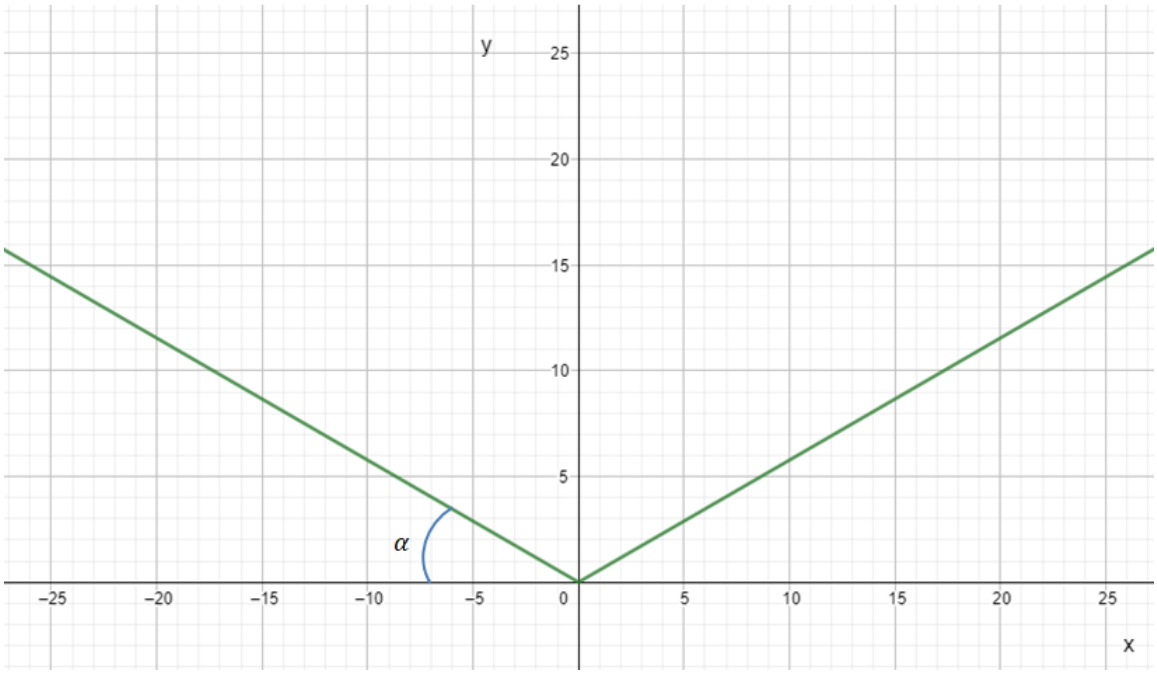


Figure 4.9: A cone will be the result if a landing angle of α radians is defined. The lander should ascend if outside the cone and descend if inside. In this example, an angle of 30 °or $\frac{\pi}{6}$ radians is shown.

The second reference point to be defined is the orientation angle θ. This angle should be oriented to the center. On the other hand, if the velocity in the horizontal direction is high, the lander may overshoot. In that case, the lander should slow down by orienting opposite to the direction of the center. The reference orientation angle θ could be defined to be proportional with x-coordinate with proportionality constant a and proportional with v_x with proportionality constant b. A maximum angle θ_{max} should also be defined. The angle θ_{ref} may now be defined by the following function:

$$\theta_{ref} = \begin{cases} -\theta_{max} & ax + bv_x \leq -\theta_{max} \\ ax + bv_x & -\theta_{max} < ax + bv_x \leq \theta_{max} \\ \theta_{max} & ax + bv_x > \theta_{max} \end{cases} \tag{4.10}$$

The error in θ is defines as,

$$\theta_{error} = \theta_{ref} - \theta \tag{4.11}$$

The derivatives $\dot{y}_{error}$ and $\dot{\theta}_{error}$ may be calculated in a numerical way. If y_{error_i} and θ_{error_i} are the tracking errors in the current time step, $y_{error_{i-1}}$ and $\theta_{error_{i-1}}$ are the errors in the previous time step. The time between two consequent time steps is Δt. The derivatives of y_{error} and θ_{error} at the current time step t_i will be:

$$\dot{y}_{error_i} = \frac{y_{error_i} - y_{error_{i-1}}}{\Delta t} \tag{4.12}$$

$$\dot{\theta}_{error_i} = \frac{\theta_{error_i} - \theta_{error_{i-1}}}{\Delta t} \tag{4.13}$$

The inputs are given by the main engine and side engines. The input from the main engine is called u_m and the input from the side engines is called u_s. The side engines are not controlled by two separate inputs in the model. The input will be negative if the left engine is on and positive if the right engine is on. A integral gain is not necessary because of the fact that there are no sensor errors.

$$u_m = k_{p_y} y_{error} + k_{d_y} \dot{y}_{error} \tag{4.14}$$

$$u_s = k_{p_\theta} \theta_{error} + k_{d_\theta} \dot{\theta}_{error} \tag{4.15}$$

It should also be noted that u_m and u_s are limited to a maximum. There is also a minimum power for all of the engines, which is half of the maximum power. The parameters mention in this subsection are shown in table 4.2.

Table 4.2: The parameters used in the PD controller

a	b	α	θ_{max}	k_{p_y}	k_{d_y}	k_{p_θ}	k_{d_θ}
1	1	60 °	45 °	1 .2	1.8	-2.4	-1.2

The PD gains were found by defining a four dimensional grid. Five number were tried for k_{p_y}, k_{d_y}, k_{p_θ} and k_{d_θ} each to calculate an average reward over 10 simulations per possible combination. Random seeds were used to be sure that all of the environments were the same for all combinations.

All the gains tried for k_{p_y} and k_{d_y} were positive. The values were between 0.6 and 3 including the boundaries. All the gains tried for k_{p_θ} and k_{d_theta} on the other hand were negative. The values were between -0.6 and -3 including the boundaries. The step between consecutive values for the gains that were tried was 0.6.

The values for the other parameters were constant. This means that $5^4 = 625$ combinations were tried.

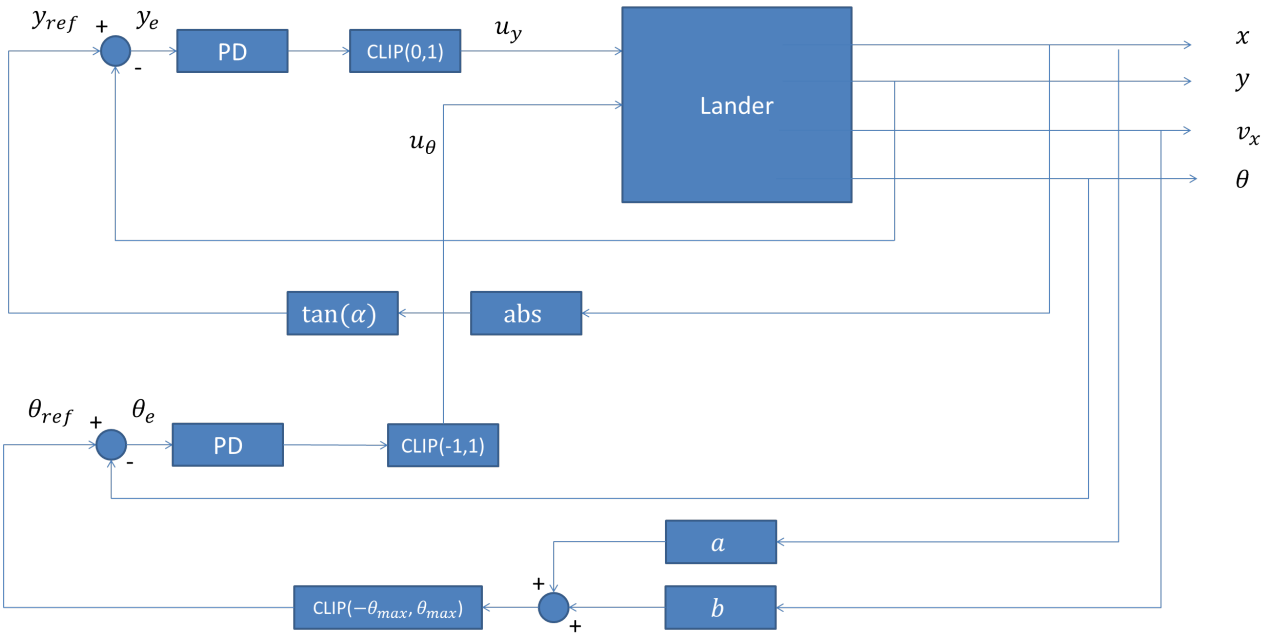


Figure 4.10: Block diagram of the PD controller

The PD controller is ran a thousand times. The mean of the total reward and standard deviation of the total reward is calculated. The mean of the total reward is 275.0 while the standard deviation of the total reward is 23.7.

4.4. Comparison PD Controller and Reinforcement Learning Controller

In this section, a comparison between the PD controller and reinforcement learning controller will be made. A few features will be compared. A thousand episodes will be run for both controllers. The features that will be considered are:

- Success rate:
- Number of time steps
- Control effort of the main engine

- Control effort of the side engines
- Position error of the landing
- Reward per episode

The result of this comparison is shown in table 4.3.

Table 4.3: The performance of the PD controller and RL controller

	PD controller		RL controller	
	mean	standard deviation	mean	standard deviation
succes rate	99.5%		96.7%	
total reward	275.0	23.7	258.6	46.5
total power main engine	137.0	11.1	144.0	46.0
total power side engine	133.3	15.3	134.5	55.9
position error landing	0.0452	0.0235	0.1173	0.0808
total number of time steps	229.5	22.8	245.3	85.4

It is noted that the PD controller is performing better on all aspects. The PD controller has a higher success rate and is collecting more rewards per episode. The PD controller is landing in a more fuel efficient way. The control effort of the main engine and side engines is lower for the PD controller. The PD controller is also able to land the lunar lander in less time steps. The PD controller is landing far more accurate. The mean of the landing error is about 2.5 smaller for the PD controller while the standard deviation of the position error is about 3.4. The standard deviations in table 4.3 were all lower for the PD controller which makes the PD controller more reliable.

The PD controller has a superior performance. It is far more accurate than the RL controller. It should be noted however that the performance of the RL controller also depends on the reward function. The RL controller may become more accurate for example if the position error is penalized more and the control effort less.

The conclusion is that a RL controller is not always better than a PD controller. There were no sensor errors and environmental disturbances in the lunar lander environment. The environment is also not changing which makes adaptive controllers unnecessary. The controller does not have to be robust and a PD controller is performing even better.

4.4.1. Conclusion

A RL and a PD controller were designed for a lunar lander simulation made by OpenAI. The RL controller was trained by the PPO algorithm. Seven different training sessions were performed to train the agent. The best one was chosen to compare with the PD controller.

The PD controller was made by making use of two control laws because of the fact that the lunar lander agent is controlled by a two-dimensional action space. No integral gain was used because the states are measured without errors.

Overall, the PD controller turned out to have the better performance. The PD controlled lunar lander failed less and was also more accurate. On the other hand, the RL controlled lunar lander was able to land with less control effort and in less time steps. The total collected reward was higher for the PD controlled lunar lander although no optimal controller was used.

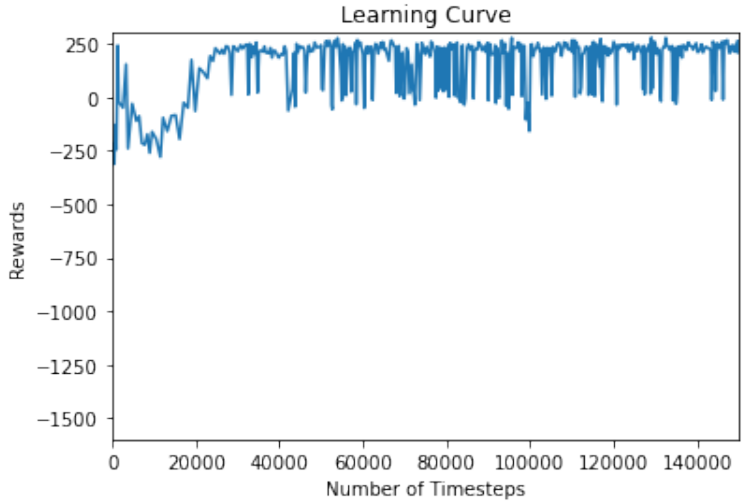


Figure 4.2: The learning curve if $\alpha = 0.0003$, $\gamma = 0.99$ and $n_{steps} = 2048$. The reward function is calculated every 10000 time steps.

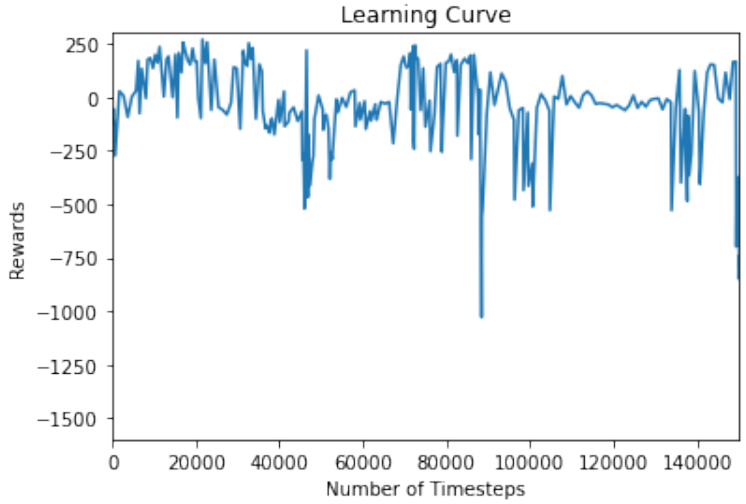


Figure 4.3: The learning curve if $\alpha = 0.003$, $\gamma = 0.99$ and $n_{steps} = 2048$. The reward function is calculated every 10000 time steps.

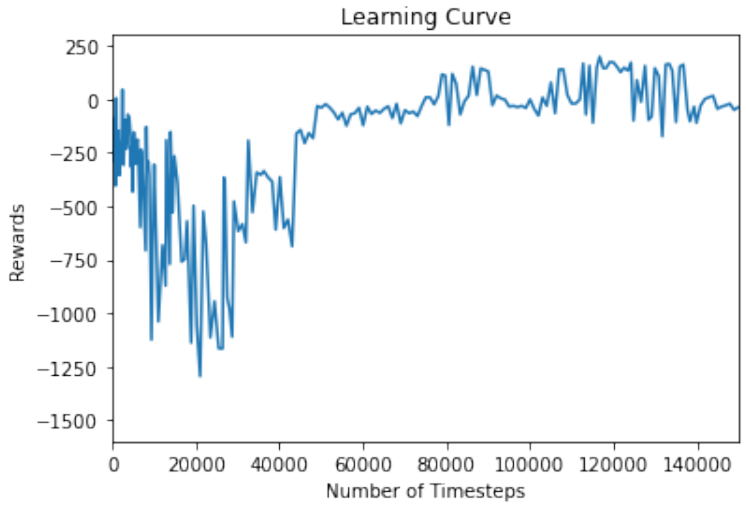


Figure 4.4: The learning curve if $\alpha = 0.00003$, $\gamma = 0.99$ and $n_{steps} = 2048$. The reward function is calculated every 10000 time steps.

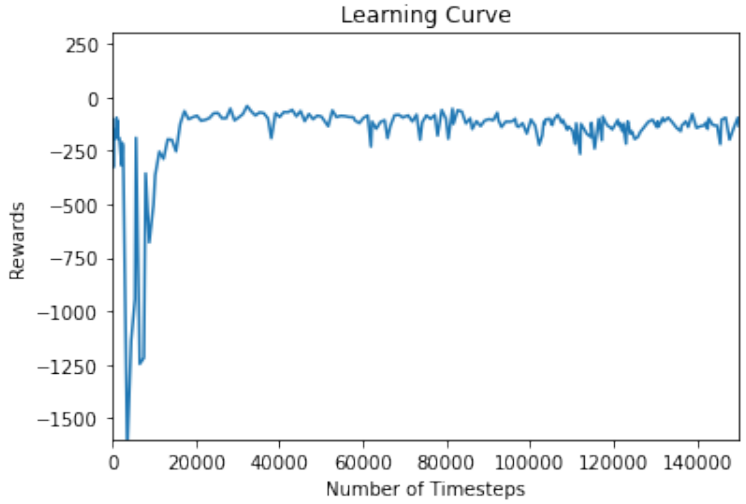


Figure 4.5: The learning curve if $\alpha = 0.0003$, $\gamma = 0.95$ and $n_{steps} = 2048$. The reward function is calculated every 10000 time steps.

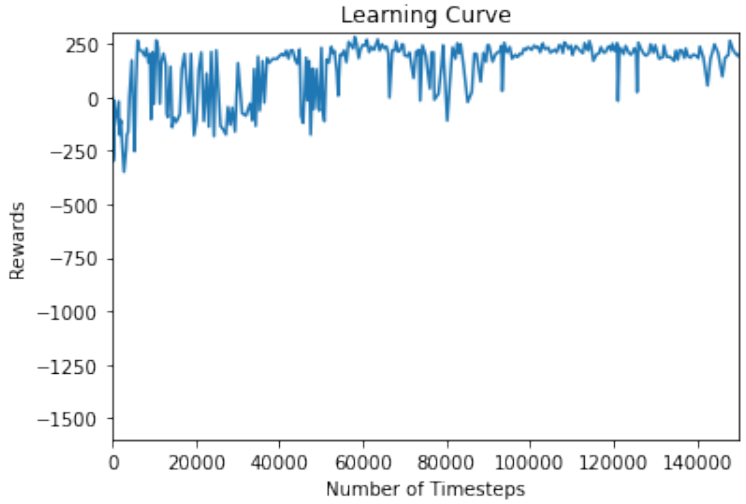


Figure 4.6: The learning curve if $\alpha = 0.0003$, $\gamma = 1.00$ and $n_{steps} = 2048$. The reward function is calculated every 10000 time steps.

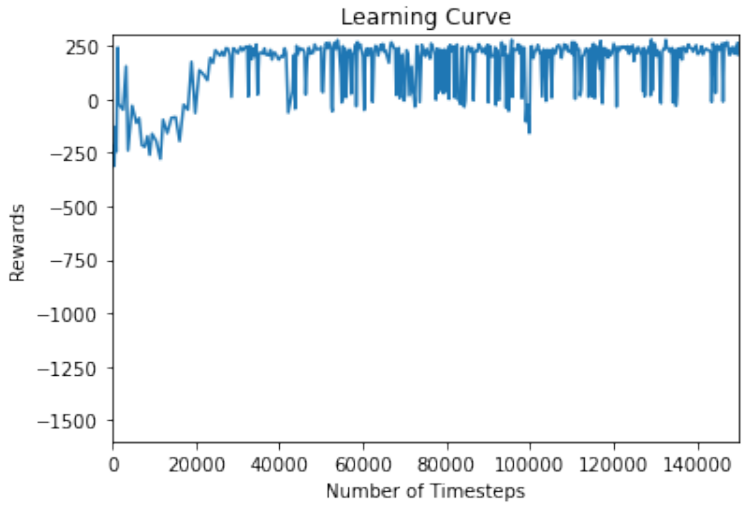


Figure 4.7: The learning curve if $\alpha = 0.0003$, $\gamma = 0.99$ and $n_{steps} = 1024$. The reward function is calculated every 10000 time steps.

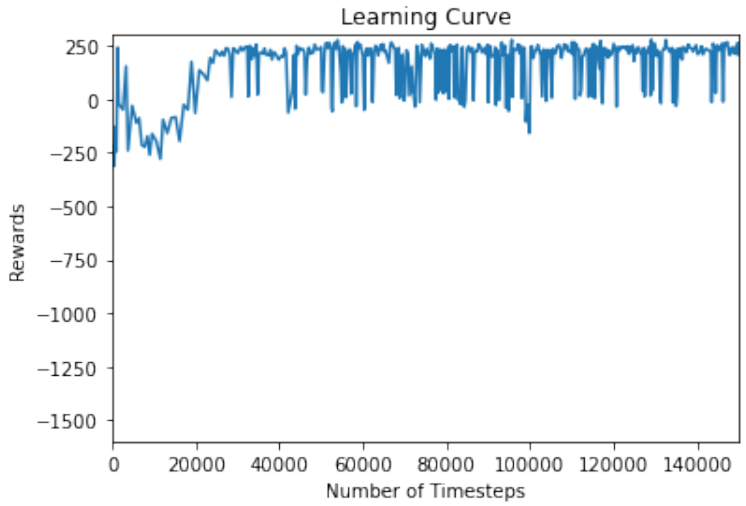


Figure 4.8: The learning curve if $\alpha = 0.0003$, $\gamma = 0.99$ and $n_{steps} = 4096$. The reward function is calculated every 10000 time steps.

5

Additional Results

5.1. Six Degrees of Freedom Control Allocation Model Validation

Assume that the lander has a radius of 4 meters. The mass of the thruster is 2000 kg to validate the model. The initial Euler angles are all 0°. The initial x-position and y-position are both 0 while the initial z-position is 600 m. The gravitation is acceleration is always -3.7 m/s^2. Four tests are performed:

- All forces and moments are 0. The expectation is that the altitude will decrease over time while the x-position, y-position and the Euler angles are staying 0.
- Hovering. The lander has a mass of 2000 kg. The gravity on the lander will be $m \cdot g = 7400$ N. The lander is expected to hover if $F_z = 7400$ N. The x-position, y-position and the Euler angles are also expected to stay 0.
- Setting $F_x = 2500$ N and $F_z = 7400$ N. The x-positon is expected to increase quadratically while the other forces and moments are constant.
- Setting $M_x = 1000$ Nm. The lander is expected to fall while phi is increasing.

The results of the tests are shown in figure 5.1-5.4. In figure 5.1, it is observed that the lander falls if no forces and moments are working on the lander. The lander is accelerating to the ground like expected while the Euler angles are staying 0. The lander needs a thrust of 7400 N to counter the gravity. In figure 5.2, it is observed that the lander is hovering if the vertical thrust is equal to 7400 N. If the horizontal thrust is 2500 N, while the vertical thrust is 7400 N, the lander is moving only in the horizontal direction, which is observer in figure 5.3. In figure 5.4 it could be observed that φ increases if a moment is working on the x-direction while the lander falls because the gravitational force is not countered until the angle limit is reached.

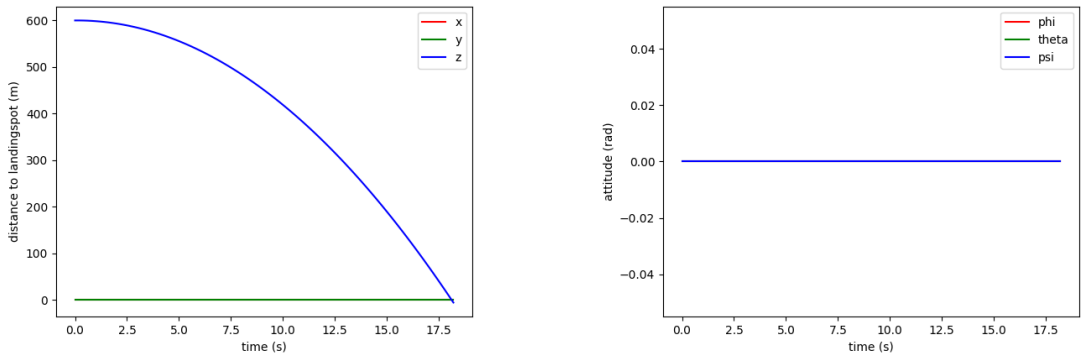


Figure 5.1: The position and Euler angles if all moments and forces are 0. The altitude is decreasing as expected. The angles are constant.

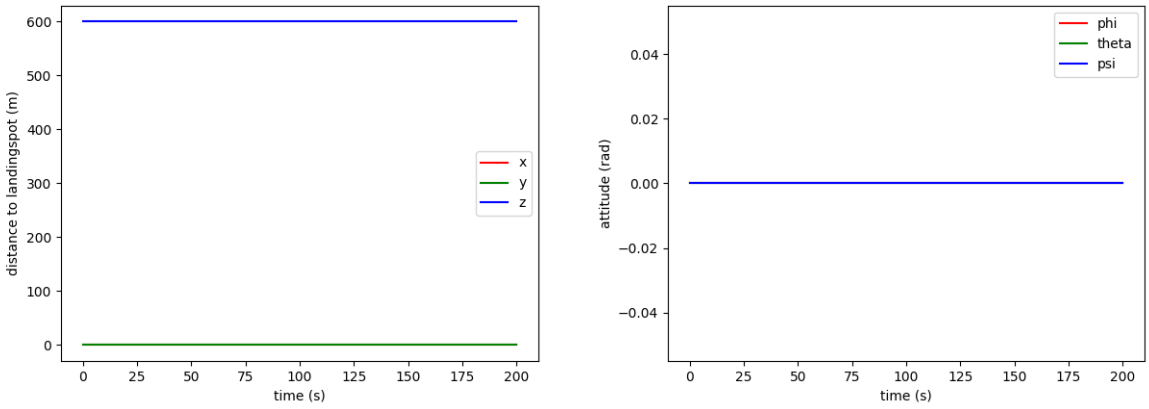


Figure 5.2: The position and Euler angles if $F_z = 7400$ N. The lander is hovering and the Euler angles remain constant.

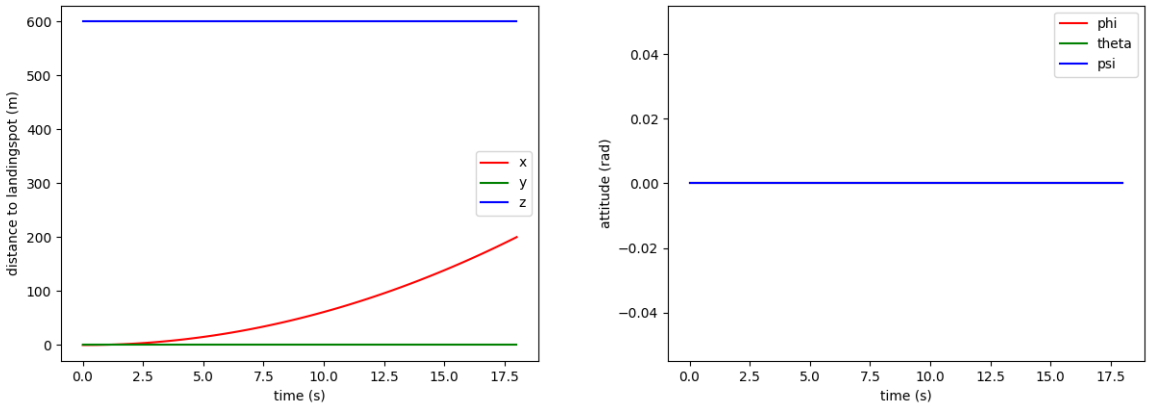


Figure 5.3: The position and Euler angles if $F_x = 2500$ N and $F_z = 7400$ N. The lander moves in the x-direction while the altitude is constant.

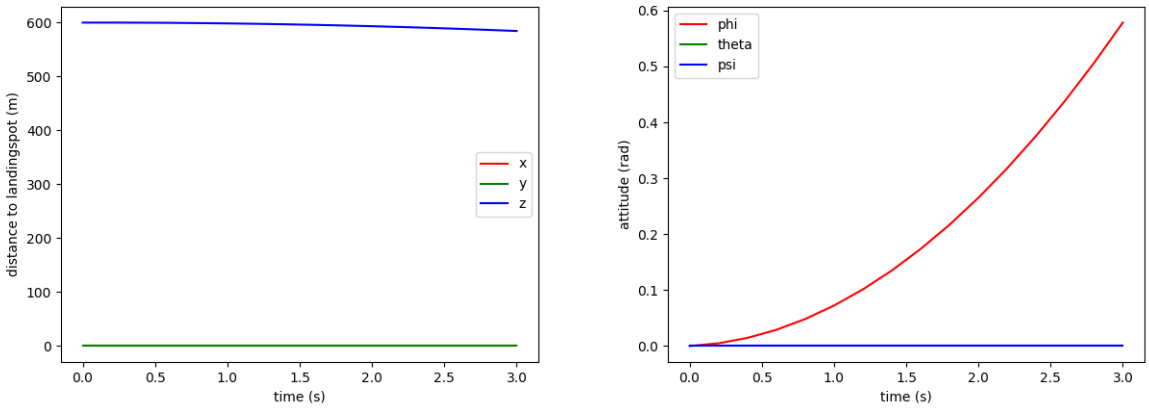


Figure 5.4: The position and Euler anlges if $M_x = 1000$ Nm. The Euler angle φ increases while the attitude drops due to lack of a counterforce to counter the gravity.

5.2. Six Degrees of Freedom Thruster Model Validation

Assume that the lander has a radius of 4 meters. The thrusters are positioned 3.5 meters from the center of the lander. The initial mass of the thruster is 2000 kg. The initial Euler angles are all 0°. The initial x-position and y-position are both 0 while the initial z-position is 450 m. The specific impulse I_sp is 225 s. The gravitational acceleration is 3.7 m/s^2. The value for ϕ_t from figure **??** is 30°

Five tests are performed:

- All thrusters are off. The expectation is that the altitude will decrease over time while the x-position, y-position and the Euler angles are staying 0.
- Hovering. The lander has a mass of 2000 kg. The gravity on the lander will be $m \cdot g$. The vertical force of the thruster has to be $(m \cdot g)/\cos\phi_t$. The force per thruster is $\frac{m \cdot g}{5 \cdot \cos\phi_t} = 1709$ N. The lander is expected to gain a little altitude if all thrusters have a force of 1709 N. The reason for this is the loss of mass while burning fuel. The x-position, y-position and the Euler angles are expected to stay 0.
- Hovering if the mass is kept constant while burning fuel. All thruster have a force of 1709 N but the mass is kept at 1709 N. The lander is expected to hover. The x-position, y-position and the Euler angles are expected to stay 0.
- Turning on only T_1 with a force of 1000 N. This thruster is expected to push the lander in the negative x-direction. This thruster will also lead to a clockwise moment around the y-axis. This means that $theta$ will become negative while the other Euler angles are 0. The lander is also expected to lose altitude because of the fact that the vertical forces are not enough to maintain altitude. The y-position is expected to stay 0 because no forces are pushing the lander in that direction.
- Turning on all thruster but T_1 with a force of 250 N. The expectation is that the thrusters would push the lander in positive x-direction because T_1 is not countering the forces of the other thrusters anymore to keep the x-position constant. The thrusters are also expected to lead to a counter-clockwise moment around the y-axis which means that θ will become positive. The y-position is expected to stay 0 because no forces are pushing the lander in that direction.

The results of the tests are shown in figure 5.5-5.9. In figure 5.5 it is observed that the lander falls if no forces and moments are working on the lander. It is calculated that each thrust has to deliver 1709 N to counter the gravity. In figure 5.6, it is observed that the lander is hovering if the vertical thrust is equal to 7400 N. The lander is moving upward because it losses mass. If the mass is kept constant like in figure 5.7, the lander will not move in vertical direction. In figure 5.8 it is observed that the the lander is moving in the negative x-direction and is rotating in clockwise moment around the y-axis if T_1 is 1000 N while the other thrusters are off. This means that θ will become negative while the other Euler angles are 0. There is a singularity at 90° because the Euler angle representation is used instead of the quaternion representation. The Euler angle representation is not correct if one of the angles is approaching 90°. This is not a problem because the thruster angle limit is set much less than 90°. If thruster T_1 is off, while the other thrusters are on, the lander is showing the opposite behavior which is observed in 5.9.

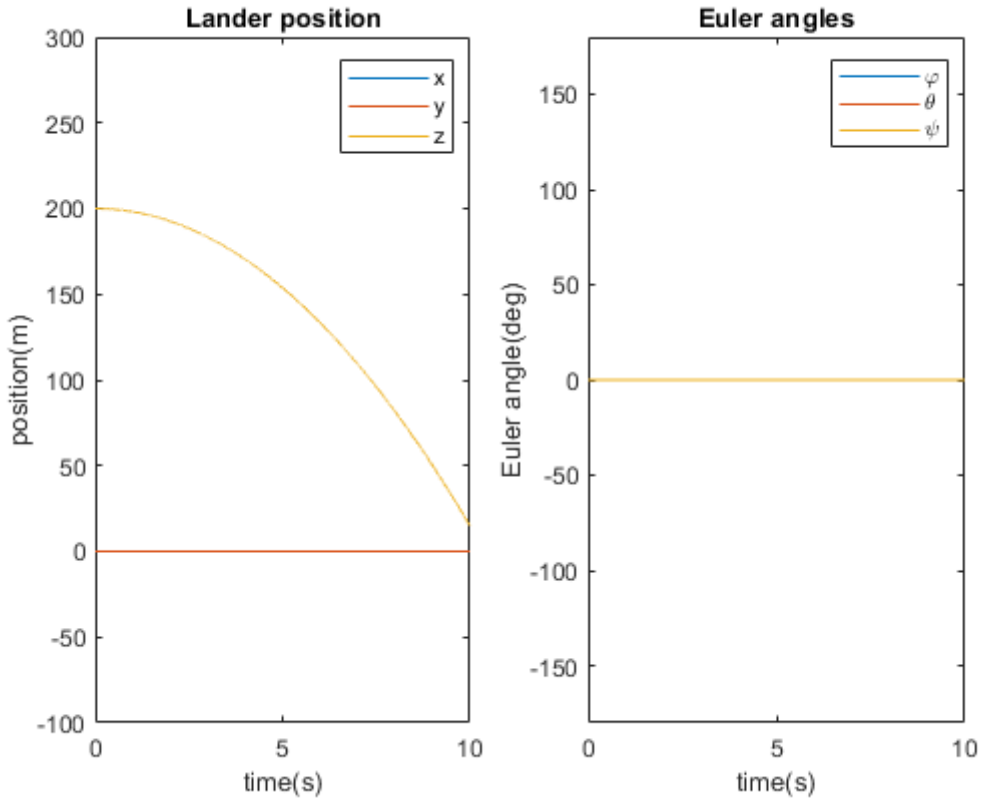


Figure 5.5: The position and Euler angles if all thrusters are off. The altitude is decreasing while the Euler angles are constant.

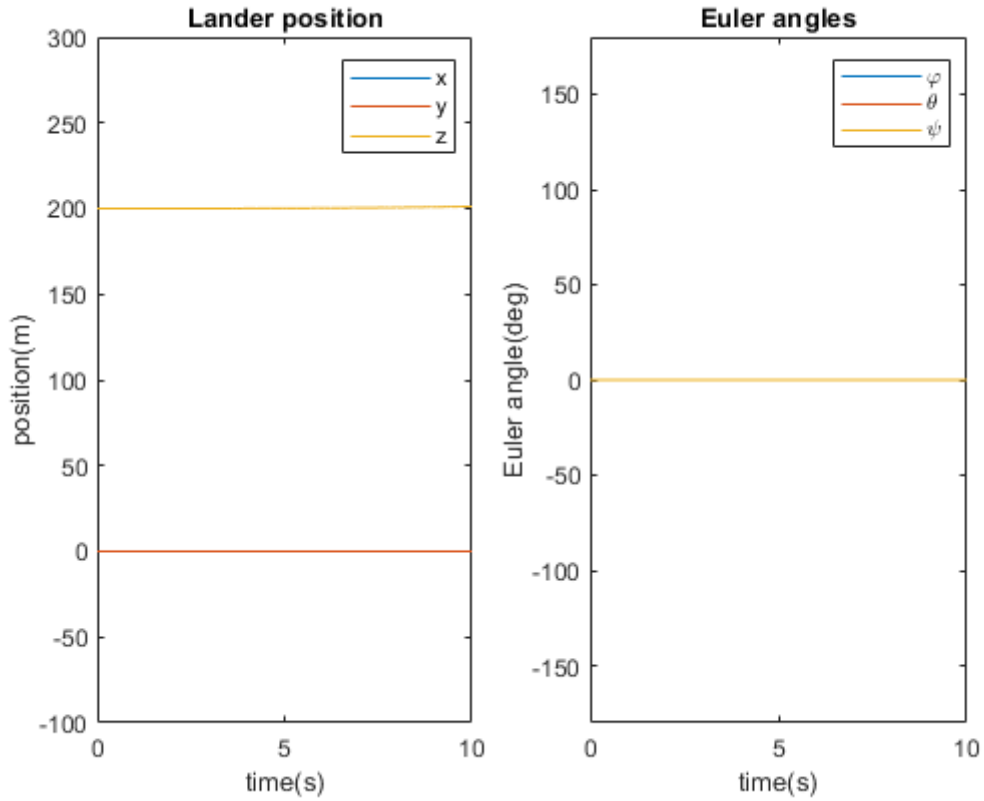


Figure 5.6: The position and Euler angles if all thrusters have a force of 1709 N. This should be just enough force to make the thruster hover. The lander gains 4 meter of attitude because of the fact that it loses mass while burning fuel.

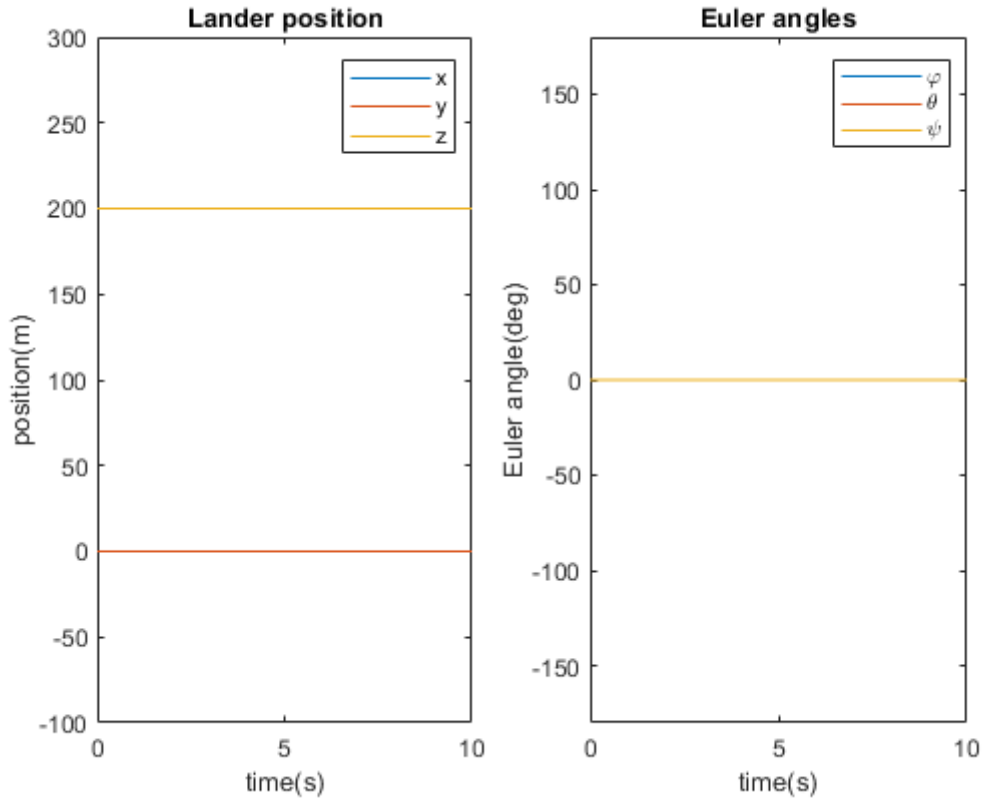


Figure 5.7: The position and Euler angles if all thrusters have a force of 1709 N and the mass is kept constant. The lander is not gaining altitude.

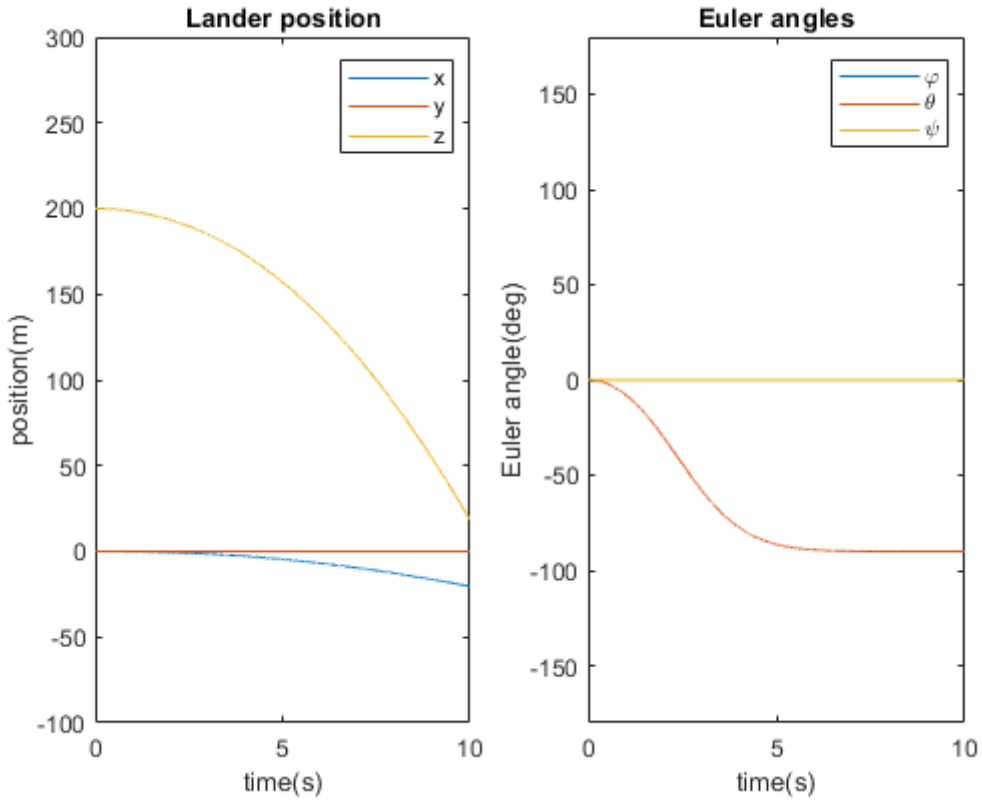


Figure 5.8: The position and Euler angles if only T_1 is on with a force of 1000 N.

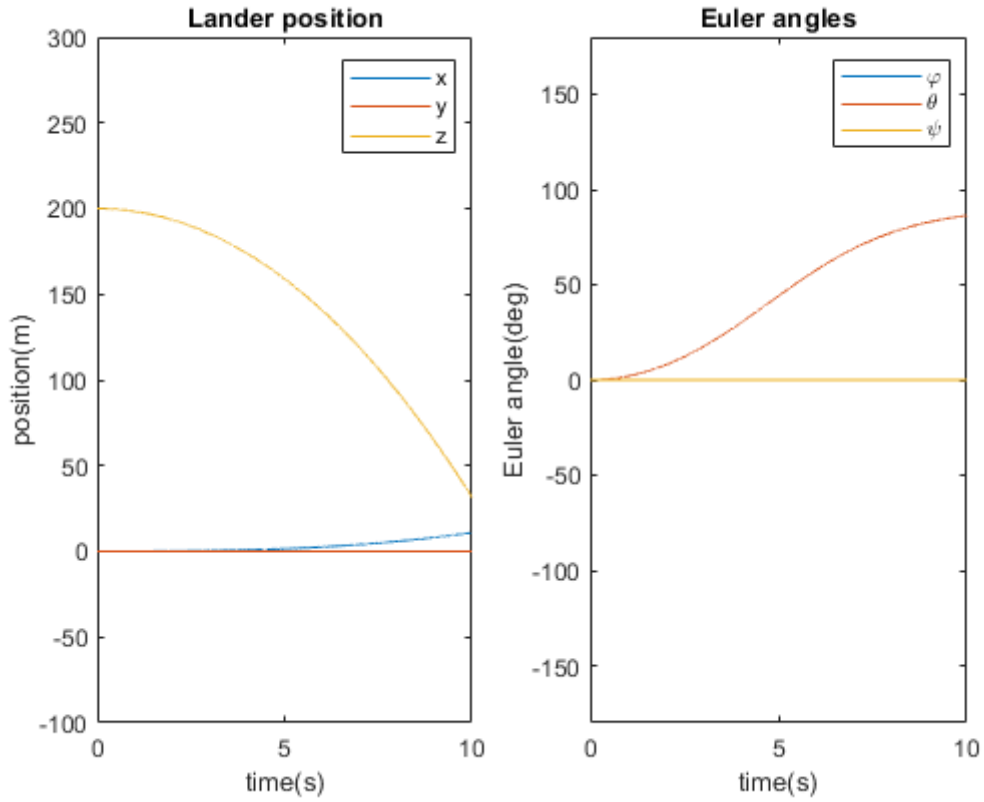


Figure 5.9: The position and Euler angles if only T_1 is off while the other thrusters are on with a force of 250 N.

5.3. PD and RL controller Robustness Test

The PD controller and RL controller were tuned for a mass between 1900 kg and 2100 kg. The robustness of the PD and RL controller were also tested. This was done by comparing the same same variables if the mass is set equal to 1500 kg and 2500 kg. The results are shown below:

	PD		RL	
	Mean	Standard deviation	Mean	Standard deviation
Win rate	97.6%		99.3%	
Reward	439.4	76.8	418.6	45.8
Distance (m)	0.65	0.35	6.67	0.54
$\int Fdt$ (Ns)	317469	9205	342901	9178
$\int Mdt$ (Nms)	4864	1797	51342	1652
Time (s)	26.1	0.75	28.0	0.7

Table 5.1: The PD and RL controller compared for a lander with a mass of 1500 kg

	PD		RL	
	Mean	Standard deviation	Mean	Standard deviation
Win rate	100%		88.8%	
Reward	424.4	13.0	341.4	173.0
Distance (m)	0.67	1.52	7.61	1.23
$\int Fdt$ (Ns)	438078	13357	440600	16747
$\int Mdt$ (Nms)	8069	3123	43772	2773
Time (s)	25.3	0.8	25.3	1.3

Table 5.2: The PD and RL controller compared for a lander with a mass of 2500 kg

Also the gravity was changed to compare the robustness of the PD and RL controller. The controllers were both tuned for a gravitational acceleration of 3.7 m/s^2. The gravitational acceleration were set to 3.3 m/s^2 and 4.1 m/s^2 to compare the controllers.

	PD		RL	
	Mean	Standard deviation	Mean	Standard deviation
Win rate	99.2%		98.6%	
Reward	437.3	45.2	407.6	65.8
Distance (m)	0.43	0.28	7.03	0.70
$\int Fdt$ (Ns)	357450	12996	381942	12317
$\int Mdt$ (Nms)	6500	2441	48671	1863
Time (s)	25.7	0.7	27.5	0.8

Table 5.3: The PD and RL controller compared if the gravitational acceleration is set to 3.3 m/s^2

	PD		RL	
	Mean	Standard deviation	Mean	Standard deviation
Win rate	100.0%		97.7%	
Reward	437.2	10.5	401.7	83.5
Distance (m)	0.43	0.31	7.02	0.67
$\int Fdt$ (Ns)	398898	12962	382540	12020
$\int Mdt$ (Nms)	6714	2442	48760	1887
Time (s)	25.7	0.8	27.5	0.8

Table 5.4: The PD and RL controller compared if the gravitational acceleration is set to 4.1 m/s^2

The PD controller is found to perform better. The PD controller is also found to be more robust to changes in the environment, for example a change in the lander mass or gravitational acceleration.

5.4. PD controller Test

A PD controller was made to compare with the RL control allocation controller. The forces and moments were controlled independently. It is observed in figure 5.13, 5.11 and 5.12 that the PD controller is much better in tracking the designated landing spot, reference velocity and reference Euler angles compared with the RL control allocation model.

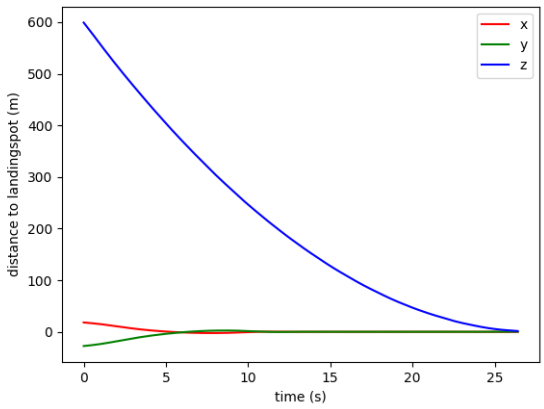


Figure 5.10: The position of the lander controlled by the PD controller. The designated landing spot is (0,0,0). The lander is landing more accurate in comparison with the PD controller.

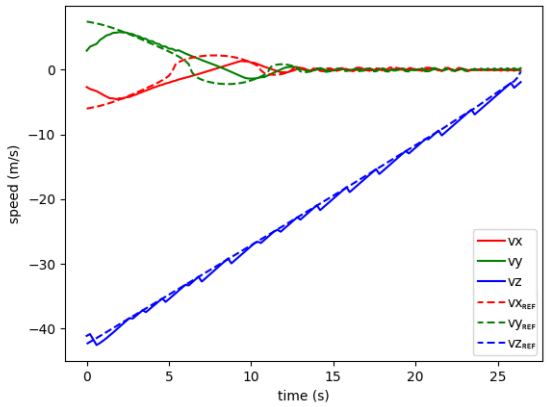


Figure 5.11: The speed of the lander controlled by the PD controller. The reference velocity is tracked better in comparison with the RL controller.

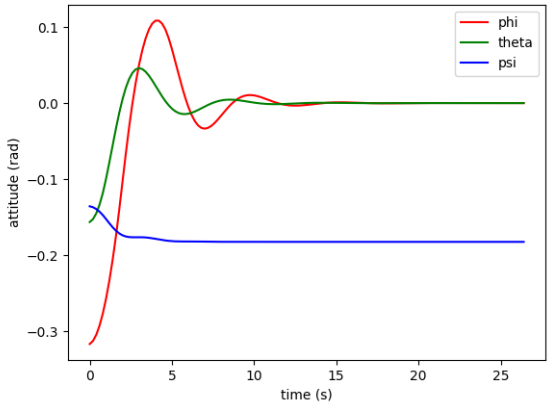


Figure 5.12: The attitude of the lander controlled by the PD controller. The goal is to land without pitch and roll.

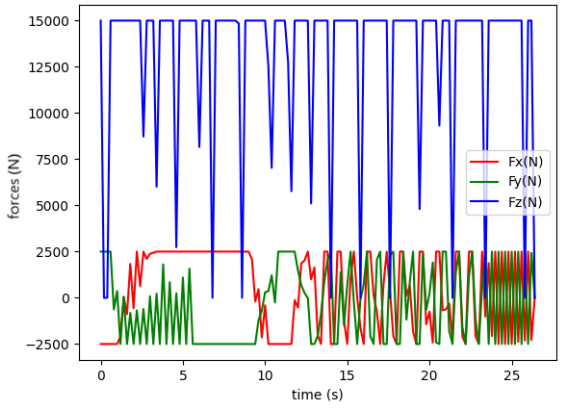


Figure 5.13: The forces of the lander controlled by the PD controller. The maximum thrust is 15000 N in the lateral direction and 2500 N for both radial forces.

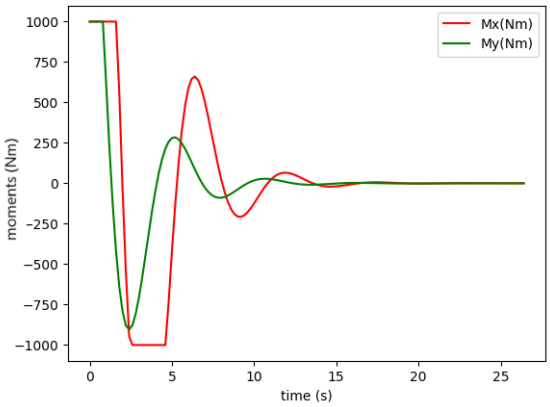


Figure 5.14: The moments of the lander controlled by the PD controller. There is no moment to control the yaw directly.

6

Conclusions

In this chapter, the answers of the research questions will be summarized briefly. The chapter will end with recommendations.

6.1. Conclusions

1. *What is the state of the art of reinforcement learning in space guidance, navigation and control?*

   Reinforcement learning is not applied on real space crafts. All of the research so far have been done in simulations. Research on the field of planetary landings, docking, satellite attitude control and hovering have been done.

2. *What are reasons to apply reinforcement learning instead of conventional control methods?*

   A good reason to use reinforcement learning is that the dynamics don't have to be represented as an ordinary differential equation. In fact, there are no constraints at all in the dynamics representation. Because of this,there are less constraints in the model. This means that reinforcement learning agents can learn form higher fidelity models. Many conventional control methods require the dynamics to be linearized. This linearized equations are only valid near the trim points. Reinforcement learning agents on the other hand are able to learn if the dynamics are non-linear and uncertain. Reinforcement learning agents are also able to learn directly to compensate for sensor error and environmental distortions .It may also be a good option to use reinforcement learning algorithms if the environment is not stationary. Reinforcement learning does not need full information about the environment because it relies on measured data.

3. *What are the challenges of reinforcement learning in space guidance, navigation and control?*

   First there is the curse of dimensionality. If the number of dimensions grow, the state-space grows exponentially. The second problem is the curse of real-world samples. Exploration should be safe if reinforcement learning is applied on-line. Switching between controllers may be a way to do this. The dynamics also may change as a result of external factors, for example temperature effects or wear. External factors may also have an influence on the sensors. The third problem is is the curse of under modeling and model uncertainty. Reinforcement learning may learn off-line in a computer model. The learned policy can be transferred to the real world. Creating a good model is important if this approach is chosen but may be challenging. Due to accumulating model errors, a real robot may diverge from the robot in a simulation. Transferring policies from a simulation to the real world works the best for self-stabilizing systems. In unstable systems, small model variations may have very big consequences. The last problem is the curse of goal specification. The goal is specified by a reward function. Specifying a reward function may be easier than specifying the desired behavior. However, choosing a good reward function is not always simple. The reward may also be exploited in a way not meant by the designer.

4. *How could the dynamics of a planetary lander be modeled?*

   The spacecraft is modeled as a semi-sphere. There are 12 observables, x, y, z, v_x, v_y, v_z, φ, θ, ψ, p, q and r.

   Two different methods are used in this thesis to control the spacecraft, a control allocation model and a thruster model. The control allocation model has five outputs, F_x, F_y, F_z, M_x and M_y. There is a wind in an Martian environment and also uncertainties in spacecraft mass and gravitational acceleration are present.

   The same spacecraft could also be modeled by a thruster model. There are five thrusters. The outputs are T_1, T_2, T_3, T_4 and T_5 which are all thruster forces. The thrusters are placed under the spacecraft in a pentagonal line-up. The moment and forces in the control allocation model cannot be controlled independently in the thruster model.

5. *How could a 3-DOF planetary landing be completed successfully by training a reinforcement learning agent to land it?*

   The PPO algorithm is used to train the 3-DOF lander. Many papers have been read about RL algorithms, this algorithm seems to outperform all other algorithms. This was also observed while doing the research, this method seems to be the fastest of all methods. Because of the complex nature of the problem and the time limit of the Delft Blue, it is important to use a fast algorithm to do as many as possible training sessions in a limited amount of time. A penalty is given for being off from the landing spot, control effort and being off from the reference velocity.

6. *What is a good set of learning rate, discount rate, entropy coefficient and clip range to train a 3-DOF lander?*

   A good set of training parameters and reward function were found empirically. Four training parameters were varied, the learning rate, discount rate, entropy coefficient and a clip range. A nominal value was set for each training parameter. The nominal learning rate was 0.0003, the nominal discount rate was 0.95, the nominal entropy coefficient was 0.015 and the nominal clip range was 0.20. Four or five different values were chosen for each hyperparameter. One of the training parameters were varied while the other three were set equal to the nominal value. Because of the stochastic nature of the training, five training sessions were performed for each set of training parameters. After each training session, 1000 episodes were ran to evaluate the policy. The conclusion is that the best set of hyperparameters for the learning rate, discount rate, entropy coefficient and clip range are 0.0003, 0.99, 0.005 and 0.20

7. *How is a 5-DOF lander performing if the same set of learning rate, discount rate, entropy coefficient and clip range are used to train the lander for a control allocation model and thruster model?*

   Both reinforcement learning models are evaluated by a thousand episodes. The control allocation model is the easier model and fastest to train. The control allocation model reaches its optimal performance after about 450 000 episodes while it need about 750 000 episodes to reach its optimal performance if it is controlled by a thruster model.

   The landing success rate seems also to be higher for the control allocation model. The success rate is 97.6% for the control allocation model and was initially 57.6% for the thruster model. The win rate converged to about 80% if the initial condition ranges are halved.

   The control allocation RL controller was also performing better than the thruster model RL controller if the landing accuracy and landing time are considered. The first one was able to land closer to the designated landing spot in a shorter amount of time.

8. *How is the 6-DOF reinforcement learning controller, trained by the same set of hyper-parameters, performing in comparison with a PD controller in the nominal situation?*

   The PD controller seems to outperform the RL controller in almost all aspects. The landing success rate was similar for both controllers, but the PD controller was able to land the spacecraft more accurately with less control effort.

9. *How is the 5-DOF reinforcement learning controller performing in comparison with a PD controller for a mass and gravitational acceleration different than the controllers were tuned for?*

   The PD controller also seems to outperform for the non-nominal situation, where the mass and gravitational acceleration are different than the controllers were tuned for.

10. *How is the 5-DOF reinforcement learning controller performing if one of the thrusters fail.*

    The thruster model RL controller was not able to learn to land if one of the thruster fails. The problem becomes harder to solve. A explanation for this could be the loss of symmetry of the line-up of the thrusters. Gaudet, Linares and Furfaro have shown that four thrusters should be enough to land a spacecraft by using a RL controller [1]. It was shown however that the agent was to learn to cope with the situation that one of the thrusters is able to deliver 75% thrust. The win rate becomes 13%, but this is better than the case if the lander has to land with a failed thruster without having been trained to do so.

6.2. Recommendations

The end goal was to create a spacecraft with a failed thruster on Mars, trained by reinforcement learning. This was simulated by turning off on of the thrusters. The spacecraft was not able to land safely with four thrusters. This may be because the thrust becomes asymmetric if one of the thrusters is turned off.

Further research have to be done to create a fault-tolerant controller and to improve the landing success rate of the thruster model reinforcement learning controller for a failed thruster. A possible way to create a more fault tolerant controller could be to make a thruster model with more thrusters. In the case that one of the controllers fail, the asymmetric forces are less. There are also disadvantages. Adding thruster will also add weight and the training will become more intensive because it increases the dimension of the action space. Faster computers should be used or the training should be done on parallel environments by using multiple CPU's.

The reference velocity was derived by the assumption that the acceleration has to be constant. This is the case if the forces and lander mass are constant. The lander mass is however not constant because fuel is burned. In the future, a reference velocity that depends on the lander mass could be used instead. Further research have to be done to create a fault-tolerant controller and to improve the landing success rate of the thruster model reinforcement learning controller. It was however shown that a RL controller is better able to cope with 75% if it is trained to do so, compared with a RL controller were the agent is trained by using fully working controllers.

Another option would be the introduction of curriculum learning [29]. The agent is trained in an adapting environment. At the start of the training session there is no failed thruster. The failure is introduced step by step by decreasing the maximum thrust for one thruster.

Bibliography

[1] Gaudet, B., Linares, R., & Furfaro, R. (2020). Deep reinforcement learning for six degree-of-freedom planetary landing. Advances in Space Research, 65(7), 1723–1741. https://doi.org/10.1016/j.asr.2019.12.030

[2] Ogata, K. (2010). Modern Control Engineering. Upper Saddle River, NJ, USA: Prentice Hall.

[3] Yang, X., Liu, D., & Huang, Y. (2013). Neural-network-based online optimal control for uncertain non-linear continuous-time systems with control constraints. IET Control Theory & Applications, 7(17), 2037–2047. https://doi.org/10.1049/iet-cta.2013.0472

[4] Schulman, John, e.a. 'Proximal Policy Optimization Algorithms'. arXiv:1707.06347 [cs], augustus 2017. arXiv.org, http://arxiv.org/abs/1707.06347

[5] Oestreich, Charles E., e.a. 'Autonomous Six-Degree-of-Freedom Spacecraft Docking Maneuvers via Reinforcement Learning'. arXiv:2008.03215 [cs, eess], augustus 2020. arXiv.org, http://arxiv.org/abs/2008.03215.

[6] Oestreich, C. E., Linares, R., & Gondhalekar, R. (2021). Autonomous Six-Degree-of-Freedom Spacecraft Docking with Rotating Targets via Reinforcement Learning. Journal of Aerospace Information Systems, 1–12. https://doi.org/10.2514/1.i010914

[7] Ng A.Y. et al. (2006) Autonomous Inverted Helicopter Flight via Reinforcement Learning. In: Ang M.H., Khatib O. (eds) Experimental Robotics IX. Springer Tracts in Advanced Robotics, vol 21. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11552246_35

[8] Howell, M., & Best, M. (2000). On-line PID tuning for engine idle-speed control using continuous action reinforcement learning automata. Control Engineering Practice, 8(2), 147–154. https://doi.org/10.1016/s0967-0661(99)00141-0

[9] W. M. van Buijtenen, G. Schram, R. Babuska and H. B. Verbruggen, "Adaptive fuzzy control of satellite attitude by reinforcement learning," in IEEE Transactions on Fuzzy Systems, vol. 6, no. 2, pp. 185-194, May 1998, doi: 10.1109/91.669012

[10] Bishop, C. M. (2016). Pattern Recognition and Machine Learning. New York, USA: Springer Publishing.

[11] Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11), 1238–1274. https://doi.org/10.1177/0278364913495721

[12] Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems (pp. 1057-1063).

[13] Acikmese, B., & Ploen, S. R. (2007). Convex Programming Approach to Powered Descent Guidance for Mars Landing. Journal of Guidance, Control, and Dynamics, 30(5), 1353–1366. https://doi.org/10.2514/1.27553

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

[15] D'Souza, C., & D'Souza, C. (1997). An optimal guidance law for planetary landing. In Guidance, Navigation, and Control Conference (p. 3709).

[16] 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.

[17] Broida, J., & Linares, R. (2019). Spacecraft Rendezvous Guidance in Cluttered Environments Via Reinforcement Learning. In 29th AAS/AIAA Space Flight Mechanics Meeting January. Ka'anapali, HI: American Astronautical Society, Univelt.

[18] Izzo, D., Märtens, M. & Pan, B. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. Astrodyn 3, 287–299 (2019). https://doi.org/10.1007/s42064-018-0053-6

[19] Gaudet, B., Furfaro, R. Robust spacecraft hovering near small bodies in environments with unknown dynamics using reinforcement learning. In: Proceedings of AIAA/AAS Astrodynamics Specialist Conference, 2012, 5072.

[20] Gaudet, B., Linares, R., & Furfaro, R. (2020). Six degree-of-freedom body-fixed hovering over unmapped asteroids via LIDAR altimetry and reinforcement meta-learning. Acta Astronautica, 172, 90-99.

[21] OpenAI (2016). LunarLanderContinuous-v2. OpenAI Gym. Retrieved January 12th 2021, `https://gym.openai.com/envs/LunarLanderContinuous-v2`

[22] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning, second edition: An Introduction (Adaptive Computation and Machine Learning series) (second edition). Bradford Books.

[23] Dunlap, K., Mote, M., Delsing, K., & Hobbs, K. L. (2022). Run time assured reinforcement learning for safe satellite docking. In AIAA SCITECH 2022 Forum (p. 1853).

[24] Scorsoglio, A., D'Ambrosio, A., Ghilardi, L., Furfaro, R., Gaudet, B., Linares, R., & Curti, F. (2020, August). Safe Lunar landing via images: A Reinforcement Meta-Learning application to autonomous hazard avoidance and landing. In Proceedings of the 2020 AAS/AIAA Astrodynamics Specialist Conference, Virtual (pp. 9-12).

[25] Scorsoglio, A., D'Ambrosio, A., Ghilardi, L., Gaudet, B., Curti, F., & Furfaro, R. (2022). Image-Based Deep Reinforcement Meta-Learning for Autonomous Lunar Landing. Journal of Spacecraft and Rockets, 59(1), 153-165.

[26] Gaudet, B., & Furfaro, R. (2022). Integrated Guidance and Control for Lunar Landing using a Stabilized Seeker. In AIAA SCITECH 2022 Forum (p. 1838).

[27] G. Ciabatti, S. Daftry and R. Capobianco, "Autonomous Planetary Landing via Deep Reinforcement Learning and Transfer Learning," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2021, pp. 2031-2038, doi: 10.1109/CVPRW53098.2021.00231.

[28] Xu, X., Chen, Y., & Bai, C. (2021). Deep Reinforcement Learning-Based Accurate Control of Planetary Soft Landing. Sensors, 21(23), 8161.

[29] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009, June). Curriculum learning. In Proceedings of the 26th annual international conference on machine learning (pp. 41-48).

[30] Stable-Baselines3 team. *PPO — Stable Baselines3 Documentation*. Stable-Baselines3. `https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html`