# A Robust Control Approach for Rocket Landing

*Reuben Ferrante*


Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2017

# Abstract

The high cost of rocket technology has led to system reusability developments, particularly in the field of first stage rockets. With the motivation of decreasing production costs, successful vertical rocket landing attempts by SpaceX and Blue Origin have led the path for autonomous recovery and reusability of rocket engines. Such a feat can only be accomplished by complex control algorithms executing in real-time aboard the rocket. This project aims to develop a vertical rocket landing simulation environment where algorithms based on classical control and machine learning can be designed and evaluated.

After developing the simulated environment in Python using a robust physics engine known as Box2D, two control algorithms were designed; once classical control method and the other from the optimal control domain. The classical control Proportional Integral Derivative (PID) controller served as a benchmark, whereas Model Predictive Control (MPC) makes use of an optimizer to find the best performing action to take based on an objective function. Two Reinforcement Learning algorithms were then designed with the aim of automatically developing the required control without explicitly defining the dynamics of the system. These methods, known formally as Function Approximation Q-Learning and Deep Deterministic Policy Gradients (DDPG) provided a contrasting approach to the PID and MPC controllers. While the classical controllers achieve acceptable performance, the Artificial Intelligent counterparts, specifically the DDPG converged to a more stable and consistent rocket landing.

# Acknowledgements

I would like to thank my supervisor, Dr. Zhibin Li, for his patience, guidance, and insights throughout this project.

Without the financial help provided by the Endeavour Scholarship Scheme (Malta), this research wouldn't have been possible. I thank them for giving me this invaluable opportunity.

The research work disclosed in this publication is partially funded by the Endeavour Scholarship Scheme (Malta). Scholarships are part-financed by the European Union - European Social Fund (ESF) - Operational Programme II – Cohesion Policy 2014-2020

**"*Investing in human capital to create more opportunities and promote the well-being of society*".**

European Union – European Structural and Investment Funds
Operational Programme II – Cohesion Policy 2014-2020
*"Investing in human capital to create more opportunities and promote the well-being of society"*
Scholarships are part-financed by the European Union - European Social Funds (ESF)
Co-financing rate: 80% EU Funds;20% National Funds

# Declaration

I declare that the work presented in this thesis is my own and has been personally composed, except where it is explicitly stated otherwise in the text. This work has not been submitted for any other degree or professional qualification except as specified in the title page.

*(Reuben Ferrante)*

# Table of Contents

# 1 Introduction

This project concerns itself with the multidisciplinary subject of reusable space systems, specifically first stage rockets. Vertical take-off and landing (VTOL) of a rocket is a very primitive field which exploits the reusability of the launch vehicle used to transport highly valuable payloads, such as satellites, into space. Work on reusable launch systems is motivated by economic and material reasons; a significant cost reduction is attained by reusing the first stage rocket hardware [1], [2]. On top of this, improved scalability, as well as increased launch frequency are two positive by-products of this reusability.

Since the subject of VTOL of a rocket is still in its infancy, there is very little to no published material on specific topics, such as control methods used. Moreover, these industries are subject to sensitive military information which prevents any publications. Therefore, some references to exemplary non-technical methods and plans are made from respectable leaders in the industry who spearheaded the subject, such as Space Exploration Technologies Corporation (SpaceX).

The first successful relaunch of a previously used rocket was performed by SpaceX with the Falcon 9 rocket [3] on March 30$^{th}$, 2017. It is important to point out that not the entire rocket is reused. As an example, SpaceX's Falcon 9 rocket is made up of two main stages [4], excluding the payload that fits on top of the second stage. The first stage houses the clustered rocket engines as well as the aluminium-lithium alloy tanks, whereas the second stage contains a single engine to drive the payload to the desired orbit. After separation, the first stage is propelled back to earth in a controlled manner as shown in Figure 1 [1].

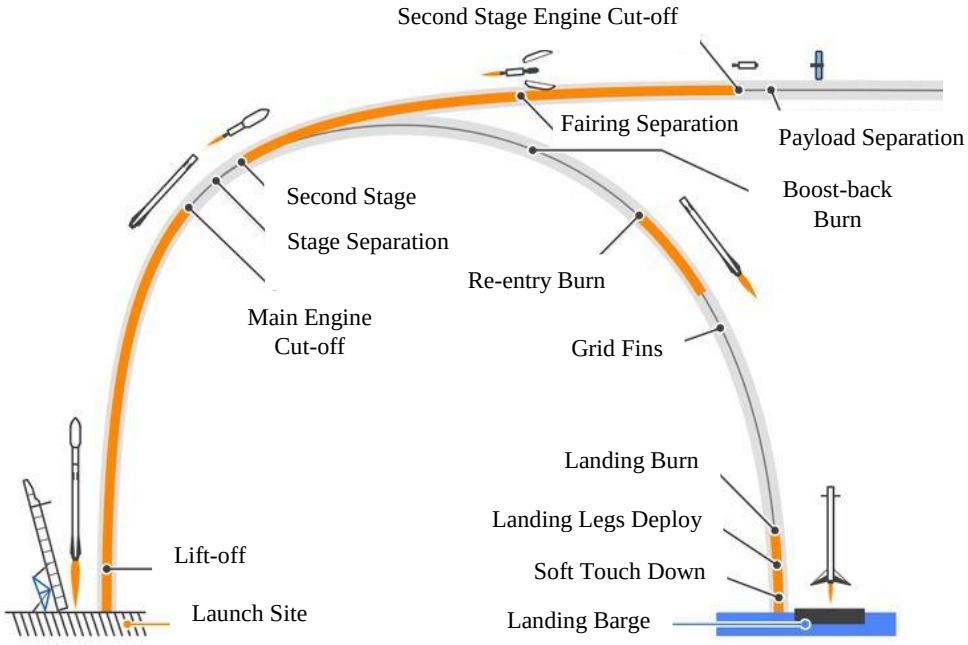



***Figure 1*** *SpaceX Falcon 9 Launch Profile* [1]

The focus of this thesis was on the very last stage: designing controllers to land the rocket on the barge with the main firing engine without exceeding any hardware limits or running out of fuel. In other words, this work assumed that all previous stages of the rocket were successful and that the rocket was guided to within reasonable proximity (≈ 200 meters) of the landing area, having a relative downward velocity similar to that experienced in real life.

Tests conducted by SpaceX and Blue Origin [5] successfully show the use of the main engine, grid fins as well as cold gas Nitrogen thrusters to land the first stage rockets, each used to a different extent and in different stages. Gimbaled thrust [6] is also used as part of the control. Together with the main and side thrusters, gimbaled thrust will form the backbone of the mathematical model of the rocket in this thesis.

## 1.1 Objectives and Contributions

The main objective of the project is to design and compare classical and optimal control algorithms with machine learning algorithms with increasing sophistication. Since designing stable closed loop controllers for non-linear and multivariable systems is non-trivial, the Artificial Intelligence (AI) approach represents an unsupervised way to tackle such complex problems. To this end, two control approaches, as well as two AI approaches were designed and implemented.

Before designing any controller, a vertical rocket simulation environment was developed in Python 3.5 using the physics engine Box2D [7]. This environment allows for not just landing simulations, but also launches and trajectory tracking. Therefore, contributions from this thesis include:

- Simulation environment in Python 3.5
- Implementation of:
    - Proportional Integral Derivative (PID) controller
    - Model Predictive Control (MPC) method on the linearized problem
    - Linear function approximation Q-Learning controller
    - Deep Deterministic Policy Gradient (DDPG) controller
- A thorough comparison between methods from both domains.

The rest of the thesis is organized as follows: Section 2 presents related work done on some of the control algorithms that are explored, as well as important rocket related details that must be addressed. Section 3 gives a brief overview of the developed simulation environment. Section 4 presents the problem as well as technical background on the algorithms that were adopted. It also describes the methodology of the models and establishes the baseline that was implemented for the rest of the models to be compared with. This is followed by the evaluation and comparison with the improvements and conclusion drawing final remarks in Sections 6 and 7 respectively.

## 1.2 Problem Definition

Consider Figure 2 below, illustrating the rocket landing on a barge.

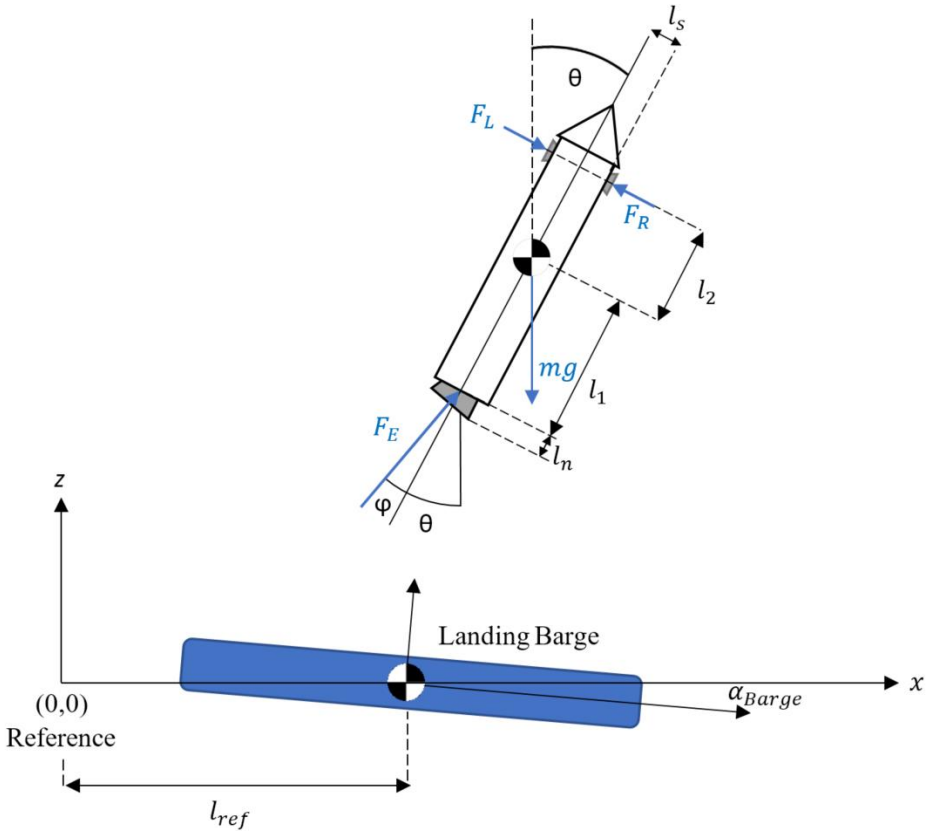



***Figure 2*** *Vertical Rocket Landing Model*

Let

$F_E = Main\ Thruster\ Force$
$F_R = Right\ Thruster\ Force$
$F_L = Left\ Thruster\ Force$
$F_S = F_L - F_R$
$\theta = Angle\ between\ the\ z-axis\ and\ the\ longitudinal\ axis\ of\ the\ rocket$
$\varphi = Angle\ between\ the\ Nozzle\ and\ the\ longitudinal\ axis\ of\ the\ rocket$
$l_1 = Londigutdinal\ length\ between\ the\ Center\ of\ Gravity\ (COG)\ and\ F_E$
$l_2 = Longitudinal\ length\ between\ the\ COG\ and\ F_R, F_L$
$l_n = Nozzle\ length$
$m = Rocket\ Dry\ Mass + Fuel\ Mass$
$x = Horizontal\ Position\ of\ the\ Rocket$
$z = Vertical\ Position\ of\ the\ Rocket$
$\alpha = Real\ Constant$

As explained in the introduction, the main controls of the rocket are the:

- Main engine thrust, $F_E$
- Side Nitrogen gas thrusters, $F_L, F_R$
    - Summarized in a single input $F_S = F_L - F_R$
- Nozzle angle, φ.

The objective is to land the rocket in a controlled manner such that the *final* state of the rocket at landing is as close to a target state as possible. This will later be defined numerically in a utility function. The inputs, defined as $u = [F_E, F_S, \varphi]$ have the following constraints:

$$\begin{aligned} 0\ N \leq F_E \leq 6486\ N \\ -130\ N \leq F_S \leq 130\ N \\ -15^o \leq \varphi \leq 15^o \end{aligned}$$

These constraints are *scaled* (1:30) estimates for the first stage rocket of Falcon 9 during landing [4], [8]. Except for the *position* of the center of gravity (COG), the simulation was developed to reflect real life conditions where possible, including dimensions and force magnitudes. In the simulation, the COG was higher than in real rockets, making control harder.

Let the state of the rocket dynamics at any time be defined by $\boldsymbol{x_i} = [x_i, \dot{x}_i, z_i, \dot{z}_i, \theta_i, \dot{\theta}_i]$ and the final state by $\boldsymbol{x_\tau}$. For a successful landing, $\boldsymbol{x_\tau}$ must be within the following numerical thresholds, defined by $\boldsymbol{x_{\tau_{max}}}$:

$$\begin{aligned} -Left\ Barge\ Edge \leq x_\tau \leq Right\ Barge\ Edge \\ -2\ m/s \leq \dot{x}_\tau \leq 2\ m/s \\ z_\tau = Barge\ Height \\ \dot{z}_\tau = 0\ m/s \\ -10^o \leq \theta_\tau \leq 10^o \\ -2^o/s \leq \dot{\theta}_\tau \leq 2^o/s \end{aligned}$$

Therefore, a cost can be imposed on the *final* state, defined as:

$$J = \omega^T \left(x_{target} - x^+\right)$$

where $\omega$ is a $6 \times 1$ weighting matrix and $x_{target}$ is the desired final state. This formally defines a successful landing, such that $J$ lies within a threshold. The closer $J$ is to 0, the better the landing. The cost associated with the states will be formally defined in Section 5.2.

# 2 Background

Vertical rocket landing has only been practically explored in the last few years. Given that this is a multidisciplinary topic, a brief background on thrust vectoring, different aspects of the rocket, the barge, and control methods is necessary, with special attention given to the latter.

## 2.1 Launch Vehicle Reuse

The motivations of launch vehicle re-use are two-fold: saving of the first stage engine and structure leading to significant economic savings. However, there can be different types of recovery systems, dependent on the type of launch vehicle. Ragab et al. [9] review the different techniques used. They highlight that propellant and gases contribute less than 5% of the first stage cost of the rocket, further cementing the argument for recovery. However, they do mention that simpler recovery, such as with the use of parachutes, is more cost effective than booster fly back. At the same time, the landing accuracy of simpler methods is measured in miles, whereas with vertical rocket landing it is measured in meters.

After separation of the first and second stages, reduction of translational velocity is necessary. Blue Origin achieves this both passively with brake fins, as well as actively by re-starting the main engine, powered by liquid Hydrogen and liquid Oxygen [5]. On the other hand, Falcon 9 uses its grid fins for re-entry manoeuvrability before switching on its engines again [4]. It also achieves controllability using main engine gimballing and cold gas Nitrogen thrusters [8]. The latter two are included in this project's mathematical model as illustrated in Figure 2.

## 2.2 Thrust Vector Control

Thrust vectoring will be the main control method to keep $\theta$ as close to $0^o$ as possible, keeping the rocket upright whilst still following a reference trajectory. Vectoring refers to the gimballing action of the engine or the flexibility of the nozzle, where the nozzle direction is changed relative to the COG of the rocket. Since the direction of the nozzle dictates the angle at which thrust is exerted, a torque about the COG is created if $\varphi \neq 0$ as shown in Figure 3 [10]. In reality, the nozzle is moved along 3-dimensions with actuators. Since the simulation developed in this thesis is in 2-dimensions, only a single rotational movement is needed.

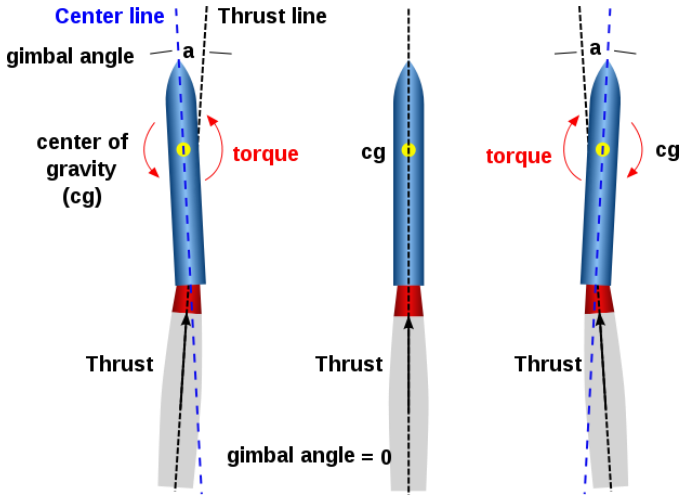



***Figure 3*** *Thrust Vector Control of a Rocket [10]*

The nozzle itself can take many forms, and the generated thrust magnitude and profile are directly dependent on the shape of the nozzle. Thrust reduction and increased wake turbulence can result from a sub-optimal nozzle profile [6], [11], however, this project will assume ideal thrust profiles utilizing a single flexible nozzle joint [9]. Hence, the gimbal can be represented by a rotary ball joint at the lower end of the rocket. The flow of the thrust will be assumed to act along a single directional vector, $F_E$.

## 2.3 The Landing Site

Simple recovery methods with parachutes were used to collect launch vehicles from the ocean, however, no first stage rocket had landed on an ocean barge before. Falcon 9 successfully did this on April 18$^{th}$, 2014 [12] in a historic landing.

The rocket's take-off location is usually from the east coast of the United States, having an abundance of area away from civilization. However, if the first stage is to be recovered, not enough fuel can be carried by the rocket for the first stage to make it back to land. Even then, it would be a dangerous endeavour. For this reason, SpaceX opted to use a floating platform in the Pacific Ocean.

The barge's landing area measures approximately 74 by 52 meters and is navigated with 4 diesel-powered thrusters and a Global Positioning System for self-navigation [13]. Given this setting, landing a rocket on a self-navigated barge presents a control problem on its own, especially since sudden weather changes can cause disturbances in the barge's position and angle. For this reason, the floating platform's position and angle relative to the horizontal plane are variables in the simulation and can be adjusted for testing purposes and included in the mathematical model.

## 2.4 Control Algorithms

Closed-loop control systems [14] are the pivot on which such landings are made possible. Both classical control and Artificial Intelligence (AI) techniques rely on state variables to analyze the error with respect to an ideal state and execute corrective measures. A controller's job is to perform these actions in a stable and controlled manner.

In classical control, these states must be bound by a well-defined mathematical model, whereas in AI input variables are defined loosely since the method has no knowledge of what the variables represent. This underlines the difference in the approaches of creating a controller. Classical control follows a set of rules and known methodology that have been widely used and tested, whereas AI techniques, such as Reinforcement Learning (RL), are less structured.

This section introduces the background required for understanding the material from both domains.

### 2.4.1 System Representation

The design of classical controllers requires a model to be defined in a certain standardized format. In control theory, functions known as *transfer functions* are used to characterize the input-output relationships of the system defined by differential equations [15].

Consider a linear time-invariant system defined by the following differential equation:

$$a_o \frac{d^n y}{dt^n} + a_1 \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_n y = b_o \frac{d^m x}{dt^m} + b_1 \frac{d^{m-1} x}{dt^{m-1}} + \cdots + b_m x$$

Where $n \geq m$, $y$ is the output of the system, $x$ is the input and $a_{o\ldots n}$, $b_{o\ldots m}$ are their respective coefficients. The transfer function of the system is defined as the ratio of the Laplace transform of the output to the input under zero initial conditions. Formally:

$$\begin{aligned} G(s) &= \left. \frac{\mathcal{L}[output]}{\mathcal{L}[input]} \right|_{zero\ initial\ conditions} \\ &= \frac{Y(s)}{X(s)} \\ &= \frac{b_o s^m + b_1 s^{m-1} + \cdots + b_m}{a_o s^n + a_1 s^{n-1} + \cdots + a_n} \end{aligned}$$

The Laplace transform is a convenient operation that allows the input-output relationship to be represented algebraically. A typical closed loop Single Input Single Output (SISO) system can be represented as shown in Figure 4.

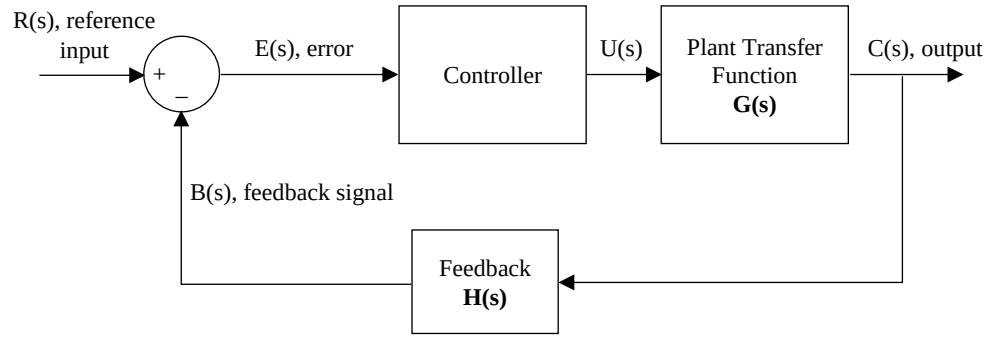



***Figure 4*** *Single Input Single Output Closed-Loop System*

As complexity increases, systems typically become Multiple Input Multiple Output (MIMO) problems, at which point such a representation becomes too limiting. Therefore, the linearized mathematical model in this project is represented in state space form [16].

The state vector was previously defined as $x$ whilst the input vector was defined as $u$. Using state space, the dynamics of a *linear* system can be calculated for any time $t \geq t_o$. The system can be defined by:

$$States: x = [x_1, x_2, \dots, x_n]$$
$$Output: y = [y_1, y_2, \dots, y_m]$$
$$Input: u = [u_1, \dots, u_r]$$

$$\dot{x}(t) = f(x, u, t)$$
$$y(t) = g(x, u, t)$$

If the system is non-linear, as is the rocket VTOL, then it must be linearized about an operating point using Taylor series expansion [17]. Otherwise, it can be written directly in the following shorthand notation:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Where $\boldsymbol{A}$ is the state matrix, $\boldsymbol{B}$ is the input matrix, $\boldsymbol{C}$ is the output matrix and $\boldsymbol{D}$ is the direct transmission matrix, which is usually $\mathbf{0}$. State space allows for a concise representation of a system, exposing all the dynamic relationships in well-defined matrices. Moreover, whereas Figure 4 models a SISO system, state feedback can be used by designing a $K$ matrix that together with a reference state commands the plant

through its input as shown in Figure 5. The SISO model in Figure 4 was used to implement 3 decoupled PIDs to control $u$ in the benchmark model and is outlined in Section 4.2. State feedback was used for more advanced optimal control as shown in Section 4.3.

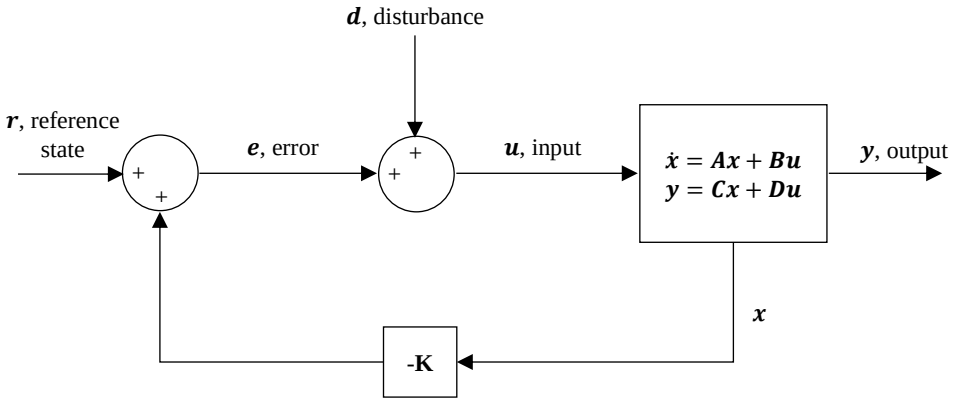



***Figure 5*** *State feedback system. The state vector* ***x*** *is used together with the reference state to control the plant. The disturbance,* ***d,*** *can manifest itself at any point in the process but is shown in the feedforward loop.*

### 2.4.2 Proportional Integral Derivative Controller

A Proportional Integral Derivative (PID) controller is an intuitive controller that is suitable for SISO systems shown in Figure 4. The PID is a simple yet effective controller that computes the proportional, integral and derivative of the difference between the output and the reference input (error) and outputs a control signal depending on the defined PID coefficients [18]. The input-output relationship is defined as:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

Taking the Laplace transform leads to:

$$\frac{E(s)}{U(s)} = \frac{s}{K_d s^2 + K_p s + K_i}$$

Fine-tuning of $K_p, K_i$ and $K_d$ has been refined in the continuous time domain, frequency domain, and even discrete time domain. These parameters are designed based on the desired response to a step input. The response is characterised by the rise time, settling time, bandwidth and overshoot. Well-known methods for tuning are the Ziegler-Nichols rules [19] and root locus [20]. A system may require certain timing and damping characteristics which impose design criteria during the design

process. As an example, it is imperative that a chemical process does not experience any overshoots, but must be critically damped.

A second order system is typically used as an approximation for many systems because of its balance between complexity and ease of design. The following transfer function represents a *closed-loop* second order system:

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2}$$

Where $\omega_n$ is the natural frequency of the system and $\zeta$ is the damping factor. Yang et al. [21] designed a second order PID controller calibrated by trial and error on a thrust vectored nozzle. The authors also highlight the effect that the individual PID constants have on the system, particularly stressing the need to balance the transient response with steady-state errors and oscillations.

The proportional term, $K_p$ tends to make short transients and caters for the present error as well as to decrease the steady state error. The integral term allows for the elimination of steady state errors but increases the order of the system, potentially rendering it unstable. On the other hand, $K_d$ leads to faster rise times and increases the system's bandwidth. The PID is used as a benchmark controller in this project and the design is described in detail in Section 4.2.

### 2.4.3 Optimal Control

Even though the PID is widely used to control simple systems, it does not guarantee optimal control or stability. Furthermore, problems with coupled variables and MIMO systems increase the complexity and make the manual-tuning of a PID a naïve approach. To this end, the field of optimal control is introduced. The Linear Quadratic Regulator (LQR) and MPC are two such controllers in this field. In this project, LQR was used as a stepping stone for MPC.

As opposed to manual tuning of constants, optimal controllers minimize a cost function with constraints associated with the state and input in order to iteratively compute the optimal control strategy. A general optimization problem takes the following form [22]:

$$\begin{aligned} &minimize && f(x) \\ &subject\ to && f_i(x) \leq 0, i = 1, \dots, m \\ & && g_i(x) = 0, i = 1, \dots, p \end{aligned}$$

Where $f(x)$ represents the objective function and the constraints represent inequality and equality constraints respectively. This is a convex program if the objective function and constraints are both convex, satisfying the inequality:

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$$

Not all problems are solved equally; different classes of optimization problems, such as least squares, linear programs or quadratic programs may use different optimizers to obtain a solution using the least amount of computing power. Moreover, non-convex problems require more time to ignore local optima and reach a global solution [22].

### 2.4.4 Linear Quadratic Regulator

For a model given by $\dot{x} = Ax + Bu$, the LQR seeks to find a feedback matrix $-K$ that leads to optimal control by minimizing a quadratic cost defined as $J = \int_0^\tau (x^T Q x + u^T R u) dt + x_\tau^T Q_\tau x_\tau$ [23] where $Q \succcurlyeq 0, Q_\tau \succcurlyeq 0$ and $R \succ 0$ (positive semi-definite and positive definite respectively). The symmetric matrix $Q_\tau$ represents the cost given to the final state.

$Q$ represents the penalty given to the distance between the state and target, whilst $R$ is the penalty paid to execute the actions. Therefore, $J$ represents a trade-off between state accuracy and action penalty [24]. Low values of $R$ indicate that the controller has more flexibility in executing the actions. Moreover, $Q$ and $R$ must be balanced to achieve the required transient response as well as steady state error, a process requiring a certain degree of trial and error [23].

Note that this method requires a *linear* system. This suggests that the matrix $K$ can be found analytically to achieve the feedback control law $u = -Kx$. In fact, there are many ways to derive $K$, most notably the dynamic programming approach. By differentiating $J$ with respect to $u$ over a single time step, the following solution is derived for the *continuous time case*:

$$u = -R^{-1} B^T P_t x$$

Where $P_t$ is the solution of the algebraic Riccati equation [23]. It can be shown that this solution achieves desirable control characteristics and robustness [25]. Note that a distinction must be made between continuous and discrete time LQR, since they lead to slightly different solutions.

Mohammadbagheri et al. [18] compare PID and LQR controllers on voltage-source inverters and concluded that the rise time as well as settling time of LQR controllers were superior to those of the simple PID. Kumar et al. [26] also design robust LQR controllers for both stabilizing the inverted pendulum and trajectory tracking to a

reference input. Note that the inverted pendulum model is not that far from the rocket landing problem, on the contrary, the mathematical models are very similar since both tend to naturally unstable equilibrium positions. The authors showed that LQR can optimize even with the most stringent of parameters.

The disadvantage of LQR is that the optimal feedback is independent of constraints. This poses a problem in processes such as the rocket landing, where all inputs are bounded. The problem can be redefined to include constraints, but more advanced methods, such as MPC, cater for such an issue.

### 2.4.5 Model Predictive Control

MPC is a relatively new field in control that has only been proven useful thanks to the increased computational power. However, it found widespread use in industry; from precision landings [27] to trajectory planning in missiles [28]. Like LQR, MPC solves a quadratic program by minimizing an objective function. However, unlike LQR, it includes equality and inequality constraints, and the *linearized* plant dynamics form part of these constraints. This enables the optimizer to find a solution that is optimal for not just the present state, but also future states. The extent of predictability is referred to as the *time horizon.* A generalized convex quadratic MPC program takes the following form [29]:

$$
\begin{aligned}
&minimize && J = \sum_{t=t_i}^{t_i+T_h} (x_t^T\, Q x_t + u_t^T R u_t) \\
&subject\ to && u_t \in \mathrm{U}, x_t \in \mathrm{X} \\
& && x_{t+1} = Ax_t + Bu_t \\
& && x_{t_i+T_h} = x_{target} \\
& && x_{t_i} = x_{initial} \\
& && for\ t = t_i \ldots T_h
\end{aligned}
$$

$x_{initial}$ represents the initial state and $x_{target}$ represents the target state at the end of the time horizon $T_h$. This constrains the problem to converge to a desired final state. Note that the problem can be restructured in many ways, and the choice of the utility function, cost matrices, constraints, linearization, control horizon, prediction horizon, sampling interval and error tolerance should be treated as hyper parameters. As an example, the penalization of the control action $u_t^T R u_t$ in $J$ causes the objective function to have a non-zero value even under steady state conditions, where $u_t$ might not be zero for proper control action. Meadows et al. [30] propose to penalize the change in actions, $\Delta u$, instead. Garcia et al. [31] noted that a large sampling interval causes oscillations and suggest that only the present resulting action $u_{t_i}$ is used, and to resolve the problem again at $(t_i + 1)$ using the new state.

Meadows et al. [30] reported increased overshoot and faster response with longer time horizons. Moreover, longer prediction horizons are more susceptible and sensitive to disturbances.

Bryson et al. [32] suggest that weight values should be inversely proportional to the maximum limit. On a contrasting note, Meadows et al. [30] found this to be too penalizing and instead suggest to apply a penalty to constraint deviations. This would be in accordance with general optimization theory, where instead of fixing equality constraints and introducing more variables, a slack variable is added instead [33].

$$
\begin{aligned}
&minimize && J = \sum_{t=t_i}^{t_i+T_h} (x_t^T\, Q x_t + \Delta u_t^T R \Delta u_t + \omega^T S \omega) \\
&subject\ to && u_t - u_{limits} \leq \omega, x_t \in \mathrm{X} \\
& && x_{t+1} = A x_t + B u_t \\
& && x_{t_i+T_h} = x_{target} \\
& && x_{t_i} = x_{initial} \\
& && for\ t = t_i \ldots T_h
\end{aligned}
$$

Where $S$ is the penalty given if the control action approaches $u_{limits}$. For the optimizer to find a meaningful solution, the control problem must be both observable and controllable [34]. Figure 6 illustrates the MPC workings, showing the forecasted state as part of the solution. It is interesting to note that MPC has been used in the design of thrust-vectored flight [35] and also guidance and control [36]. The former achieved better results than the static gain LQR, and similar performance to gain-scheduled LQR, where many different LQR solutions are applied depending on the state in which the plant is in.

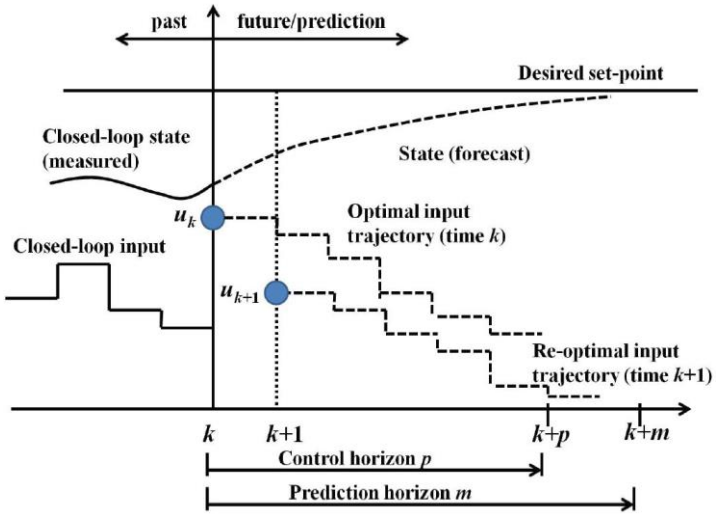



***Figure 6*** MPC takes the current state and input and finds a solution for the optimal input based on the required constraints and prediction horizon [55].

## 2.5 Reinforcement Learning

As opposed to explicitly defining a model, RL applies the theory of dynamic programming to define a framework that adapts itself through episodic interaction with an environment. The environment is typically defined as a Markov Decision Process [37], where a set of actions are available to choose from at any state. A general RL problem would involve a policy that takes actions, states that represent where the agent is, and rewards that enforce good actions when in a state. The goal is to learn the *policy*, the method of choosing actions, that maximizes rewards. Figure 7 illustrates this system.

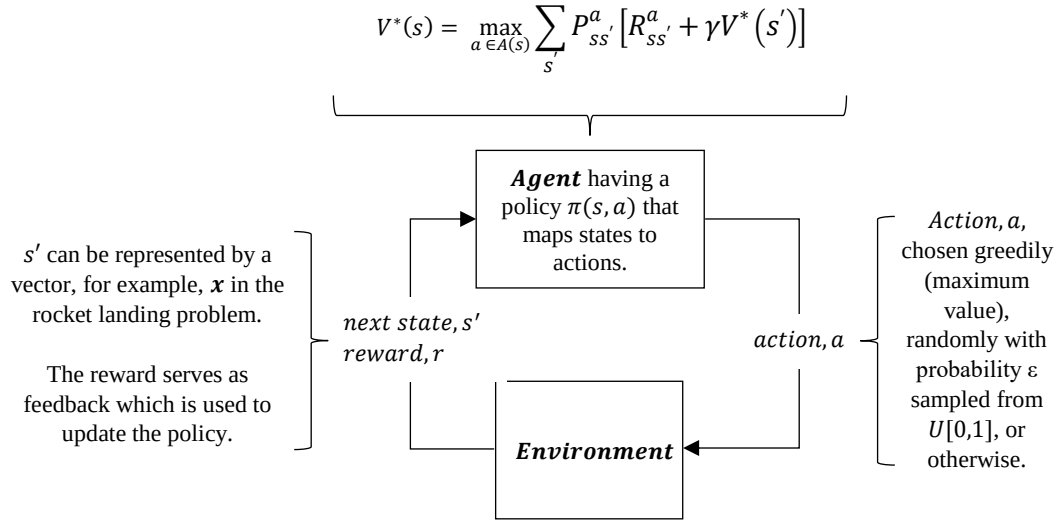



***Figure 7*** RL problem defined by a value function, which is used to determine how good it is for an agent to be in that state. An action is then taken and the environment reciprocates with a reward which is then used to update the policy.

The value function, $V(s)$, represents the expected cumulative reward that a policy would get if it followed that policy from there onwards. Formally, let:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$
$$P^a_{ss'} = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$$
$$R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

In linguistic terms, given action $a$ and state $s$, $P^a_{ss'}$ defines the transition probabilities of going from one state to each possible next state whilst $R^a_{ss'}$ defines the expected value of the next reward [38]. $R_t$ represents the discounted future rewards, where $0 \leq \gamma \leq 1$. Note that like MPC, the aim in RL is to maximize the reward and reach an optimal solution by breaking down a multiperiod problem into smaller pieces.

This notion is known as Dynamic Programming and it's central to Bellman's equation [39]:

$$\begin{aligned} V(s) &= E_\pi\{R_t|s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma \textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|s_t = s\} \\ &= \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V(s')\right] \end{aligned}$$

Where $\pi(s,a)$ is the probability of choosing action $a$ when in state $s$. Although theoretically sound, for an update of the value function to take place, all states must be visited. Instead, an iterative version known as Temporal Differencing (TD) [40] makes an online update of the form:

$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha[Target - V(s_t)] \\ &\leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \end{aligned}$$

This equation provides an updated estimate of $V(s)$ as soon as the state is visited. It only has one hyper parameter, α (learning rate), and can also be expanded to include additional terms that reinforce not just the current state, but previous states that led to a reward. This is known as TD(λ) [40].

### 2.5.1 Discrete State-Action Q-Learning

Bellman's equation above makes use of transition probabilities. However, what if these were unknown? Transition probabilities imply knowledge of a model, and the aim is to obtain a model-free framework that can learn iteratively. $V(s)$ is a notation for a value of a state of the *optimal* policy that maximizes the expected reward. On the other hand, $Q(s,a)$ represents the value of a state if action $a$ is chosen, and then continue with that policy from that state onwards. This means that it eliminates the need to know and execute the optimal policy, $\pi^*(s,a)$, and instead, execute any action available according to the policy at that time step. Formally, this can be written as:

$$Q(s_t, a_t) = r_t + \gamma E_{s_{t+1}}[V(s_{t+1})]$$

This form eliminates the need for a model and can be adapted to many problems.

Sutton et al. [41] give an example with a cliff walking grid world having discrete actions $u \in \{up, down, left, right\}$. The environment awards a reward of -1 for all transitions except for the cliff region, in which a reward of -100 is given as shown in Figure 8. They compare off-policy with on-policy Q-Learning, where the latter is known as State Action Reward State Action (SARSA). The only difference between these two algorithms is in the degree of greediness, as illustrated overleaf.

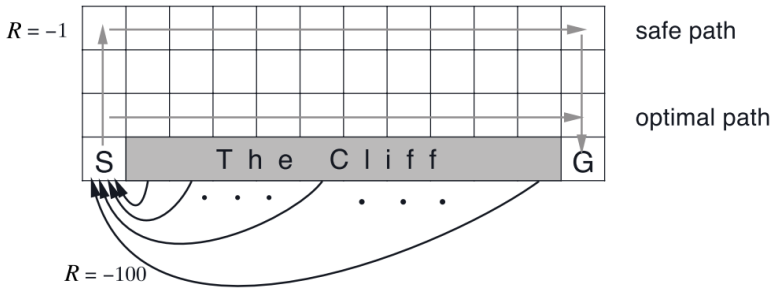



***Figure 8*** Cliff walk example [41], where the aim is to arrive to point G without falling off the cliff. A reward of -1 is given for each transition, whereas -100 is given for walking on the cliff.

*On-Policy Q-Learning (SARSA):* $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$

*Off-Policy Q-Learning:* $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$

Sutton et al. reported that SARSA achieves higher rewards than Q-Learning. After a few episodes, Q-Learning learns the optimal policy, that which walks along the edge of the cliff. Due to the ε-greedy action selection, sometimes this can result in falling off the cliff. Interestingly, in SARSA, the policy chooses to take the longer but safer path. The authors go on to suggest that ε be annealed for the two algorithms to converge.

Notice that this example represents discrete states with discrete actions. The state is left to the designer's choice. Although not specified, in the Cliff Walk example this can be represented by the grid in which the agent is currently in, for instance representing each square with a binary number. This leads to the tabular representation shown in Figure 9.

States accesed using binary representation

| | | **States** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **…** | **n-1** | **n** |
| **Actions** | Left | 0 | 0 | | | | 0 |
| | Right | 0.1 | 0.1 | … | … | … | 0.1 |
| | Up | 0 | 0 | | Q(s, a=up) | | 0 |
| | Down | 0 | 0 | | Q(s, a=down) | | 0 |

***Figure 9*** Tabular (2D array) representation of the state-action function, Q(s,a) with optimistic initialisation of the action Right. This initialization will lead to action Right being picked more frequently initially, contributing to faster learning.

The problem with such a formulation is known as the curse of dimensionality [42], where the number of states or actions make the problem infeasible. Such is the case in this project, where the actions $u \in \{F_E, F_S, \varphi\}$ are all continuous. One limiting solution is to discretize both states and actions, however, this is not scalable. To this

end, Lillicrap et al. [43] formulated a continuous framework based on deep learning which was implemented in this project.

## 2.6 Conclusion

In this section, theory and examples related to the methods implemented in this project were introduced. The VTOL problem relies on controllers, and each introduced algorithm solves a shortfall presented in the preceding ones.

PID, LQR, MPC, and RL were the four control methods that were discussed. The PID is the simplest and most intuitive method, using the proportional, integral and derivative of the error with respect to a reference in order to drive the process. However, the PID is suitable for SISO systems. Therefore, where multivariable, coupled and non-linear systems are presented, the PID can only be implemented to a limited extent on the linearized and decoupled plant. This leads to inaccuracies and sub-optimal control. PIDs are tackled as a baseline in Section 4.2.

Unlike PID controllers, LQRs use the state space to find the optimal feedback matrix $-K$. Moreover, whereas the PID acts on the error, the LQR acts on the state. This state is still evaluated with respect to an equilibrium point since any non-linearities need to be linearized for an LQR to be designed. The general form of LQR does not include constraints and the designed LQR will only be optimal at the linearized state.

The shortfall of LQR in dealing with constraints as well as different conditions led to MPC, where the problem is formed as an objective function with constraints. MPC finds the correct input to take by minimizing a cost function with respect to the states and actions. Like LQR, the cost function is quadratic, but it is not solved analytically. Instead, an optimizer is used to find a solution to the objective function. This is also done whilst respecting constraints and following a trajectory. This means that whereas PIDs and LQRs only consider the current state, MPC can incorporate the model in the constraints and simulate future states; enabling it to pick the best actions that maximize not just the current reward, but also future rewards. The derivation of LQR that led to the implementation of MPC is detailed in Section 4.3.

Recall that the general form of the MPC still needs a well-defined model where the state is found analytically or is estimated. The notion of rewards is further extended to RL, where the MPC framework is reformulated in an iterative and *model-free* framework that interacts with the environment. Feedback is given as a reward value as opposed to an error. Like MPC, it is derived from Bellman's equations and seeks to find the best policy that maximises future rewards. RL can take on many forms, but discrete state-actions prove to be limiting in control applications. To this end, a state-action approximator as well as a continuous-domain and relatively new method is applied to the VTOL problem and are discussed in Sections 4.4 and 4.5.

# 3 Simulation Environment

The environment is built in Python 3.5 and depends on the use of Box2D [7], [44]. Box2D is a physics engine that supports rigid body simulations. It was originally intended for games, however, it can also be used for light simulations not requiring state of the art accuracy used in critical applications.

Bodies in the environment are built using basic shapes and objects such as polygons, rigid bodies (defined as being as hard as diamond), fixtures (having attributes such as density, friction), physical constraints, joints, joint motors (specifying torque), and finally the world which contains all the defined objects.

Box2D is tuned for meters, kilograms, seconds and radians. Since the rocket is modeled after Falcon 9 [4] and the dimensions are quite large, all units are divided by a scaling factor (1:30) as suggested by the author of Box2D. The physics engine is meant to take care of collisions and physics related calculations with a user-defined frequency. In this case, each time-step was set to be 1/60 s, or updating with a frequency of 60Hz. Upon passing an action to the simulation as a $1\times3$ array $[F_E, F_S, \varphi]$, corresponding forces are applied to the respective body. As an example, a force equivalent to $F_E = 1$ would result in a force $F = F_E.MAIN\_ENGINE\_POWER$ applied to the bottom of the nozzle. $F_E$ and $F_S$ values are normalized, whereas the angle $\varphi$ is not.

Two types of bodies are defined in Box2D; static and dynamic. The former is meant to be indestructible, defined as having zero mass. On the other hand, dynamic bodies are meant for collisions, forces and general dynamics. All rocket parts are defined to be dynamic bodies, whereas the barge and sea are static. The nozzle is fixed with the lower part of the main rocket body by a revolute joint. The revolute joint is given a motor with a specified torque, having a certain delay. In this case, the torque was defined large enough such that the angle is driven in real-time with little to no delay.

The legs are also defined as dynamic bodies and are connected via a revolute joint with the rocket. Angle constraints of $\pm5^o$ and a torque were given to the joints to simulate a spring once in contact with the ground. Forces are applied at $90^o$ with respect to the defined body at the relative coordinates. Since visuals need to represent the physics simulation and body coordinates, all simulation dynamics are updated on the actual defined objects in the physics engine. As an example, the $(x, y)$ position of the rocket can be accessed with: $lander.position$ and this is then used to render the rocket. Particles are used to represent forces. This is done for visual aid and is very useful for verifying controls visually. However, rendering is not required for the physics simulation to take place. Finally, the environment is meant to reset to the defined initial conditions if the rocket tilts by more than $\pm35^o$ with respect to the $z$ axis or touches the outside boundary. All the above is summarized in Figure 10 and Figure 11.

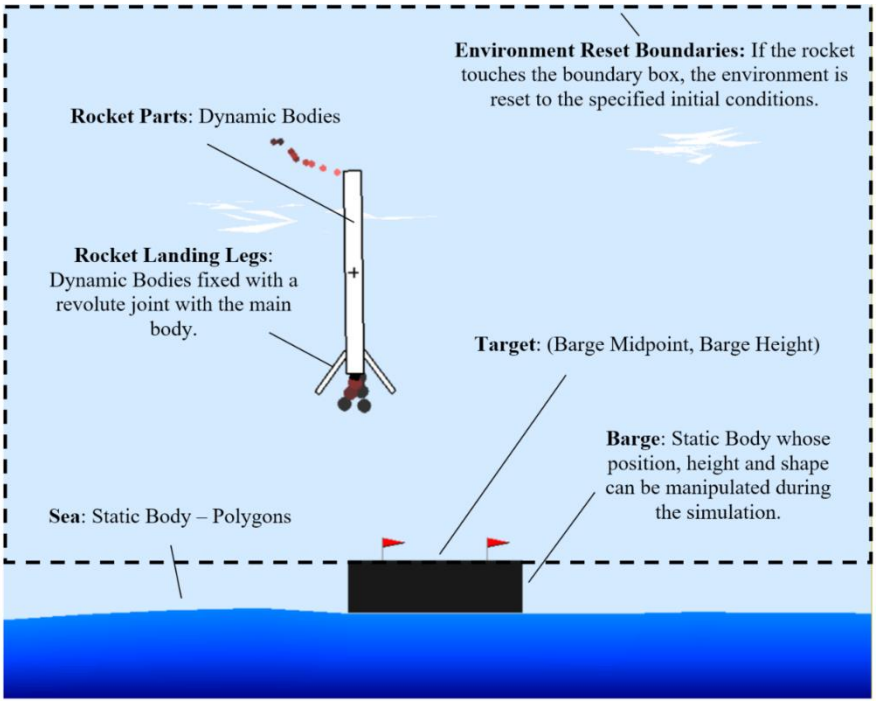



***Figure 10*** Actual simulation executing a control algorithm. Visuals are needed to correctly verify that the simulation is being executed as intended.

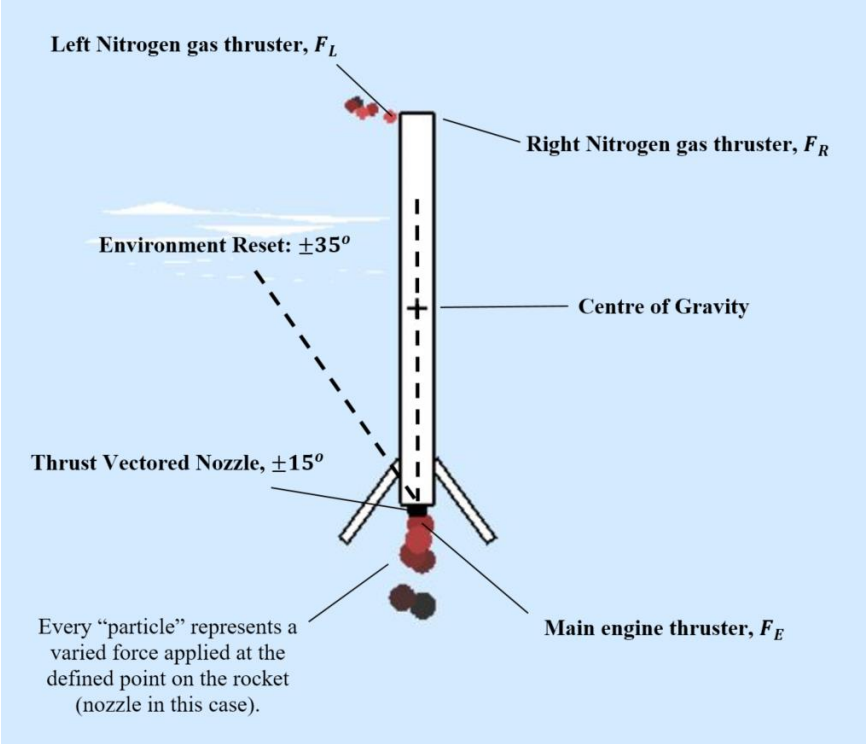



***Figure 11*** Rocket defined from multiple dynamic bodies. Particles are used as visual aid to represent forces being applied.

# 4 Methodology

This section includes the implemented algorithms with detailed derivations and methods. Reference is frequently made to theory outlined in the Background section. A brief but necessary mathematical derivation is first presented. The baseline PID controller design is then explained, followed by LQR theory which leads to MPC. Model free RL is then introduced, starting with discrete action function approximation Q-Learning, followed by the final model using DDPG, which tackles the vertical rocket landing problem in a continuous manner.

## 4.1 Mathematical Derivation

The problem and nomenclature were introduced under Problem Definition in Section 1.2, specifically Figure 2. Support legs were omitted from the diagram for clarification purposes.

The Nitrogen gas thrusters allow for more complex but stable control. However, their force can be fixed to 0 if required, simplifying the model. On the same line, the angle and position of the barge were fixed in this project, but are included as variables in the environment to reflect the possible changes that might arise.

By using Newton's 3rd law of motion, the free-body diagram shown in Figure 12 can be deduced:

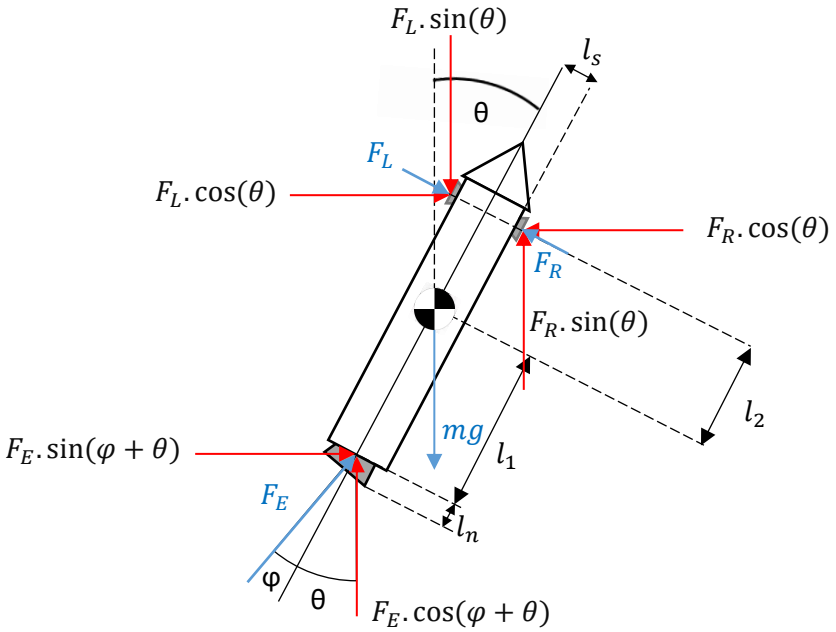



***Figure 12*** Free body diagram showing all forces considered.

Solving for translational forces in $x, z$ directions as well as rotational torque with respect to the rocket's COG leads to:

$$\begin{aligned} m\ddot{x} &= F_E \sin(\theta + \varphi) + F_s \cos(\theta) \\ \ddot{x} &= \frac{F_E \cos(\varphi) \sin(\theta) + F_E \cos(\theta) \sin(\varphi) + F_s \cos(\theta)}{m} \\ \ddot{x} &= \frac{F_E \theta + F_E \varphi + F_s}{m} \end{aligned} \tag{1}$$

In the last step, small angles were assumed for simplification. Thus, $\cos(\varphi) = \cos(\theta) \approx 1$ and $\sin(\varphi) \approx \varphi, \sin(\theta) \approx \theta$.

$$\begin{aligned} m\ddot{z} &= F_E \cos(\theta + \varphi) - F_s \sin(\theta) - mg \\ \ddot{z} &= \frac{F_E \cos(\varphi) \cos(\theta) - F_E \sin(\varphi) \sin(\theta) - F_s \sin(\theta) - mg}{m} \\ \ddot{z} &= \frac{F_E - F_E \varphi\theta - F_S \theta - mg}{m} \end{aligned} \tag{2}$$

Torque:

$$\begin{aligned} J\ddot{\theta} &= -F_E \sin(\varphi) (l_1 + l_n \cos(\varphi)) + l_2 F_S \\ \ddot{\theta} &= \frac{-F_E \varphi (l_1 + l_n) + l_2 F_S}{J} \end{aligned} \tag{3}$$

$l_n \cos(\varphi)$ represents the fact that the thrust is applied to the base of the nozzle.

$$\dot{m} = -\alpha(\beta F_E - F_s) \tag{4}$$

$$J_T \ddot{\varphi} = \tau \tag{5}$$

Equation 4 specifies that the fuel burn is directly proportional to thrust. The control problem is a multiple input, multiple output (MIMO) system where $F_E, F_s \; and \; \varphi$ represent the variable inputs that must be adjusted for the rocket to land safely at a reference position. The independent forces applied by the cold gas thrusters at the upper half of the rocket can be implied from $F_s$.

$\theta, x \; and \; z$ represent the output of the system. A suitable representation for writing a MIMO problem is state space form:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

Where

$$x = n\ state\ vector$$
$$y = n\ output\ vector$$
$$u = r\ input\ vector$$
$$A, B, C, D = Constant\ matrices$$

This formulation requires that the problem be linear time-invariant, which also implies that $\sin(\theta) \cong \theta, \cos(\theta) \cong 1$ for small angles of $\theta$. Even if this assumption is carried on, Equations 1-3 suggest that the inputs and states are coupled and not independent. For instance, varying $F_E$ leads to changes in all states. This coupling is not trivial and presents a difficult problem in control as the system needs to be linear for it to be represented in state space form.

To this end, the problem can be linearized about equilibrium points if the system operates around those points. A point is called an equilibrium point if there exists a specific input that renders all changing states to 0 [45]. Formally, for a non-linear differential equation given by:

$$\dot{x}(t) = f(x(t), u(t))$$

An equilibrium point is defined as:

$$f(\bar{x}, \bar{u}) = 0$$

Setting $\ddot{x}, \ddot{z}$ and $\ddot{\theta}$ to 0 in Equations 1-3 and solving the simultaneous equations leads to the equilibrium input $\bar{u} = [mg, 0, 0]$. This means that if we start the simulation at the equilibrium point and apply $\bar{u}$, the system will remain at that point assuming no external disturbances or randomness. However, starting away from $\bar{x}$ and applying a different $\bar{u}$ leads to a deviation:

$$\delta_x(t) = x(t) - \bar{x}$$
$$\delta_u(t) = u(t) - \bar{u}$$

Then it follows that:

$$\dot{\delta}_x(t) = f(\bar{x} + \delta_x(t), \bar{u} + \delta_u(t))$$

Expanding by Taylor's theorem and ignoring high order terms leads to:

$$\begin{aligned}\dot{\delta}_x(t) &\approx \frac{\partial f}{\partial x}\Big|_{x=\bar{x}, u=\bar{u}} \delta_x(t) + \frac{\partial f}{\partial u}\Big|_{x=\bar{x}, u=\bar{u}} \delta_u(t) \\ &\approx A\delta_x(t) + B\delta_u(t)\end{aligned}$$

This is called Jacobian Linearization about the equilibrium point. A controller should work well if it is designed at this point, as long as the system operates near that point. Both LQR and MPC make use of the linearized state space, and which is discussed further in Section 4.3.1.

## 4.2 PID Controller Design

Recall that the PID is used for a SISO system. Therefore, we wish to transform a SISO in the general transfer function form:

$$G(s) = \frac{Y_{output}(s)}{X_{input}(s)}$$

Consider taking the Laplace transform of Equation 1:

$$s^2X(s) = \frac{F_E(s)\theta(s) + F_E(s)\varphi(s) + F_s(s)}{m}$$

$X(s)$ in this equation represents the position in Laplace domain, which would represent $Y_{output}(s)$ in the transfer function. The inputs are $[F_E(s), F_S(s), \varphi(s)]$, however, $X_{input}(s)$ can only represent one input at a time, hence the multi-variability and coupling.

Methods for decoupling, such as the Relative Gain Array [46] allow a system to be decoupled and treated as a SISO. However, since the PID is only used as a benchmark, 3 separate PIDs were designed by taking the following assumptions for Equations 1-3.

**Equation 1**

$$m\ddot{x} = F_E \sin(\theta + \varphi) + F_s \cos(\theta)$$

Assume that $F_E = 0$ and small angles apply. This leads to the following simplification:

$$\frac{X}{F_S} = \frac{1}{ms^2}$$

However, note that $F_S$ affects $\theta$ significantly, as is evident from Equation 3. This coupling becomes complicated since $x$ and $\theta$ may require different conflicting controls from $F_E$ and $F_S$. This leads to the presumption that $F_S$ should output a control based on the PID error of both $k_1\theta$ and $k_2x$, where $k_1$ and $k_2$ are constants. Moreover, both $F_E$ and $F_S$ should control the position $x$.

To simplify the problem, the design of the PID took place using just one input, with the other set to 0. $k_2$ was then obtained empirically, leaving the same PID constants that were designed. This method was also used to derive the control for $F_E$ and $\varphi$ and proved to work effectively.

**Equation 2**

$$m\ddot{z} = F_E \cos(\theta + \varphi) - F_s \sin(\theta) - mg$$

Using the same assumption as Equation 1, setting $F_S = 0$ and taking the Laplace transform:

$$\frac{Z}{F_E{}'} = \frac{1}{ms^2}$$

Where $F_E' = F_E - mg$, i.e. an offset for $mg$ is applied in practice after the PID is designed. Since $F_E$ also affects the states associated with $x$ (but to a lesser extent), the same empirical method proposed in Equation 1 above is used to design this PID.

**Equation 3**

$$J\ddot{\theta} = -F_E \sin(\varphi)\,(l_1 + l_n \cos(\varphi)) + l_2 F_S$$

The final input that needs to be controlled is $\varphi$. Note that $-F_E \sin(\varphi)\,(l_1 + l_n \cos(\varphi))$ is the small contribution of $F_E$ to the rotational moment. Also, notice the opposing forces given by $F_E$ and $F_S$ as explained in Equation 1. Since $\varphi$ is being controlled, all other inputs can be treated as constants, even though they are variables. This independence assumption simplifies the problem and allows the PIDs to be designed in a straightforward manner.

$$\frac{\theta}{\varphi} = -\frac{c}{Js^2}$$

### 4.2.1 Design by Root Locus

Although an integral term brings the steady state error down to 0, it may lead to instability due to the introduction of an additional zero. For this reason, the design process took place on a Proportional Derivative controller which is given by:

$$\frac{U_{control}(s)}{E_{error}(s)} = K_d\left(s + \frac{K_p}{K_d}\right)$$

Consider the design on Equation 2 where the goal is to track a reference altitude $Z(s)_{ref}$. The closed loop system can be represented as shown in Figure 13.

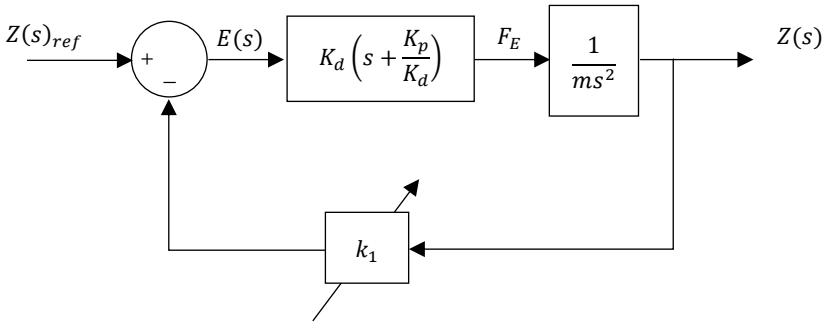



***Figure 13*** Closed loop system diagram.

The open loop poles of the *uncompensated* system $\frac{1}{ms^2}$ leads to double poles at $s = 0$, implying the system is constantly oscillating with no change in magnitude. We would like to design a controller that tracks a reference input with a certain transient behaviour and steady state error.

Let the response time be 2 seconds, implying 90% of the final steady state value should be obtained no later than 2 seconds. Also, let the damping factor, ζ, be 0.7, implying a peak overshoot of 4.5% according to the equation:

$$Peak\ Overshoot = 100.e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}$$

Using the angle and magnitude criteria and root locus [47], the following PD controller can be computed analytically:

$$9.9811(1 + 2.7s)$$

Together with the original transfer function, the compensated system traverses the locus shown in Figure 14 once a PD controller is added to the system. Notice the double pole marked in 'x' at $s = 0$. The closed loop poles occur at $-0.534 \pm 0.332j$ in accordance with the design as shown in Figure 15. In the actual algorithm, the derivative term was re-calibrated as part of the empirical tuning.

The step response for $Z$ is illustrated in Figure 16 overleaf. Note that this controller was designed in the time domain. To translate it into the digital domain, the derivative term needs to be scaled by $\frac{1}{Sample\ Time}$, which is $\frac{1}{60}$ in this case.

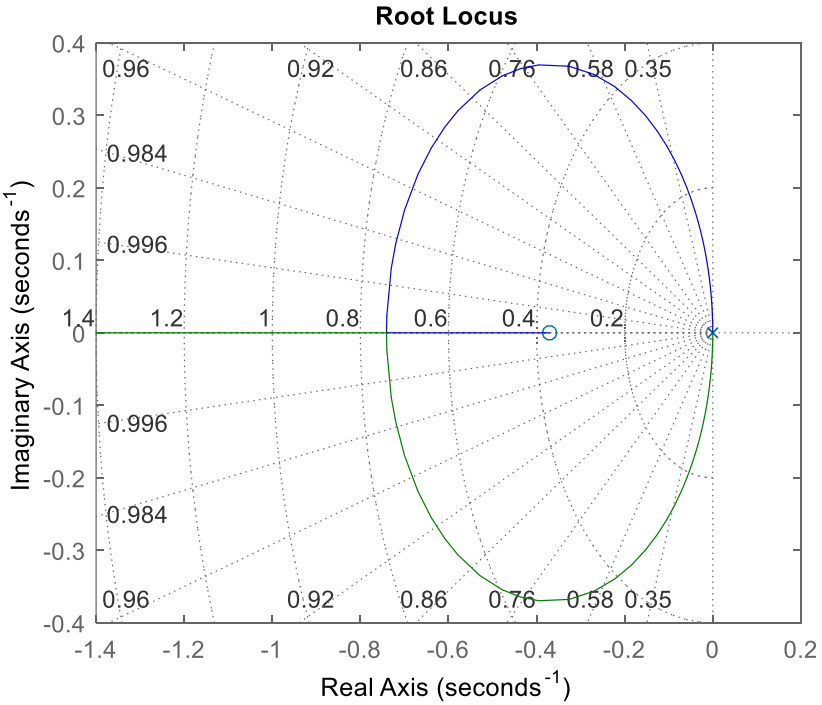



***Figure 14*** Compensated system – open loop root locus.

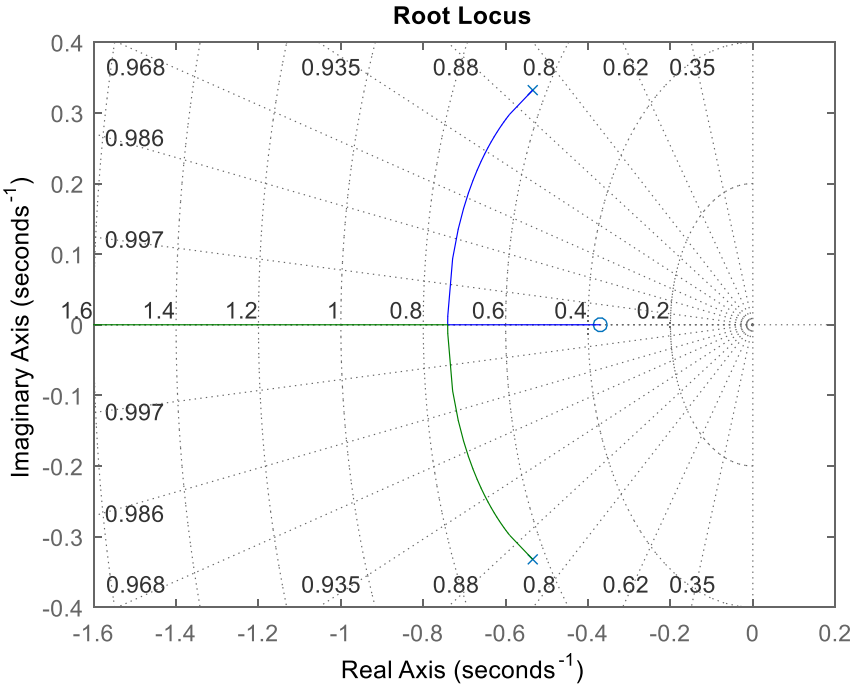



***Figure 15*** Closed loop root locus showing closed loop poles and zeros.

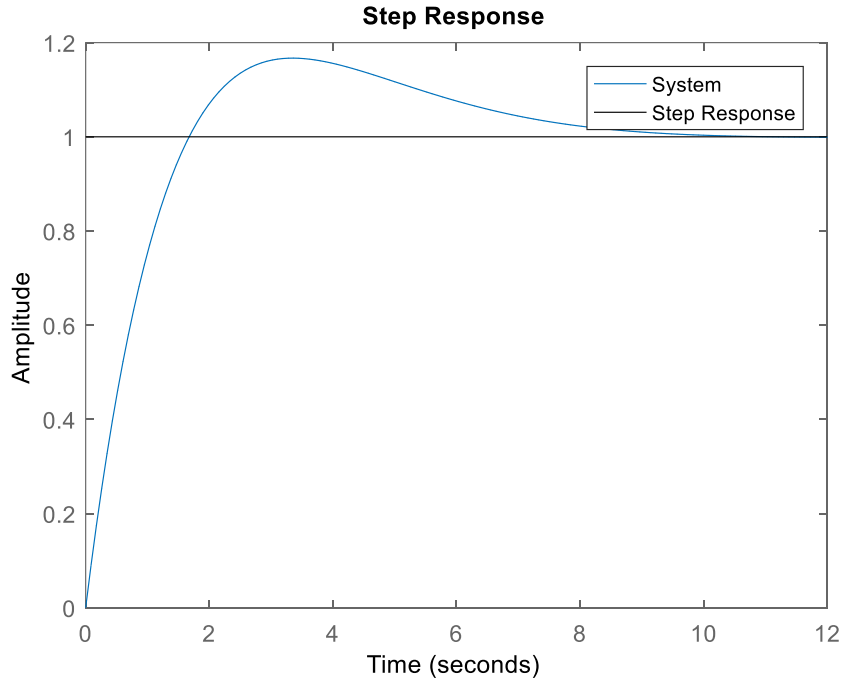



***Figure 16*** Step response of the closed loop system with the PID designed to track $z$.

Having designed the z-PID controller having output $F_E$, $k_1$ was set to 1, whereas $k_2$, the term used to represent $x$ in the error function, was found empirically during the simulation. As with any hyper parameter tuning, all other parameters of the simulation were held constant whilst finding $k_2$. This means that $\varphi$ and $F_S$ were held to 0, with no initial acceleration in the $x$ direction. This enabled the rocket to descend in a controlled manner, adjusting $k_2$ until the descent was satisfactory with respect to different $x$ initializations.

The same procedure was repeated with the other two controllers, leading to the following algorithm:

**Main Algorithm 1** PID Control

Initialize PIDs as:

$$F_{E_{PID}} = PID(K_p = 10, K_i = 0, K_d = 10)$$
$$F_{S_{PID}} = PID(K_p = 5, K_i = 0.01, K_d = 6)$$
$$\varphi_{PID} = PID(K_p = 0.085, K_i = 0.001, K_d = 10.55)$$

When called to perform control every $\frac{1}{T_s}$ seconds (default: 60Hz):

$$x, z, \dot{x}, \dot{z}, \theta, \dot{\theta}, left\ leg\ contact, right\ leg\ contact = \boldsymbol{state}$$

---

$$z_{error} = z_{ref} - z + 0.1x$$
$$\dot{z}_{error} = -\dot{z} + 0.1\dot{x}$$
$$\boldsymbol{F_E} = F_{E_{PID}}.computeOutput(z_{error}, \dot{z}_{error}) * K_{F_{E\,calibration}}$$

---

$$\theta_{error} = -\theta_{ref} + \theta + 0.2x$$
$$\dot{\theta}_{error} = \omega + 0.2\dot{x}$$
$$\boldsymbol{F_S} = F_{S_{PID}}.computeOutput(\theta_{error}, \dot{\theta}_{error})$$

---

$$\theta_{error} = -\theta_{ref} + \theta$$
$$\dot{\theta}_{error} = \omega$$
$$if\ (absolute(x) > 0.01\ and\ dy < 0.5)$$
$$\quad \theta_{error} = \theta_{error} - 0.06x$$
$$\quad \dot{\theta}_{error} = \dot{\theta}_{error} - 0.06\dot{x}$$
$$\boldsymbol{\varphi} = \varphi_{PID}.computeOutput\left(\theta_{error}, \dot{\theta}_{error}\right) * K_{\varphi\,calibration}$$

---

$$if\ (left\ leg\ contact\ or\ right\ leg\ contact)$$
$$\quad F_E = 0$$

## 4.3 MPC Controller Design

The failure of LQR to include constraints in the general case led to inputs experiencing values outside their intended limit. This was commented on in Section 2.4.4. To this end, MPC was proposed as a more sophisticated and robust numerical method to solve an objective function at every step as opposed to designing a controller at an equilibrium point. The design process is discussed in this section.

### 4.3.1 Linearization

Jacobian linearization was proposed as a viable numerical solution to approximate non-linear functions using Taylor's series expansion. Formally;

$$\begin{aligned}\dot{\delta}_x(t) &\approx \frac{\partial f}{\partial x}\Big|_{x=\bar{x},u=\bar{u}} \delta_x(t) + \frac{\partial f}{\partial u}\Big|_{x=\bar{x},u=\bar{u}} \delta_u(t) \\ &\approx A\delta_x(t) + B\delta_u(t)\end{aligned}$$

Where:

$$A = \nabla_x f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}, B = \nabla_u f = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_n} \end{bmatrix}$$

This problem can be solved in two ways; analytically or by direct computation in the simulation. Analytically, this would be equivalent to simply performing partial differentiation on the state equations $x = [x, \dot{x}, z, \dot{z}, \theta, \dot{\theta}]$ and $u = [F_E, F_S, \varphi]$ leading to the following representations:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{F_E}{m} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{-F_E\varphi - F_S}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathrm{B} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{\theta+\varphi}{m} & \frac{1}{m} & \frac{F_E}{m} \\ 0 & 0 & 0 \\ \frac{1-\varphi\theta}{m} & -\frac{\theta}{m} & -\frac{F_E\theta}{m} \\ 0 & 0 & 0 \\ -\frac{\varphi(l_1+l_n)}{J} & \frac{l_2}{J} & -\frac{F_E(l_1+l_n)}{J} \end{bmatrix}$$

Although analytically inexpensive, this method proved to be less effective than computing the Jacobians using numerical values from the simulation itself. Therefore, multi-threading was used to create $(6 + 6)$ new simulations initialized with the exact dynamics and coordinates as the current time step. A matrix $\varepsilon I$, was used to change each successive state or input by a small amount, $\varepsilon$, in either direction. Each simulation was then run through one time step and the state was returned. Finite differences were then used to create the entire Jacobian based on the actual resulting values. This had the advantage of capturing unconsidered dynamics, such as the effect of $\dot{m}$, and resulted in better performance than the analytical solution. The procedure is documented in Background Algorithm 1 below.

**Background Algorithm 1** Partial Differentiation with Finite Differences

Partial Differentiation function called, passing any *simulation settings*:

$\varepsilon = step\ change$
$\Delta_+state = state + \varepsilon I$ # This results in a 6×6 matrix
$\Delta\_state = state - \varepsilon I$

$S_+ = runSimulations(\Delta_+state, currentInput)$
$S\_ = runSimulations(\Delta\_state, currentInput)$

$A = \frac{\partial f}{\partial x} = \frac{S_+ - S\_}{2\varepsilon}$

---

$\Delta_+input = input + \varepsilon I$
$\Delta\_input = input - \varepsilon I$

$U_+ = runSimulations(\Delta_+input, state)$ # This results in a 6×3 matrix
$U\_ = runSimulations(\Delta\_input, state)$

$B = \frac{\partial f}{\partial u} = \frac{U_+ - U\_}{2\varepsilon}$

$runSimulations$ loops through the state and input matrices and executes a simulation using values from each row. This results in a total of $2N$ ($N = Number\ of\ States$) independent simulations for a total of 12, each simulating just 1 step and appending the results to matrices $A$ and $B$.

In finite differences, the smaller the value of $\varepsilon$, the more local and accurate the differentiation. A somewhat conflicting result was obtained in this case. Setting ε such that the action under test become saturated led to better results than simply incrementing by a small step change, such as $\varepsilon = 0.001$. This result can be explained by the fact that a small step change leads to a negligible change in state, resulting in $A$ and $B$ becoming almost equal to $\mathbf{0}$. This effectively leads to the optimizer not finding a solution or outputting a skewed trajectory. Testing ε with values equivalent [0.01, 0.1, 10, 50, 100] suggested that there is a range, which depends on the scale (1:30) of the simulation, in which proper trajectory planning takes place. This range was found to be between 10-100, with a value of 50 used in the simulation. This effectively saturates actions in the case of matrix $B$.

### 4.3.2 Design

Recall that a general MPC controller is given by:

$$
\begin{aligned}
&minimize && J = \sum_{t=t_i}^{t_i+T_h} (x_t^T\, Q x_t + u_t^T R u_t) \\
&subject\ to && u_t \in \mathrm{U}, x_t \in \mathrm{X} \\
& && x_{t+1} = Ax_t + Bu_t \\
& && |x_t - x_{target_t}| \leq \text{error} \\
& && x_{t_i} = x_{initial} \\
& && for\ t = t_i \ldots T_h
\end{aligned}
$$

The above problem presents the following challenges and design decisions:

- Whether to penalize the *change* in actions as opposed to penalizing the actual action values.
- Whether to use slack variables to penalize the optimizer for choosing values of actions or states above the constraints, or simply leave hard constraints.
- What value to use for the maximum error between the ideal states and the actual states.
- What values to use in matrices $Q$ and $R$;
- What time horizon, $T_h$, to use;
- What control horizon, $T_c$, to use.

A number of experiments were conducted for the first two options to find the best solution for fixed values of $T_h, T_c, Q$ and $R$. Once that result was established, $Q, R, T_c$ and $T_h$ were treated as hyper parameters and are discussed in Section 5.5. The general problem was restructured as shown below.

---

**Main Algorithm 2** MPC

---

$$
\begin{aligned}
&minimize && J = \sum_{t=t_i}^{t_i+T_h} (\Delta x_t^T\, Q \Delta x_t + u_t^T R u_t + \Delta u_t^T (0.1R) \Delta u_t\,) \\
&subject\ to && x_{t+1} = Ax_t + Bu_t \\
& && \frac{Max\ Power_{Main\ Engine}}{6} \leq u_t[\mathrm{F_E}] \leq Max\ Power_{Main\ Engine} \\
& && -Max\ Power_{Side\ Engine} \leq u_t[\mathrm{F_S}] \leq Max\ Power_{Side\ Engine} \\
& && -\varphi_{Max} \leq u_t[\varphi] \leq \varphi_{Max} \\
& && \left|x_{target}[T_h] - x_t[T_h]\right| \leq 0.01 \\
& && x_0 = x_{t_i} \\
&where && \Delta x_t = x_{Target\ trajectory} - x_t \\
& && \Delta u_t = u_{previous} - u_{current}
\end{aligned}
$$

---

Note that a problem can be tested for convexity by computing the Hessian matrix and checking if it is positive semi-definite, that is if it's Eigen values are greater or equal to 0. When considering the constraints mixed with the cost function, this test becomes non-trivial. For this reason, the constraints were left to be as linear as possible since a convex framework, $cvx$, was used to solve the problem. The Splitting Conic Solver was used for the optimization problem since it can solve convex second-order cone programs of the type:

$$\begin{aligned} &minimize && J = c^T x \\ &subject\ to && Ax = b \\ & && Gx \leq Kh \\ & && x_t \in \mathrm{X} \end{aligned}$$

Experiments were conducted to find the optimal values for the hyper parameters discussed earlier and results are discussed in Section 5.5.

### 4.3.3 Trajectory Generation

An $x_{Target\ trajectory}$ was passed to the optimizer to track an *ideal* trajectory at every $x_t$ iteration. This trajectory was computed empirically by first defining an ideal z-profile. The target state was then computed, always starting from $[x_t, \dot{x}_t, z_t, \dot{z}_t, \theta_t, \dot{\theta}_t]$ and computing each state iteration according to $x_{Target\ trajectory}$ shown overleaf. This preserves the constraint $x_{t+1} = Ax_t + Bu_t$ while providing realistic targets within the given time horizon. If instead we gave the default values of $x_{final} = [Barge\ x, 0, Barge\ z, 0,0,0]$, the optimizer would have tried to reach $x_{final}$ within the given horizon, which leads to an infeasible solution.

The Z-altitude profile was structured as:

$$z(t) = z_{Max} e^{-0.05t} + offset_{Barge-z}$$

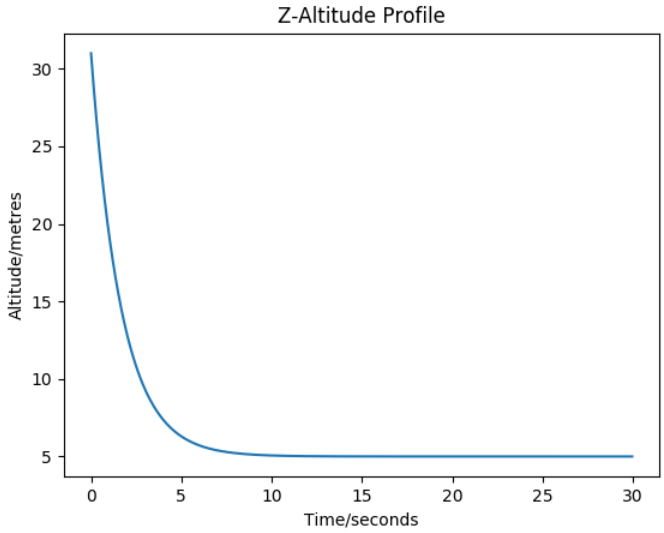


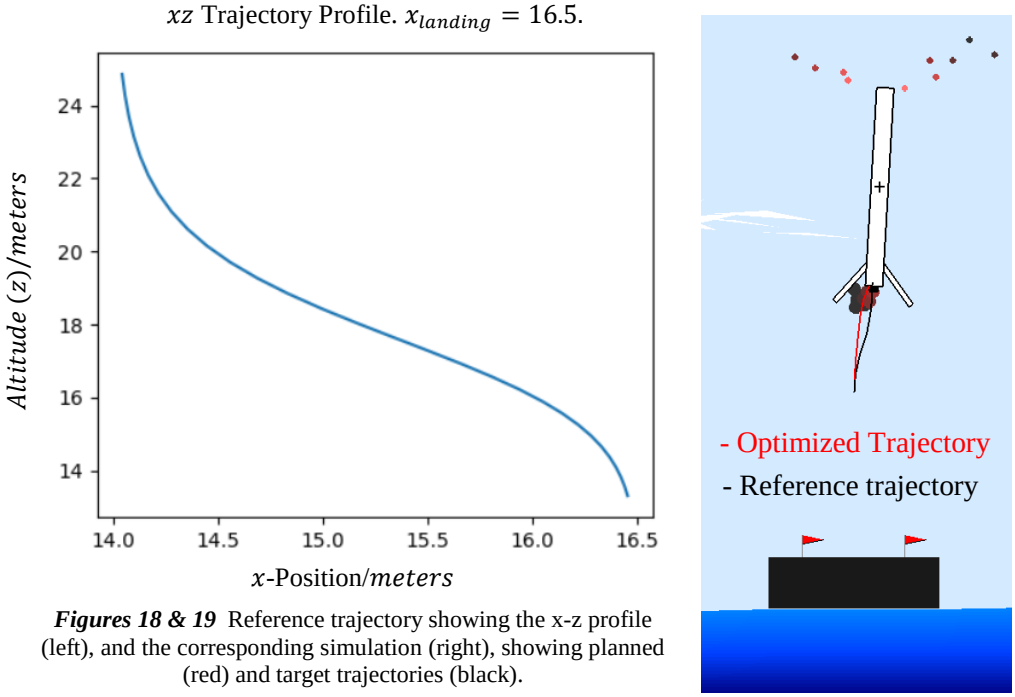

***Figure 17*** Altitude profile against time to land.

Using this graph, the target state was obtained at each time step for the specified time horizon according to the equations shown below:

$$\boldsymbol{x_{Target\ trajectory}} = \begin{bmatrix} x_{current} + \dfrac{(x_{final} - x_{current})}{1 + e^{-t}} \\ x_{current} - x_{final} \\ z_{target}[t_{current} \dots T_h] \\ \dot{z}_{current} * e^{-linspace(0,2,T_h)} \\ \theta_{current} * (0.3 + e^{-\beta t}) \\ \gamma\theta_{current} \end{bmatrix}$$

This target provided a sigmoid-like trajectory with respect to $x, z$ as well as realistic values for $\theta$ and $\dot{\theta}$, tuned using the baseline PID. As an example, using a time horizon of 30 and a time step of 0.1, the reference trajectory specifies that the rocket should decrease the $x, z$ displacements gradually as well as correct for $\theta$ over the entire period, not just one. The reference trajectory is shown in Figures 18.

In Figure 19, the reference trajectory is shown as a black line and the optimized trajectory returned from the optimizer is shown in red. One can notice that the planned and target trajectories do not coincide at every point. This depends on the $error$ value between target states and planned states. It is of utmost importance that the constraint for the final state $\left|x_{target}[T_h] - x_t[T_h]\right| \leq error$ is also included as without this the MPC would fail to provide a reasonable trajectory.




***Figures 18 & 19*** Reference trajectory showing the x-z profile (left), and the corresponding simulation (right), showing planned (red) and target trajectories (black).

## 4.4 Linear Function Approximation Q-Learning

RL refers to the episodic learning of an agent whose goal is to maximize the return. Different types of representations were discussed in Section 2.5, and this sub-chapter expands on Q-learning.

Up till now, the benchmark PID and the MPC controllers were designed and their algorithms were outlined. The PID represented a classical control approach, whereas MPC uses a combination of state space representation with optimization methods. Given that MPC is derived from the same theoretical background as RL, both solve the same problem using the same framework in different ways.

Whereas MPC uses an optimizer and a known model, RL approaches the problem from an interaction-reward point of view. In this section, the state is modeled by an algebraic sum of weighted features. Features can include the actual state $x$ used in MPC, binary states that are activated with a condition, transformed functions such as $\sqrt{x^2 + y^2}$ and any other relevant functions. The general Q-learning algorithm is given by:

---

**Background Algorithm 2** General Q-Learning

---

Initialize $Q(s, a)$ either optimistically or randomly, in tabular form (2-dimensional array)
For every episode:
    Initialize the state $s$
    Until s is the terminal state, do:
        Choose action $a$ from current state $s$ using an ε-greedy policy
        Execute action $a$ and get reward, $r$, as well as the next state, $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$

---

However, Background Algorithm 2 above uses discrete states and actions. On the other hand, our problem spans both continuous states as well as actions. This introduces the problem of the curse of dimensionality; if we try to discretize all states and actions, $Q(s, a)$ becomes infeasible.

Mnih et al. [48] chose to replace the tabular approach with a deep convolutional neural network trained using stochastic gradient descent and a replay mechanism which randomly samples previous transitions. The replay buffer was also used for this project in the implementation of the DDPG algorithm discussed in Section 4.5. The researchers effectively used the video images having a resolution of 84×84 at 60Hz as input to the neural network, and output *all* actions as a posterior probability of the state. The highest valued action would then be executed. Tile coding and

Radial Basis Functions are other state representations that have proven to be effective in certain simple continuous state problems but were side-lined for a simpler solution in this implementation due to their scalability limitations. Note that these solve the problem of states, but not actions.

Consider the true state-action matrix to be linearly approximated by another weighted matrix:

$$Q(s,a) \approx Q(s,a,\omega)$$

The goal is to find a representation for $\omega$ that minimizes the cost function:

$$\min J(\omega) = E\left[\left(Q(s,a) - Q(s,a,\omega)\right)^2\right]$$

To this end, gradient descent can be used to find the local minimum in an online fashion:

$$-\frac{1}{2}\nabla J(\omega) = \alpha[Q(s,a) - Q(s,a,\omega)].\nabla Q(s,a,\omega)$$

Where α is the learning rate and $Q(s,a)$ represents the target. The target depends on how the problem is formed; Monte Carlo RL implies that $Q(s,a)$ be equal to the average reward at the end of the episode, whereas for temporal differencing it would be the immediate reward following state $s$: $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}, \omega)$. Using such a method, $Q(s,a,\omega)$ still converges to the global solution if on-policy linear function approximation is used.

The main algorithm used is shown below, with α being annealed from 0.01 to 0.001 using an exponential decay. All features are explained overleaf.

---

**Main Algorithm 3** Linear Function Approximation Q-Learning – Off Policy

---

Initialize $\omega$ randomly from $U[0,1]$ with size equivalent to the number of features.
For every episode:
    Until s is the terminal state, do:
        Choose action $a$ using an ε-greedy policy, $argmax\ Q(s, a \in A, \omega)$
        Execute action $a$ and get reward, $r$, as well as the next state, $s'$
        $features = build\ features\ using\ (s,a)$
        $Q(s,a,\omega) = \omega^T features$
        $\omega \leftarrow \omega + \alpha[r + \gamma \max_a Q(s',a,\omega) - Q(s,a,\omega)]\nabla\omega$
        $s \leftarrow s'$

---

The action space was discretized in the following way:

$$F_E = \{0.6, 0.8\}$$
$$F_S = \{-1, 0, 1\}$$
$$\varphi = \{-5, -1.5, 0, 1.5, 5\}$$

The actions were built using each tuple combination of $\{f_E, f_S, \varphi_i\}, f_E \in F_E, f_S \in F_S, \varphi_i \in \varphi$, for a total of 30 action combinations. The discretization was chosen in an intuitive manner, making sure that the values made sense for the landing to be successful. A total of 20 features were used and are listed below in Table 1. These features represent a combination of action dependent features and general state descriptors, giving enough information for a robust and flexible representation of $Q(s, a, \omega)$. Features were experimented upon to a limited extent; there are many combinations and additional features that can be included, especially when the action space is discretized finely.

This method proved to perform reasonably well given the fact that the state was approximated and the action space was discretized. It also laid the foundation for the more advanced DDPG algorithm, which was specifically created to cater for both continuous states and continuous actions.

***Table 1*** *Features used in linear function approximation Q-Learning*

| Feature Type | Feature | Size | Description |
|---|---|---|---|
| **General** | $state\ \boldsymbol{x}$ | 8 by 1 | $[x, \dot{x}, z, \dot{z}, \theta, \dot{\theta}, leg\ contact_{left}, leg\ contact_{right}]$ |
| | $k$ | 1 by 1 | $Free\ parameter$ |
| | 3 binary states | 3 by 1 | Thresholds for $[\dot{x}, \dot{z}, \dot{\theta}]$, $[1\ m/s, 1\ m/s, 5^o/s\ ]$ |
| **Action Dependent** | 8 binary states. Conditions:<br><br>$\theta > 0\ and\ F_S < 0$<br>$\theta < 0\ and\ F_S > 0$<br>$\theta > 0\ and\ \varphi < 0$<br>$\theta < 0\ and\ \varphi_S > 0$<br>$\theta < 0\ and\ \ddot{\theta} < 0$<br>$\theta > 0\ and\ \ddot{\theta} > 0$<br>$x > 0\ and\ \ddot{x} < 0$<br>$\ddot{z} < 2$ | 8 by 1 | Simulate rocket kinematics using Equations 1-3 and return $\ddot{x}, \ddot{z}, \ddot{\theta}$ using the current state and passed action. |

## 4.5 Deep Deterministic Policy Gradient

The notion of having policy evaluation separate from the policy improvement step is known as Actor-Critic (AC) architecture. The policy evaluation step, known as the critic, criticizes the actions chosen by the actor based on the target reward and outcome discrepancy as shown in Figure 20 [49].

The actor is represented by a function $h(a_t|s_t;\theta)$ that indicates the value of each action to be taken given the current state *estimate*. Actions are chosen in a probabilistic manner given by Gibbs softmax method;

$$\pi_t(a|s;\theta) = P[a|s;\theta] = \frac{e^{h(a|s;\theta)}}{\sum_b e^{h(b|s;\theta)}}$$

The dependence on $\theta$ implies the representation of the policy as a probability distribution, a domain known as policy gradients. The idea is to adjust the parameters $\theta$ in the direction of the gradient $\nabla_\theta J(\pi_\theta) = E[\nabla_\theta \log(\pi_\theta(a|s))Q^\pi(s,a)]$ [50]. The actor adjusts $\theta$, whereas the critic estimates the state-action function $Q^\omega(s,a) \approx Q^\pi(s,a)$ using a policy evaluation algorithm such as temporal differencing. Notice that this still relies on function approximation, however, the number of parameters is much greater, and features are created by the network and are not tuned manually.

Lillicrap et al. [43] presented an AC, model-free algorithm based on deterministic policy gradients, that is capable of operating over continuous action spaces. They reported that using the same architecture, hyper parameters, as well as learning algorithm, the network solved more than 20 physics tasks, including control problems. However, they still used image data as input. Although this is certainly possible in the rocket landing simulation, it does not present a realistic and practical implementation. This is because control problems usually rely on sensory input, and not imagery input. In fact, noisy sensory measurements on board rockets, such as gyroscope data, are filtered, estimated and compensated for during flight. To this end, in this project, we refrained from using any video input.

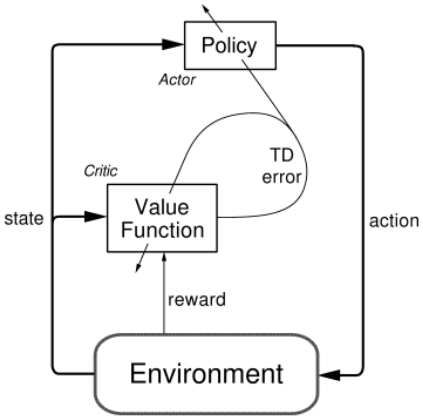



***Figure 20*** Actor-Critic architecture [49]

### 4.5.1 Architecture

Lillicrap's algorithm takes the concepts of AC and deterministic policy gradients and combines them with deep learning taken from DQN as well as Q-learning in order to have both continuous states and continuous actions.

The authors highlight that although non-linear function approximators were avoided in the past due to instability and convergence problems, taking certain measures, such as introducing a replay buffer, makes them very effective. Consequently, neural networks were used in this case.

The architecture involved:

- A Neural network for the actor, $Q(s, a|\theta)$.
- A Neural network for the critic, $\mu(s|\theta)$.
- Learning in mini-batches.
- A replay buffer which stores a finite amount of $(s, a, r, s')$ transitions. At each time step, the actor and critic are updated by sampling from the replay buffer. The replay buffer is essential because it provides uncorrelated samples, unlike the normal sequential steps required for the simulation.
- Copies of both networks to estimate the target, improving convergence and stability. This is done because learning diverges if the critic network ($Q(s, a|\theta)$) is updated whilst also being used for calculating the target. The authors noted that both networks need to be copied.
- Exponential averaging for the copied networks' weights to track the actual networks' weights. Changes are guaranteed to be slow, slowing down learning but improving stability.
- Batch normalization for normalization of features across all dimensions. This makes the features have 0 mean and unit variance, effectively whitening the input given to each layer [51]. This was proven to be very effective, enabling higher learning rates and serves as a regulariser.
- Maintaining exploration of the action space by adding a noise term to the output of the actor: $\mu(s') = \mu(s|\theta) + N$. The authors implement an Ornstein-Uhlenbeck process given by:

$$\mu(s') = \mu(s|\theta) + \omega\left(\mu_{given\ mean} - \mu(s|\theta)\right) + N(0, \sigma)$$

The neural networks for the actor and critic were both created using 3 layers. The first two consisted of 300 and 400 neurons using $relu$ and $sigmoid$ activation functions respectively. The last layer of the actor was set as the size of the action space (3), implemented with a $tanh$ function to bind the outputs to their proper sizes. The critic simply had 1 output value with no implemented activation function.

The algorithm is written overleaf, and Figure 21 graphically illustrates the architecture.

### 4.5.2 Algorithm

**Main Algorithm 4** Deep Deterministic Policy Gradients [43] – without Batch Norm.

Initialize actor and critic networks using Xavier initializer using the specified parameters. $Actor = \mu(s|\theta^{\mu})$, $Critic = Q(s, a|\theta^{Q})$, where $\theta^{Q}$ and $\theta^{\mu}$ indicate the weights of the respective networks.
Initialize target networks $\mu'$ and $Q'$, the same as the actor and critic respectively.
Initialize Replay Buffer, $R$
For every episode:
    $state, s = reset\ simulation$
    $s = normalize\ state$
    Until s is the terminal state, do:
        Choose action $a$; $a = \mu(s|\theta^{\mu}) + N$
        Execute action $a$ and get reward, $r$, as well as the next state, $s'$
        Store $(s, a, r, s', done)$ in $R$
        Sample a random minibatch of $N$ transitions from $R$ to update the critic
        If $done\ is\ True$ (terminal state):
            Set $y_i = r_i$
        Else:
            Set $y_i = r_i + \gamma Q'(s', \mu'(s'|\theta^{\mu'})|\theta^{Q'})$
        Update the critic by minimizing the loss $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^{Q})^2)$
        Update the actor:
            $\nabla_{\theta^{\mu}}\mu \approx \frac{1}{N}\sum_i [\nabla_a Q(s_i, \mu(s_i)|\theta^{Q})\, \nabla_{\theta^{\mu}}\mu(s_i|\theta^{\mu})]$
        Update the target (copied) networks:
            $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$

Notice that the critic is updated by minimizing the loss function given the new target obtained from the target network. $Q'(s', \mu'(s'|\theta^{\mu'}))$ is obtained by inputting the new state as well as actions into the critic *target* network. This implies that the input to the critic network is equivalent to $[states, actions]$ whereas for the actor it is simply $[states]$. Moreover, the actor uses the gradients of the critic as well as its own gradients to update the actual network. This is simply an extension of the chain rule with respect to the equation shown below:

$$Q^{\mu}(s_t, a_t) = E[r(s, a) + \gamma Q^{\mu}\big(s', \mu(s')\big)]$$

The description, as well as complete algorithm, is well documented in [43]. The reward in both linear function approximation and DDPG was given by:

$$\begin{aligned} reward = \; & -20\sqrt{x^2 + z^2} - 10\sqrt{\dot{x}^2 + \dot{z}^2} - 100\theta - 3\dot{\theta} \\ & + 2\big(left_{leg_{contact}} + right_{leg_{contact}}\big) - 0.03 EnginePower_{Side} \\ & - 0.003 EnginePower_{Main} \end{aligned}$$

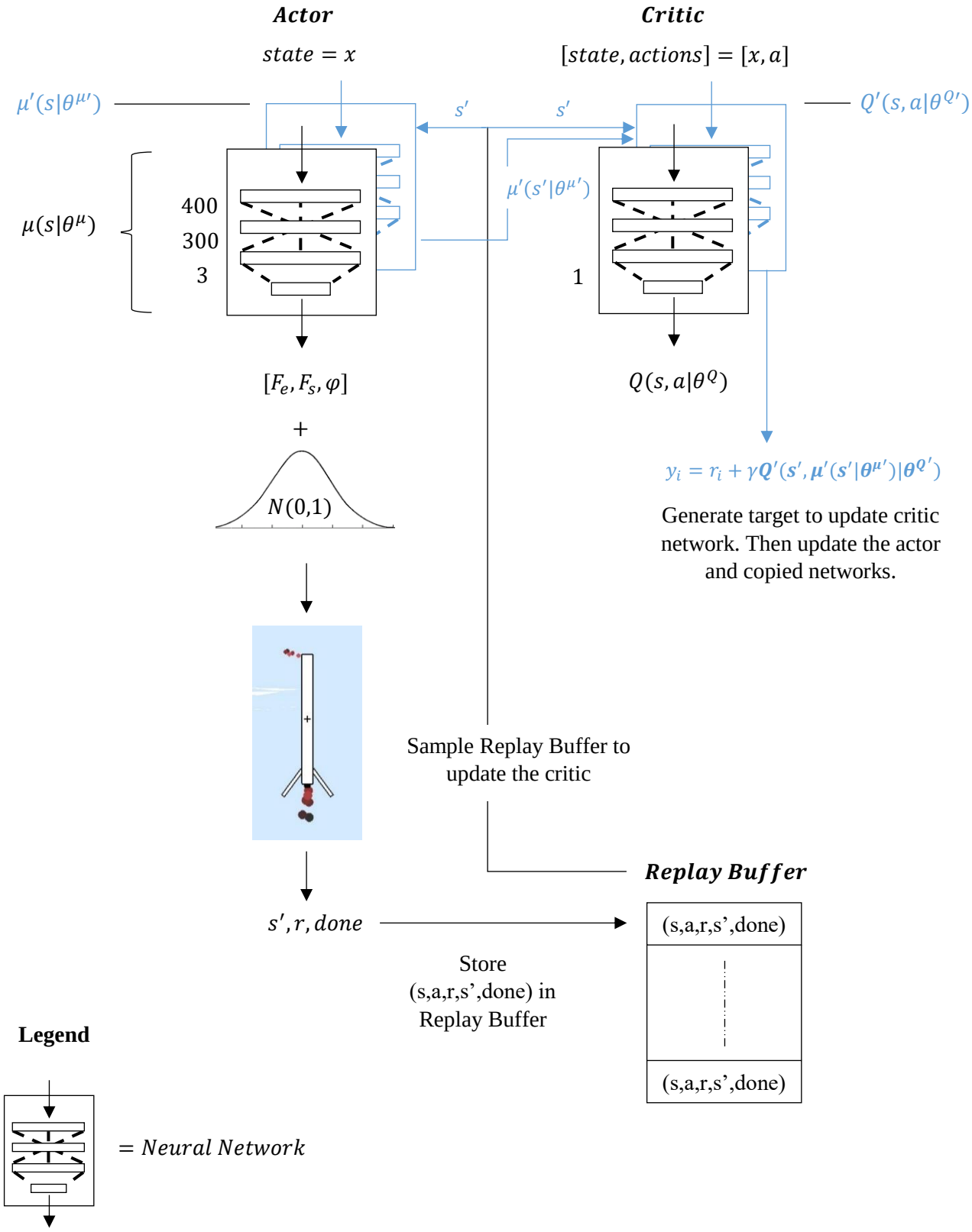



***Figure 21*** Visual guide for the DDPG algorithm

## 4.6 Conclusion

The designs of the 4 algorithms presented in the Background were presented and discussed in detail. Each algorithm was meant to serve as a framework for the next, with each method tackling a drawback that the previous one had.

The AI section in particular was meant to contrast with the traditional control by solving the problem in a different way. It is clear that the AI algorithms have the advantage of handling model-free problems in a complex, yet effective way. This enables the same model to be trained on different problems without needing to redefine a model, an appealing attribute for control problems.

MPC and RL were both derived from Bellman's equations, but RL takes the dynamic programming approach to further simplify the problem to a model-free, online algorithm, whereas MPC uses the linearized model to predict future states and find optimal inputs that yield the lowest cost within the specified time horizon.

The classical control approach requires quite a bit of design and knowledge on the subject to even be able to design the simplest algorithm. However, PIDs, LQR, and MPC are widely used in many systems in the industry, whereas AI is just starting to have a large, yet slow impact in many industries.

These 4 algorithms were evaluated against 24 tests using different hyper parameters and algorithm variants. These are presented and discussed in the next section.

# 5 Evaluation and Discussion

In this section, each controller is investigated individually with respect to different initial conditions, impulse and step disturbances, total reward, final cost, fuel consumption and successful landings. Moreover, a thorough analysis of the effect that multiple hyper parameters have on learning or computation is discussed for each controller excluding the PID (no hyper parameters exist for this case). Discussions follow the results of every controller, with comparisons being drawn to the benchmark PID controller or otherwise. Furthermore, trajectories and actions are analysed for certain tests to further contrast the controllers.

## 5.1 Tests

Each controller was evaluated with a total of 24 tests that vary the initial coordinates, translational $x, z$ velocities and $\theta$. In a small subset of tests, the rocket was exposed to impulse and step forces in the $x$ direction after a given time. The impulses are meant to represent sudden random forces that the rocket might experience, such as an external force, whereas the constant force is meant to represent a constant side wind. The tests are outlined in Table 2.

***Table 2*** *Tests performed on each controller varied in initial conditions and external disturbances.*

| | **Test Number** | **Initial Conditions** | | | | | | **Disturbances** | |
|---|---|---|---|---|---|---|---|---|---|
| | | $x$ | $z$ | $\dot{x}$ | $\dot{z}$ | $\theta$ | $\dot{\theta}$ | **Step** | **Impulse** |
| **No Translational Velocity** | 1 | 0.5 | 1 | 0 | -15 | 0 | 0 | - | - |
| | 2 | 0.5 | 1 | 0 | -15 | 5 | 0 | - | - |
| | 3 | 0.3 | 1 | 0 | -15 | 0 | 0 | - | - |
| | 4 | 0.3 | 1 | 0 | -15 | 5 | 0 | - | - |
| | 5 | 0.3 | 1 | 0 | -15 | -5 | 0 | - | - |
| **$x$ translational velocity varied** | 6 | 0.5 | 1 | 3 | -15 | 0 | 0 | - | - |
| | 7 | 0.5 | 1 | -3 | -15 | 5 | 0 | - | - |
| | 8 | 0.3 | 1 | 3 | -15 | 0 | 0 | - | - |
| | 9 | 0.3 | 1 | -3 | -15 | 5 | 0 | - | - |
| | 10 | 0.3 | 1 | -3 | -15 | -5 | 0 | - | - |
| **$z$ translational velocity increased** | 11 | 0.5 | 1 | 3 | -19 | 0 | 0 | - | - |
| | 12 | 0.5 | 1 | -3 | -19 | 5 | 0 | - | - |
| | 13 | 0.3 | 1 | 3 | -19 | 0 | 0 | - | - |
| | 14 | 0.3 | 1 | -3 | -19 | 5 | 0 | - | - |
| | 15 | 0.3 | 1 | -3 | -19 | -5 | 0 | - | - |
| **Initializations with higher $\theta$** | 16 | 0.5 | 1 | 0 | -15 | 10 | 0 | - | - |
| | 17 | 0.3 | 1 | 0 | -15 | 10 | 0 | - | - |
| | 18 | 0.3 | 1 | 0 | -15 | -10 | 0 | - | - |
| **Added step forces to tests 1-3** | 19 | 0.5 | 1 | 0 | -15 | 0 | 0 | (12, 0) | - |
| | 20 | 0.5 | 1 | 0 | -15 | 5 | 0 | (12, 0) | - |
| | 21 | 0.5 | 1 | 0 | -15 | -5 | 0 | (12, 0) | - |
| **Added impulses to tests 1-3** | 22 | 0.5 | 1 | 0 | -15 | 0 | 0 | - | (2000, 0) |
| | 23 | 0.5 | 1 | 0 | -15 | 5 | 0 | - | (2000, 0) |
| | 24 | 0.5 | 1 | 0 | -15 | -5 | 0 | - | (2000, 0) |
| **Units** | | Norm. | Norm. | m/s | m/s | Deg. | Deg./s | $N$ | $N\delta t$ |

$x$ and $z$ coordinates are normalized, meaning that $x = 0.5$ refers to the middle of the simulation space. Translational velocities are quoted in meters/second, whereas angles are all in degrees. Where applicable, the impulses were applied precisely 1 second after the simulation started, whereas forces were applied from time $t = 0$ and continued until termination. Both forces and impulses were applied to the center of moment of the rocket. Note that in the simulation, an impulse represents a force that is applied only at time $t = t_i$; that is the normal force multiplied by the time step. The difference between forces and impulses is that forces act over time, whereas impulses are able change to the velocity of an object immediately.

## 5.2 Test Measures

Controllers were evaluated with respect to (i) total reward, (ii) final cost, (iii) fuel consumption and (iv) successful landings. The percentage use of each action was also included in each result due to the insight it gives on the measures that the controller took to land the rocket successfully. Together with the average rocket angle throughout each test, $\theta_{average}$, the results provide an objective view of how each controller performed as well as what happened. For instance, a controller experiencing $|\theta_{average}| > 8^o$, high use of gas thrusters with a landing on one leg indicates that the rocket landed with a high angle of attack and eventually toppled over.

The metrics are defined as follows. The total reward is given by:

$$\begin{aligned} reward = \; & -20\sqrt{x^2 + z^2} - 10\sqrt{\dot{x}^2 + \dot{z}^2} - 100\theta - 3\dot{\theta} \\ & + 2\left(left_{leg_{contact}} + right_{leg_{contact}}\right) - 0.03EnginePower_{Side} \\ & - 0.003EnginePower_{Main} \end{aligned}$$

$$returned\ reward = reward(t) - reward(t-1)$$

The reward penalizes both certain states as well as actions. It forms an essential part of RL, therefore positive rewards were rewarded for actions that represent "good" results, such as legs touching the barge.

The *final cost* is computed by:

$$final\ cost = \omega^T\left(state_{target} - state^+\right)$$

Where

$$state^+ = \left|\left[x, \dot{x}, z, \dot{z}, \theta, \dot{\theta}, fuel\ consumed\right]\right|$$
$$Weighting\ matrix = \omega = [2,1,1.5,3,4,2,1]$$

The final cost provides a linear, monotonic metric to judge the landing accuracy as well as resources used to get there. It ranges from $(-\infty, 0]$, with 0 being the best score. The states $x$ and $z$ in the simulation's $state$ encapsulate the difference from the target. Thus, the target state, $state_{target}$, represented the 0 vector.

The *fuel consumption* is represented by a linear weighted combination of $F_E$ and $F_S$. The mass of the rocket was divided into 80% dry mass, and 20% fuel mass. The rocket weighed 25.222 kg in total after scaling, computed from:

$$\begin{aligned} mass &= \frac{density.area}{SCALE^2} \\ &= \frac{5.\left(length_{main\ body} \times breath_{main\ body}\right)}{30^2} \\ &= \frac{5.(227 \times 20)}{900} \\ &= 25.222\ kg \end{aligned}$$

Therefore, the fuel represented just 5.04 kg of the rocket and decreased linearly according to the equation:

$$\begin{aligned} fuel\ consumed &= 0.009[F_E.ME_{Fuel\ cost} + F_S SE_{Fuel\ cost}] \\ &= 0.009\left[F_E \frac{Main\ Engine\ Power}{Side\ Engine\ Power} + F_S\right] \end{aligned}$$

The burn rate was adjusted empirically from multiple simulation runs. In reality, the simulated rocket would have weighed 22,699.8 kg, which is an estimate for a small first stage rocket on re-entry.

Finally, the *landing* metric was used to indicate whether the rocket landed successfully on both legs, a single leg, or did not land at all; $landing = \{1,0,-1\}$ respectively. Note that the speed with which the rocket landed was only taken into consideration in the $final\ cost$. Therefore, even though a landing with high velocity would result in the rocket's destruction, the landing metric would still indicate a successful landing. This highlights why multiple metrics were needed.

## 5.3 Test Conditions

In all tests, the barge length covered 46% of the total width of the simulator, approximately 5 times the length between the rocket legs as shown in Figure 22. The barge was left static, with the landing target located always at the center point $\left(x - x_{target}, z - z_{target}\right) = (x - \frac{Barge\ Length}{2}, z - Barge\ Height)$.

The maximum main engine power was limited to 60% of its maximum value, whereas if activated, the side engines had to operate between 50%-100% of the total power. All other variables were kept constant unless otherwise specified in each individual test.

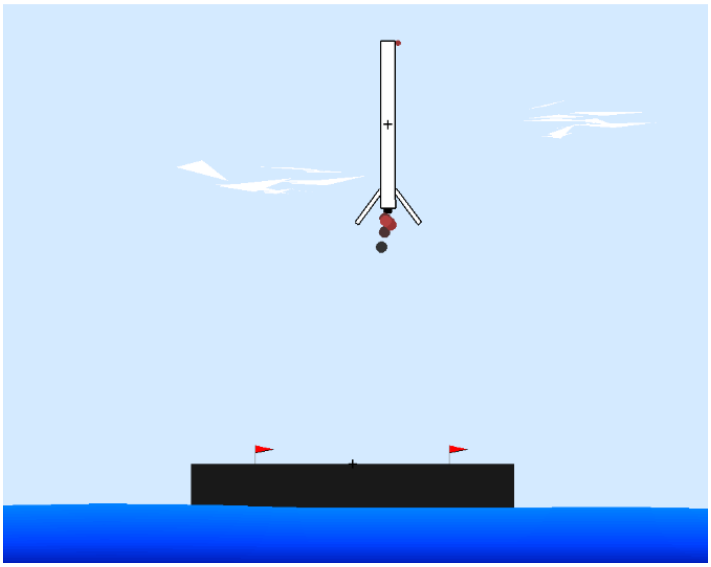


***Figure 22*** Simulation showing the relative dimensions of the barge to the simulator used during the tests. The middle of the barge represented the landing target location.

## 5.4 PID Results

Recall that classical control was included as a benchmark, and so the 3 PIDs were designed by taking strong independence assumptions on certain states, and later introducing the ignored states as part of the error of each PID. This is the empirical method of finding feedback weights between states to aid decoupling.

Table 3 shows the results for the PID control, where green highlights better or more favourable numerical values, whereas red indicates worse. Blue highlights higher values but indicates neutrality.

For its simplicity, the 3 PIDs managed to achieve quite a good result for a wide range of conditions. This is not to say that the control is optimal; in fact, some defects resulting from the assumptions can be noticed even by visual inspection of the simulation.

As an example, in test 3 the rocket attempted to navigate to the target starting from $x = 0.3$ with a clear intention of keeping the rocket upright, evident from the low value of $\theta_{average}$. The controller made good use of $\varphi$, as shown by the percentage of time spent with $|\varphi| > 3$, whereas $F_s$ was used sparingly. The rocket landed in the water, even though proper balancing was achieved.

Tests 9 and 17 show the use of $F_S$ for higher lander angles. In test 17, the moment caused by $\dot{x}$ in the direction of the initialized $\theta$ was too high for it to recover, but was successful in test 18. It comes to no surprise that $F_S$ is used to a higher degree for higher values of $\theta$.

Interestingly, increasing $\dot{z}_{initial}$ decreases the amount of fuel consumed whilst still achieving high rewarding landings. At first this might seem counterintuitive, but upon further inspection, this is dependent on the tuning of the PID parameters that control $F_E$, particularly $K_D$. The PID was designed for the rocket to land in a smooth and slow manner, ideally with $\dot{z} < 2m/s$ so the legs can absorb the rest of the forces. However, this also makes for a slow descent, burning more fuel in the process. Not surprisingly, the fuel consumed in disturbance tests is on average higher than the rest of the tests, but the percentage of time that $F_E$ and $F_S$ were used had no significant differences. This suggests that in the disturbance tests, $F_E$ and $F_S$ were throttled higher to compensate for the external disturbances.

A valid point to think about is why $F_E$ is not used 100% of the time. $F_E$ cannot be less than 0, but the PID still tries to make it so since no constraint was included. Thresholding can limit $F_E$ to any level, and although in real rocket engines the engine can be throttled, it is never switched off completely. Figure 23 illustrates the control actions taken in tests 19 and 23. Notice the saturating $F_E$ and higher sinusoidal effect of $\varphi$ in the step disturbance test. If $K_p$ is large enough in the $\varphi$-PID, this sinusoidal effect leads to unstable behaviour. The impulse in test 23 was applied in the direction of the target, and since it is applied to the center of moment the rocket remained stable.

***Table 3*** *PID controller numerical results for the 24 tests.*

| Barge length = 46% of width | | | | | | | Percentage of time that action was utilised (0-1) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | Test Type | Total Reward | Final Cost | Fuel Consumed | Average θ | Landed | Fe | Fl | Fr | \|psi\| > 3 |
| 1 | No Translational Velocity | 27.52 | -6.00 | 2.75 | 2.84 | 1.00 | 0.38 | 0.01 | 0.01 | 0.38 |
| 2 | | 24.04 | -6.96 | 2.48 | 0.75 | 1.00 | 0.48 | 0.17 | 0.00 | 0.38 |
| 3 | | -11.62 | -13.64 | 2.39 | 0.62 | 0.00 | 0.61 | 0.03 | 0.00 | 0.52 |
| 4 | | 21.58 | -6.49 | 2.77 | 3.61 | 1.00 | 0.30 | 0.35 | 0.00 | 0.45 |
| 5 | | 29.27 | -4.80 | 2.65 | 0.61 | 1.00 | 0.41 | 0.07 | 0.07 | 0.38 |
| 6 | x translational velocity varied | 28.91 | -4.85 | 2.57 | 0.88 | 1.00 | 0.38 | 0.00 | 0.06 | 0.41 |
| 7 | | 22.99 | -7.37 | 2.52 | 1.07 | 1.00 | 0.59 | 0.39 | 0.00 | 0.49 |
| 8 | | 28.76 | -5.29 | 2.57 | 1.98 | 1.00 | 0.71 | 0.11 | 0.08 | 0.56 |
| 9 | | -24.12 | -16.20 | 3.00 | 13.49 | 0.00 | 0.65 | 0.98 | 0.00 | 0.75 |
| 10 | | 28.53 | -4.94 | 2.64 | 0.66 | 1.00 | 0.35 | 0.17 | 0.00 | 0.46 |
| 11 | z translational velocity varied | 27.99 | -5.40 | 1.67 | 1.92 | 1.00 | 0.37 | 0.00 | 0.10 | 0.32 |
| 12 | | 21.26 | -7.67 | 1.98 | 0.89 | 1.00 | 0.38 | 0.21 | 0.00 | 0.31 |
| 13 | | 35.01 | -4.08 | 1.30 | 1.32 | 1.00 | 0.36 | 0.01 | 0.07 | 0.38 |
| 14 | | -10.64 | -11.06 | 2.03 | 1.26 | 1.00 | 0.57 | 0.59 | 0.00 | 0.57 |
| 15 | | 27.67 | -6.44 | 1.79 | -0.22 | 1.00 | 0.34 | 0.03 | 0.00 | 0.27 |
| 16 | Initializations with higher theta | 23.89 | -5.32 | 2.48 | 5.65 | 1.00 | 0.64 | 0.45 | 0.00 | 0.41 |
| 17 | | -76.64 | -13.75 | 2.95 | 17.78 | -1.00 | 0.58 | 0.88 | 0.00 | 0.82 |
| 18 | | 28.25 | -4.83 | 2.60 | -0.60 | 1.00 | 0.40 | 0.04 | 0.15 | 0.38 |
| 19 | Added step forces to tests 1-3 | 31.24 | -5.16 | 2.56 | -0.39 | 1.00 | 0.71 | 0.00 | 0.00 | 0.40 |
| 20 | | 28.75 | -4.93 | 2.64 | 0.99 | 1.00 | 0.40 | 0.06 | 0.00 | 0.29 |
| 21 | | 23.11 | -7.35 | 2.82 | 0.72 | 1.00 | 0.34 | 0.00 | 0.12 | 0.36 |
| 22 | Added impulses to tests 1-3 | 26.71 | -6.46 | 2.67 | -0.09 | 1.00 | 0.41 | 0.00 | 0.00 | 0.27 |
| 23 | | 29.96 | -5.60 | 2.57 | 0.74 | 1.00 | 0.41 | 0.12 | 0.01 | 0.36 |
| 24 | | 22.74 | -7.87 | 2.91 | 0.46 | 1.00 | 0.40 | 0.00 | 0.08 | 0.36 |
| **Mean** | | **17.30** | **-7.19** | **2.47** | **2.37** | **0.83** | **0.46** | **0.19** | **0.03** | **0.43** |
| **Std. Dev.** | | **24.36** | **3.15** | **0.41** | **4.25** | **0.47** | **0.13** | **0.27** | **0.05** | **0.13** |

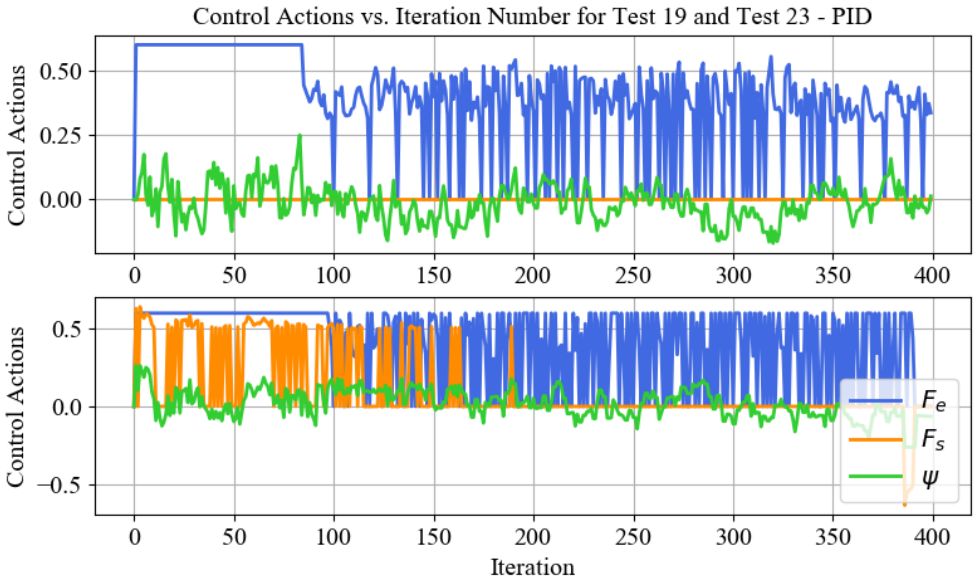



***Figure 23*** Control actions for 400 iterations for tests 19 (top) and 23 (bottom).

Figure 24 shows the correlation between the total reward and final costs obtained across all tests. The positive correlation is apparent for all tests, but outliers can still occur. For instance, although the rocket still landed successfully in test 14, the final cost suggests that the landing was not as desired; close to the target, upright, with low terminal velocity. All the above analysis serves as a preliminary for the other models.

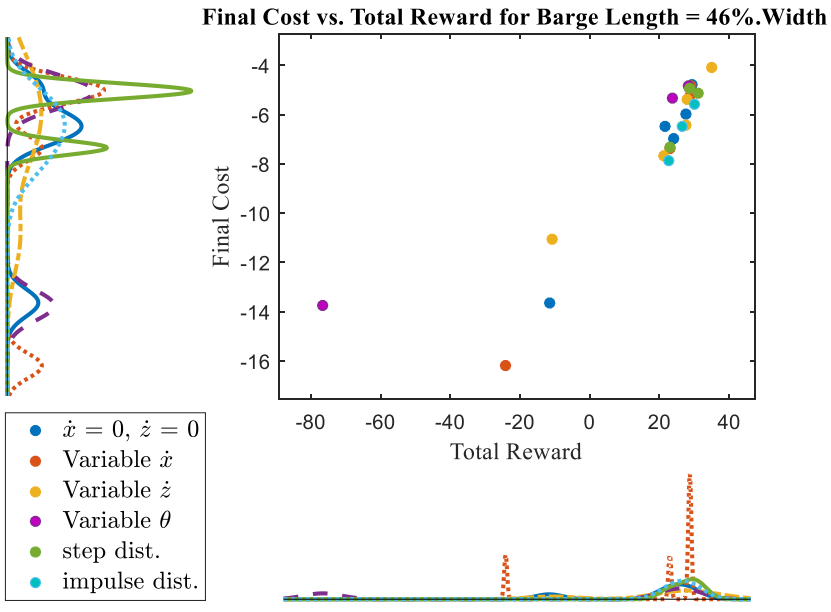



***Figure 24*** Final Cost vs. Total Reward for the different tests. The scatter plot shows the correlation between having a higher reward and having a more successful landing.

## 5.5 MPC Results

The MPC algorithm was outlined in Main Algorithm 2 in Section 4.3.2. Various hyper parameters exist for such a model, including time and control horizons, cost matrices, objective function formulation, model linearization and more. Like LQR, MPC still relies on a state space model, therefore the model was linearized by finite differences, with the reference trajectory being generated on every iteration. Preliminary testing on the effects of cost matrices Q and R as well as variable time and control horizons were conducted. Following this, a single model was evaluated against the rest of the controllers.

Unlike the rest of the models, in MPC the "untransformed state" was used. This means that the values in the state were not normalized or adjusted in any way, and the units represent the actual units of the simulation; meters, kilograms, Newtons. This is because linearization took place using finite differences by instantiating multiple simulations in parallel and adjusting each state and action separately to inspect the change in states. This was used to compute $\delta x_{t+1} = A\delta x_t + B\delta u_t$.

### 5.5.1 Effect of Costs Q and R

Recall that the MPC algorithm is structured as follows:

$$
\begin{aligned}
&minimize && J = \sum_{t=t_i}^{t_i+T_h} (x_t^T Q x_t + u_t^T R u_t + \alpha \Delta u_t^T R \Delta u_t) \\
&subject\ to && u_t \in \mathrm{U},\ x_t \in \mathrm{X} \\
& && x_{t+1} = Ax_t + Bu_t \\
& && |x_{t_i+T_h} - x_{target}| \leq \text{error} \\
& && x_{t_i} = x_{initial} \\
& && for\ t = t_i \ldots T_h
\end{aligned}
$$

$Q$ and $R$ reflect the balance between state deviation and action usage respectively. Choosing the diagonal weights to be equal to $\frac{1}{maximum\ error^2}$ and then using trial and error until a suitable response is achieved is a widely-used method in MPC. The time horizon, $T_h$ was kept at 10 time steps, whilst $T_c$ was held at 1.

Let:

$$State\ Accuracies = [\pm 2\ m, \pm 2\ m/s, \pm 0.5\ m, \pm 2\ m/s, \pm 3^o, \pm 5^o/s]$$

Then:

$$Q = k[0.25, 0.25, 4, 0.25, 365, 131].I$$

$$R = \rho[0.01, 0.1, 0.001]I$$

Where $k$ and $\rho$ were adjusted to test the effect that they had on control performance.

Test 1 was used as a preliminary test with $k = \{1, 100, 1000, 1000\}$ and $\rho = \{10, 1, 0.1, 1e^{-10}\}$ for a total of 4 runs. The last test represents an extreme case, where almost no penalty is given to the inputs.

Table 4 below shows the results of the tests discussed above. Notice how changing $p$ to a lower value did not have much of an impact, unless a very low value was assigned to it, in this case $1e^{-10}$. This increased the use of $F_l$ and $F_r$, but still resulted in an unsuccessful landing. Increasing the penalty for $F_E$ also had the same effect, without requiring such a low weight value.

The tests were repeated, and in some cases, completely different results were obtained. Such is the case in the successful landing of test 2. Even though all initial conditions were the same, tests 2 and 3 obtained completely opposite results, with the second test exhibiting proper control. This highlights the downfall of MPC in this problem; we are dealing with a non-linear and coupled problem, and convergence is not guaranteed. Solutions given by the optimizer differ, and non-convexity makes the problem susceptible to local optima. The problem with the control is evident from the high percentage use of $|\varphi| > 3$ across all failures.

To amend this, the penalty for using $\varphi$ was increased to 10, however, little difference was observed. The weight was further increased whilst keeping the rest of the weights unchanged. This came at the cost of accuracy, as the trajectory no longer followed the target, meaning the optimizer failed to find a solution for high values of $R_3$.

***Table 4*** *Results from testing different combinations of magnitudes for cost matrices Q and R.*

| **Time Horizon = 10, Control Horizon = 1** | | | | | | **Percentage of time that action was utilised (0-1)** | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Test** | **Total Reward** | **Final Cost** | **Fuel Consumed** | **Average θ** | **Landed** | **Fe** | **Fl** | **Fr** | **\|psi\| > 3** |
| (k, p) = (1, 10) | -76.52 | -28.44 | 0.52 | -9.03 | -1.00 | 0.99 | 0.00 | 0.00 | 0.98 |
| (k, p) = (100, 1) | 32.23 | -3.70 | 0.84 | 2.02 | 1.00 | 0.30 | 0.00 | 0.00 | 0.53 |
| | -79.01 | -30.80 | 0.48 | -9.62 | -1.00 | 0.99 | 0.00 | 0.00 | 0.94 |
| (k, p) = (1000, 0.1) | -81.07 | -31.93 | 0.42 | -10.10 | -1.00 | 0.98 | 0.00 | 0.00 | 0.97 |
| (k, p) = (1000, 1e-10) | -73.08 | -28.40 | 0.68 | 11.09 | -1.00 | 0.99 | 0.24 | 0.14 | 0.80 |

### 5.5.2 Effect of Time and Control Horizons

$T_h$ and $T_c$ were varied using the following values: $T_h = \{5, 10, 25\}$ for each value of $T_c = \{1, 2, 5\}$ for a total of 9 tests. The chosen test was still Test 1. Note that the time horizon reflects how many time steps the optimizer should predict using the linearized state. Since the simulation runs with a frequency of $60Hz$, a $T_h$ of 60 represents a prediction horizon of 1 second.

***Table 5*** *Time horizon and control horizon grid search results for Test 1.*

| R = [0.1, 0.01, 10], Q = 100*[0.25, 4, 0.25, 0.25, 365, 131 | | | | | | | Percentage of time that | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Th** | **Tc** | **Total Reward** | **Final Cost** | **Fuel Consumed** | **Average θ** | **Landed** | **Fe** | **Fl** | **Fr** | **\|psi\| > 3** |
| **5** | **1** | -73.82 | -28.02 | 0.59 | 9.17 | -1.00 | 0.99 | 0.15 | 0.08 | 0.96 |
| | **2** | -28.02 | -26.52 | 0.61 | 9.97 | -1.00 | 0.99 | 0.24 | 0.10 | 0.97 |
| | **5** | -17.84 | -31.01 | 0.49 | 13.62 | -1.00 | 0.99 | 0.21 | 0.10 | 0.99 |
| **10** | **1** | -71.18 | -26.06 | 0.67 | -7.85 | -1.00 | 0.99 | 0.16 | 0.23 | 0.98 |
| | **2** | -26.59 | -28.88 | 0.72 | -6.98 | -1.00 | 0.99 | 0.18 | 0.17 | 0.97 |
| | **5** | -16.63 | -30.39 | 0.64 | -6.81 | -1.00 | 0.99 | 0.22 | 0.22 | 0.99 |
| **25** | **1** | -29.87 | -29.56 | 0.59 | -11.42 | -1.00 | 0.99 | 0.22 | 0.34 | 0.96 |
| | **2** | -74.03 | -27.91 | 0.63 | 9.63 | -1.00 | 0.99 | 0.28 | 0.21 | 0.97 |
| | **5** | -18.69 | -30.74 | 0.43 | -13.74 | -1.00 | 0.98 | 0.23 | 0.38 | 0.93 |
| **50** | **1** | -21.21 | -29.98 | 0.52 | -10.67 | -1.00 | 0.99 | 0.44 | 0.44 | 0.95 |
| | **2** | -31.58 | -32.11 | 0.49 | 10.70 | -1.00 | 0.99 | 0.35 | 0.51 | 0.99 |
| | **5** | -17.85 | -30.65 | 0.65 | -11.08 | -1.00 | 0.99 | 0.33 | 0.38 | 0.95 |

If a low $T_c$ is used, then we would effectively be trying to find the best $n_{T_c}$ actions to take given the lowest cost (or highest reward) during the next $T_h$ time steps. If $T_c = 1$, then the solution would be similar, but not equivalent to the greedy policy in RL.

The MPC problem was formulated as suggested in the literature, however, even though the predicted trajectory agreed with the target in all iterations to a certain specified degree, the actions taken did not reflect such behaviour. This is illustrated in Figure 25 overleaf, where the optimized trajectory based on the linearized model agreed with the reference trajectory with a tolerance of $\pm 0.1$. Despite this, the followed trajectory using the returned actions varied drastically from that planned during the next 10 iterations ($T_h = 30, T_c = 10$).

Table 5 clearly shows MPC's unsuccessful attempts, with low total reward and no successful landings. Increasing the penalty for $\varphi$ had little effect, and the principal problem with control lies with the over usage of $\varphi$. Interestingly, being greedy does not guarantee optimal results, on the contrary, the worst results were obtained with $T_c = 1$. This is the same practical problem as the cliff walk example shown in Figure 8, where greedy actions did not necessarily imply success. This also contrasts with the expected result as well as Garcia's suggestion [31] discussed in Section 2.4.5.

The low fuel consumption but high use of $F_E$ suggests that $F_E$ was being used at its lower threshold. Upon investigation, this turned out to be the case, where the optimizer was choosing $F_E \approx 0.5 - 0.6$ for each iteration irrespective of state. However, $F_S$ was being used in an oscillatory fashion to try to maintain the rocket upright. In turn, the problematic use of $\varphi$ negated this effort, causing failures.

### 5.5.3 Conclusion

The complete result with the corresponding hyper parameters is shown in Table 6 overleaf. The failure of MPC can be attributed to multiple variables, including the type of optimizer used. If the system was linear and the objective function was convex, the optimization problem would be convex, opening the entire field of convex programming. However, linearization and no convergence guarantees hindered MPC from being an effective controller. Different targets were used and adjusted to try and mitigate the problem, including using the difference of the target as the actual target. Several hyper parameters were adjusted, but $\varphi$ remained a problem.

Creating a hybrid MPC and $\varphi$-PID controller enables the problem to be defined in a linear manner, guaranteeing observability and controllability in control terms. Although this partially solves the problem of $\varphi$, it also weakens the use of MPC since the predicted states no longer include the dynamics of $\varphi$. RL techniques become even more applicable when such problems are presented, and to that end linear function approximation as well as DDPG models are presented in the next sections.

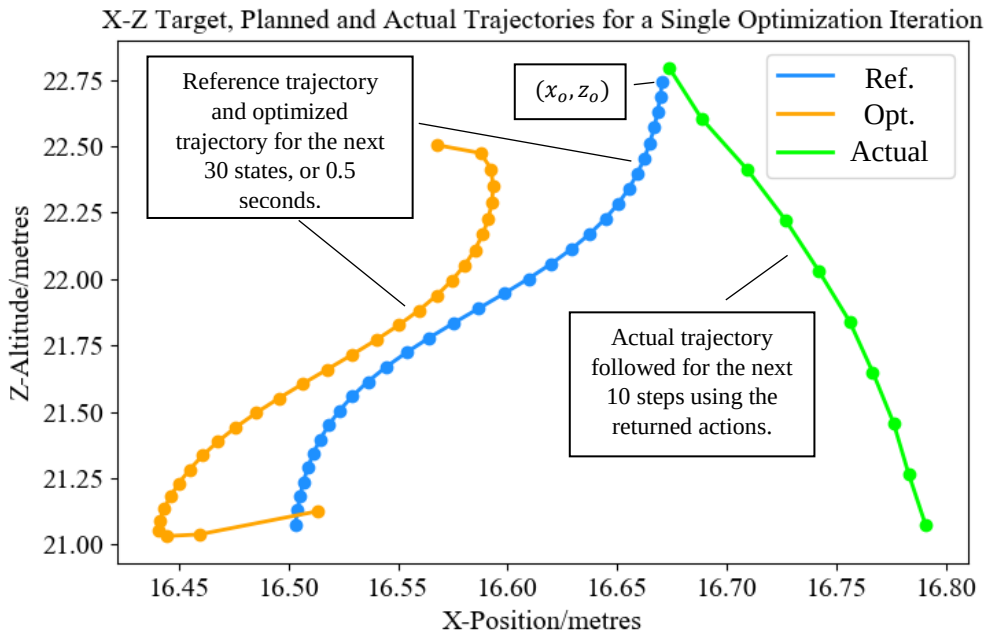



***Figure 25*** Target, planned and actual trajectories followed for a single iteration whilst running Test 1.

***Table 6*** *MPC numerical test results*

| R = [0.1, 0.01, 10], Q = 100*[0.25, 4, 0.25, 0.25, 365, 131, Th = 10, Tc = 5 | | | | | | | Percentage of time that | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | Test Type | Total Reward | Final Cost | Fuel Consumed | Average θ | Landed | Fe | Fl | Fr | \|psi\| > 3 |
| 1 | No Translational Velocity | -18.46 | -34.17 | 0.40 | -13.31 | -1.00 | 0.98 | 0.45 | 0.30 | 0.86 |
| 2 | | -17.28 | -33.84 | 0.33 | 16.92 | -1.00 | 0.98 | 0.48 | 0.46 | 0.93 |
| 3 | | -18.55 | -38.47 | 0.58 | 10.62 | -1.00 | 0.99 | 0.21 | 0.73 | 0.94 |
| 4 | | -16.20 | -32.89 | 0.62 | 13.23 | -1.00 | 0.99 | 0.50 | 0.41 | 0.95 |
| 5 | | -19.56 | -40.01 | 0.29 | -20.00 | -1.00 | 0.98 | 0.24 | 0.73 | 0.88 |
| 6 | x translational velocity varied | -21.09 | -37.95 | 0.36 | -15.65 | -1.00 | 0.98 | 0.69 | 0.29 | 0.90 |
| 7 | | -18.66 | -38.77 | 0.26 | 19.41 | -1.00 | 0.97 | 0.42 | 0.53 | 0.97 |
| 8 | | -19.40 | -38.39 | 0.44 | 11.16 | -1.00 | 0.98 | 0.26 | 0.66 | 0.93 |
| 9 | | -18.09 | -40.94 | 0.29 | 20.04 | -1.00 | 0.98 | 0.61 | 0.37 | 0.88 |
| 10 | | -20.64 | -45.39 | 0.36 | -17.39 | -1.00 | 0.98 | 0.39 | 0.45 | 0.96 |
| 11 | z translational velocity varied | -19.70 | -44.53 | 0.40 | -15.94 | -1.00 | 0.98 | 0.48 | 0.36 | 0.91 |
| 12 | | -18.13 | -44.63 | 0.25 | 21.72 | -1.00 | 0.97 | 0.69 | 0.22 | 0.86 |
| 13 | | -20.34 | -46.41 | 0.36 | -16.16 | -1.00 | 0.98 | 0.29 | 0.69 | 0.92 |
| 14 | | -17.28 | -51.63 | 0.22 | 19.76 | -1.00 | 0.97 | 0.16 | 0.81 | 0.94 |
| 15 | | -19.76 | -47.49 | 0.40 | -18.14 | -1.00 | 0.98 | 0.46 | 0.45 | 0.91 |
| 16 | Initializations with higher theta | -16.80 | -36.78 | 0.22 | 22.82 | -1.00 | 0.97 | 0.32 | 0.58 | 0.97 |
| 17 | | -17.82 | -37.69 | 0.33 | 22.86 | -1.00 | 0.98 | 0.48 | 0.41 | 0.87 |
| 18 | | -16.91 | -40.62 | 0.18 | -23.61 | -1.00 | 0.96 | 0.73 | 0.19 | 0.88 |
| 19 | Added step forces to tests 1-3 | -18.01 | -31.08 | 0.66 | -10.04 | -1.00 | 0.99 | 0.38 | 0.55 | 0.95 |
| 20 | | -17.46 | -34.58 | 0.29 | 18.03 | -1.00 | 0.98 | 0.73 | 0.24 | 0.93 |
| 21 | | -18.29 | -34.97 | 0.29 | -18.65 | -1.00 | 0.98 | 0.73 | 0.24 | 0.88 |
| 22 | Added impulses to tests 1-3 | -19.09 | -29.86 | 0.55 | -8.69 | -1.00 | 0.99 | 0.39 | 0.53 | 0.99 |
| 23 | | -17.86 | -34.67 | 0.36 | 16.53 | -1.00 | 0.98 | 0.49 | 0.49 | 0.92 |
| 24 | | -18.92 | -35.94 | 0.33 | -18.99 | -1.00 | 0.98 | 0.33 | 0.59 | 0.89 |
| **Mean** | | **-18.51** | **-38.82** | **0.37** | **0.69** | **-1.00** | **0.98** | **0.46** | **0.47** | **0.92** |
| **Std. Dev.** | | **1.23** | **5.41** | **0.12** | **17.54** | **0.00** | **0.01** | **0.17** | **0.17** | **0.04** |

## 5.6 Linear Function Approximation Q-Learning Results

The results of two models trained using Main Algorithm 3 outlined in Section 4.4 are presented in this sub-chapter.

The networks differ in the amount of action discretization, with one network having a wider range of values for each action. The objective was to investigate the effect that higher numbers of actions have on both learning and performance. It is intuitive to assume that higher discretization leads to slower learning but better performance. However, possibly simpler control could still be learned by the low-action network, suggesting that the problem can be tackled with simpler methods.

A slight complication arose when defining the actions. The rocket is meant to land using $F_E, F_S, \varphi$ simultaneously. However, the Q-Learning framework was originally devised for discrete, single action environments, such as a game having controls $\{up, down, left, right\}$. An incorrect controller would have been built if we defined the problem in this way since the degrees of freedom of the rocket cannot be controlled by a single action. To this end, actions were defined as combinations stored in tuples:

**Model 1**

$$F_E = \{0.6, 0.8\}$$
$$F_S = \{-1, 0, 1\}$$
$$\varphi = \{-5, -1.5, 0, 1.5, 5\}$$

| | |
|---|---|
| Action 1 | $(0.6, -1, -5)$ |
| Action 2 | $(0.6, -1, -1.5)$ |
| … | … |
| Action $n$ | $(0.8, 1, 5)$ |

Where $n = 30$

**Model 2**

$$F_E = \{0.6, 0.7, 0.75, 0.8, 0.9\}$$
$$F_S = \{-1, 0, 1\}$$
$$\varphi = \{-8, -6, -4, -2, 0, 2, 4, 6, 8\}$$

Where $n = 135$

During learning, state initialization took place using random $\dot{x}, \dot{z}$ limited by thresholds. This was done by simply applying a force to the center of the rocket. Episodic randomness is necessary for learning the control in different states, and to decorrelate states, preventing local optima.

The 10-period moving average learning curve for the high discretization model is shown in Figure 26. In RL, the total reward is the typical performance metric used to judge whether models are in fact learning. This tends to be very noisy since small weight changes affect the policy in multiple states. Although Mnih et al. [48] suggested the use of a more stable metric based on the policy's estimated action-value function Q, this metric was not included in this study.

Learning took place using different learning rates and number of episodes. Figure 26 suggests that the network did not converge to an optimal policy. Recall that one disadvantage of Q-learning is that the model needs to learn the correct policy in every state through multiple episodes. This requires learning on the order of 100,000 episodes. The sudden dips in reward can be due to randomness and unlearnt states. However, the networks show clear signs of learning, with the high action discretization network having a more stable learning curve. This is expected since finer control leads to less abrupt actions with changes in the weights. Recall that the implemented Q-Learning algorithm utilizes off-policy learning, tending towards greedier actions whilst sacrificing convergence.

The initial weights were sampled from a normal distribution having mean 0 and variance 1. The learning rate, α, was set at a constant of 0.001, whereas ε was annealed with every iteration, $n$, according to $\varepsilon . 0.9999^{n}$. This aids greedy behaviour as learning progresses, improving convergence.

Table 8 and Table 7 show the results for the high and low discretization models respectively. One can immediately contrast these against the PID results, where $F_l$ isn't used at all and instead $F_E$ and $\varphi$ are used almost all the time, especially in the absence of disturbances.

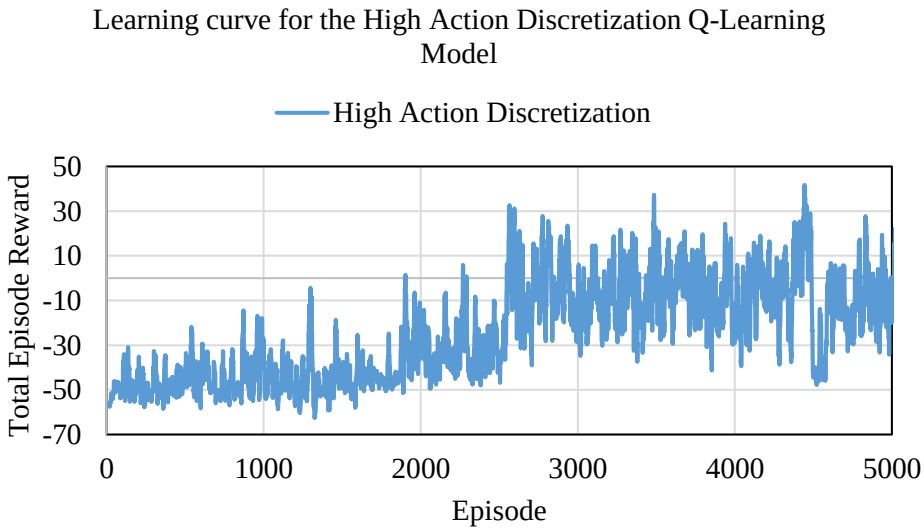



***Figure 26*** Learning curves for the high action discretization showing 5000 episodes of learning.

***Table 8*** *High discretization function approximation test results*

| | | Barge length = 46% of width | | | | | Percentage of time that | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | Test Type | Total Reward | Final Cost | Fuel Consumed | Average θ | Landed | Fe | Fl | Fr | \|psi\| > 3 |
| 1 | No Translational Velocity | 30.06 | -3.99 | 0.99 | -0.47 | 1.00 | 0.36 | 0.00 | 0.14 | 0.00 |
| 2 | | -76.41 | -28.40 | 0.99 | 11.66 | -1.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 3 | | -19.43 | -22.91 | 0.93 | 9.56 | 0.00 | 0.98 | 0.00 | 0.04 | 0.72 |
| 4 | | -81.88 | -32.51 | 1.01 | 7.17 | -1.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 5 | | -77.57 | -32.11 | 0.54 | 10.00 | -1.00 | 0.99 | 0.00 | 0.29 | 0.92 |
| 6 | x translational velocity varied | -12.94 | -15.86 | 0.99 | 5.96 | 1.00 | 0.89 | 0.00 | 0.22 | 0.27 |
| 7 | | -85.08 | -36.08 | 0.44 | 17.97 | -1.00 | 0.98 | 0.00 | 0.00 | 0.98 |
| 8 | | -65.90 | -24.38 | 0.88 | 11.91 | -1.00 | 0.94 | 0.00 | 0.08 | 0.92 |
| 9 | | -87.77 | -40.81 | 0.55 | 18.80 | -1.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 10 | | -83.93 | -36.57 | 0.42 | 12.23 | -1.00 | 0.98 | 0.00 | 0.22 | 0.92 |
| 11 | z translational velocity varied | -59.84 | -13.69 | 0.63 | -9.03 | -1.00 | 0.50 | 0.00 | 0.13 | 0.71 |
| 12 | | -89.86 | -43.56 | 0.37 | 17.66 | -1.00 | 0.98 | 0.00 | 0.00 | 0.98 |
| 13 | | 31.17 | -4.65 | 0.66 | -4.38 | 1.00 | 0.69 | 0.00 | 0.17 | 0.89 |
| 14 | | -92.06 | -48.54 | 0.45 | 18.30 | -1.00 | 0.98 | 0.00 | 0.00 | 0.98 |
| 15 | | -87.04 | -45.34 | 0.51 | 11.13 | -1.00 | 0.99 | 0.00 | 0.25 | 0.90 |
| 16 | Initializations with higher theta | -82.94 | -32.72 | 0.39 | 19.32 | -1.00 | 0.98 | 0.00 | 0.00 | 0.98 |
| 17 | | -82.16 | -37.33 | 0.65 | 19.28 | -1.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 18 | | -73.46 | -26.82 | 0.62 | 5.35 | -1.00 | 0.99 | 0.00 | 0.53 | 0.94 |
| 19 | Added step forces to tests 1-3 | 31.34 | -4.11 | 1.02 | 7.19 | 1.00 | 0.37 | 0.00 | 0.00 | 0.57 |
| 20 | | 30.06 | -4.79 | 0.89 | 8.60 | 1.00 | 0.43 | 0.00 | 0.00 | 0.42 |
| 21 | | 29.90 | -4.24 | 1.00 | -6.17 | 1.00 | 0.65 | 0.00 | 0.21 | 0.56 |
| 22 | Added impulses to tests 1-3 | -53.71 | -10.64 | 1.05 | -2.12 | 0.00 | 0.26 | 0.00 | 0.00 | 0.46 |
| 23 | | -62.41 | -16.49 | 1.03 | 15.34 | 0.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 24 | | -52.88 | -43.35 | 2.63 | -2.23 | -1.00 | 0.31 | 0.00 | 0.08 | 0.31 |
| **Mean** | | **-48.95** | **-25.41** | **0.82** | **8.46** | **-0.38** | **0.80** | **0.00** | **0.10** | **0.77** |
| **Std. Dev.** | | **45.14** | **14.59** | **0.45** | **8.46** | **0.86** | **0.27** | **0.00** | **0.13** | **0.28** |

***Table 7*** *Low discretization function approximation test results*

| | | Barge length = 46% of width | | | | | Percentage of time that | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | Test Type | Total Reward | Final Cost | Fuel Consumed | Average θ | Landed | Fe | Fl | Fr | \|psi\| > 3 |
| 1 | No Translational Velocity | 32.13 | -3.41 | 1.13 | 3.46 | 1.00 | 0.33 | 0.00 | 0.04 | 0.14 |
| 2 | | -16.62 | -21.93 | 0.98 | 2.36 | 0.00 | 0.93 | 0.00 | 0.10 | 0.93 |
| 3 | | -12.71 | -15.58 | 1.08 | 2.90 | 0.00 | 0.94 | 0.00 | 0.08 | 0.58 |
| 4 | | -25.94 | -22.05 | 1.09 | 10.20 | -1.00 | 0.99 | 0.00 | 0.20 | 0.93 |
| 5 | | -76.46 | -32.32 | 0.62 | 12.66 | -1.00 | 0.99 | 0.00 | 0.26 | 0.95 |
| 6 | x translational velocity varied | -68.98 | -25.23 | 0.83 | 13.20 | -1.00 | 0.99 | 0.00 | 0.23 | 0.95 |
| 7 | | -84.77 | -35.30 | 0.43 | 18.10 | -1.00 | 0.98 | 0.00 | 0.00 | 0.98 |
| 8 | | 30.84 | -4.72 | 1.20 | 1.86 | 1.00 | 0.40 | 0.00 | 0.08 | 0.43 |
| 9 | | -88.39 | -40.48 | 0.47 | 18.35 | -1.00 | 0.98 | 0.00 | 0.00 | 0.98 |
| 10 | | -85.42 | -37.38 | 0.66 | 12.21 | -1.00 | 0.99 | 0.00 | 0.16 | 0.96 |
| 11 | z translational velocity varied | 29.29 | -5.01 | 0.65 | 0.12 | 1.00 | 0.26 | 0.00 | 0.05 | 0.62 |
| 12 | | -87.07 | -44.85 | 0.45 | 18.87 | -1.00 | 0.98 | 0.00 | 0.00 | 0.98 |
| 13 | | 32.37 | -4.06 | 0.63 | 2.75 | 1.00 | 0.20 | 0.00 | 0.04 | 0.74 |
| 14 | | -92.44 | -48.19 | 0.53 | 18.54 | -1.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 15 | | -88.70 | -46.30 | 0.56 | 13.31 | -1.00 | 0.99 | 0.00 | 0.22 | 0.95 |
| 16 | Initializations with higher theta | -37.03 | -25.93 | 0.97 | 18.69 | -1.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 17 | | -83.01 | -36.98 | 0.54 | 18.43 | -1.00 | 0.99 | 0.00 | 0.00 | 0.99 |
| 18 | | -73.01 | -27.72 | 0.63 | 4.75 | -1.00 | 0.99 | 0.00 | 0.43 | 0.97 |
| 19 | Added step forces to tests 1-3 | -67.10 | -54.36 | 2.41 | 3.73 | -1.00 | 1.00 | 0.00 | 0.04 | 0.81 |
| 20 | | -72.62 | -19.65 | 1.38 | 12.31 | -1.00 | 0.99 | 0.00 | 0.11 | 0.94 |
| 21 | | -70.26 | -22.86 | 1.07 | 10.61 | -1.00 | 0.99 | 0.00 | 0.20 | 0.97 |
| 22 | Added impulses to tests 1-3 | -77.84 | -47.90 | 2.19 | 1.00 | -1.00 | 1.00 | 0.00 | 0.17 | 0.83 |
| 23 | | -63.24 | -17.09 | 1.13 | 2.58 | 0.00 | 0.77 | 0.00 | 0.14 | 0.95 |
| 24 | | -73.62 | -27.19 | 0.63 | 10.97 | -1.00 | 0.99 | 0.00 | 0.28 | 0.95 |
| **Mean** | | **-50.86** | **-27.77** | **0.93** | **9.66** | **-0.54** | **0.86** | **0.00** | **0.12** | **0.85** |
| **Std. Dev.** | | **42.62** | **14.74** | **0.49** | **6.59** | **0.76** | **0.26** | **0.00** | **0.11** | **0.21** |

It is good to point out that the balancing control was only partially learned by both controllers. Certain states performed much better than others, and the controllers did not display proper control for high values of $\theta$.

Training took place with varying initial conditions to include a certain degree of randomness and enable the models to train in different states, avoiding both local optima and overfitting. Although this is generally a good strategy, states closer to the barge were visited less often due to multiple failures in earlier states. In both models, the rocket could be observed to correctly control its upright position whilst approaching the barge but failed to throttle down for a smooth landing. Since $F_E$ was used consistently, the rocket mass continued to decrease, causing the rocket to accelerate upward due to increased fuel burn.

In tests 2-5 and 22-24 of the high action discretization model, the rocket descended in a controlled manner using $\varphi$ more than $F_S$ to balance, but failed to throttle down. Moreover, it was evident that the controller learned how to adjust for a negative $\theta$ but did not learn the opposite controls. Since learning took place with initializations close to coordinates $(0.5, 1)$, the controller also failed to learn how to navigate from $(0.3, 1)$, especially with $\theta \neq 0$.

The low action discretization was expected to learn simple controls quicker, and this is somewhat evident from the higher number of successful landings. However, as mentioned before, the finer controls of the high action discretization helped the second controller to progress steadily and achieve higher rewards. Even though most tests failed due to over-throttling, the low fuel consumption is worth noting.

From the points above, the following observations and comments can be made on function approximation in RL:

- Different initial conditions must be used and randomized to prevent the model from overfitting to certain states.
- The state passed to the model can be filled with information, but features elements will dominate over others during learning, causing certain weights to have higher values than others.
- Higher discretization led to more stable learning in this case.
- Lowering α from 0.01 to 0.001 and initializing ε (random action probability) to 0.01 instead of 0.1 had a positive effect on learning, achieving higher rewards for the same episode as well as resulting in more stable learning. These observations agree with what was expected.
- Convergence is not guaranteed, and divergence will sometimes happen if too many random variables are introduced.
- The control of the rocket cannot be tuned directly, and this is another disadvantage when compared to classical control algorithms. One solution to this problem is to introduce the concept of *options*, where instead of giving a reward at the end of an episode, a reward is also given when the rocket reaches “milestones” during an episode. This forces the rocket to

navigate its way to certain positions or pick certain controls consistently. The concept of options was not introduced in these tests to keep the reward comparable.

- Introducing the concept of eligibility traces [52] makes learning faster by propagating the reward to not just the current state, but also previous states. This is similar to introducing artificial memory in a model.
- Discretizing the action space for complex continuous control problems is not feasible or scalable.

Note that introducing a value of $F_E < 0.5$ in the actions solves the problem of the rocket accelerating upward after coming close to landing. Intuitively, this cannot happen with the PID, since it follows intuitive and predictable control laws. Figure 27 shows the different trajectories for a few tests corresponding to the results in Table 7.

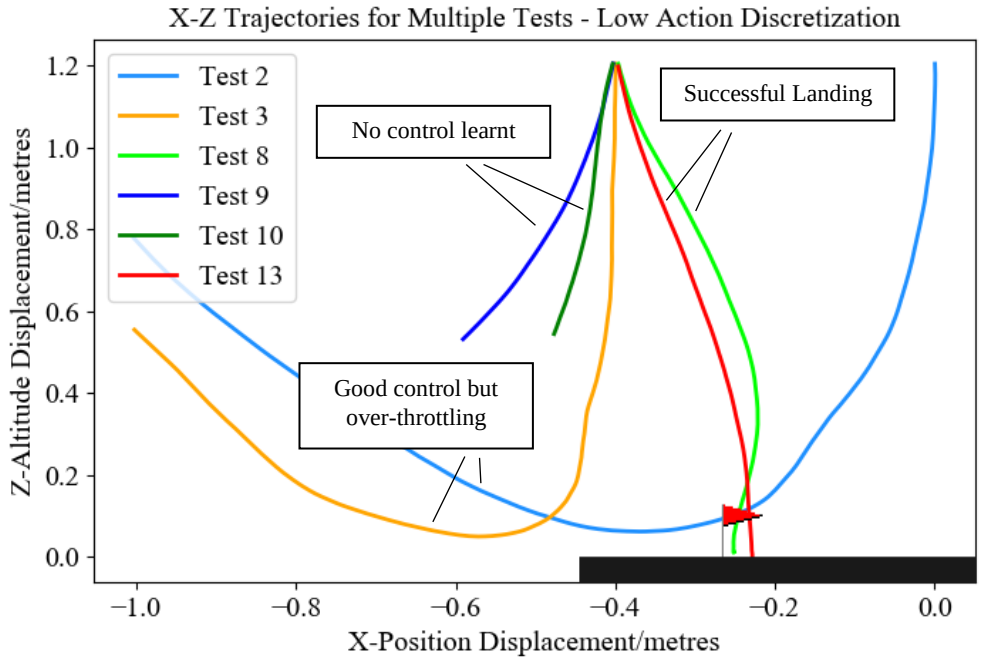



***Figure 27*** Trajectories for different tests of the low action discretization model. (0, 0) represents the landing target.

## 5.7 DDPG Results

The DDPG model is a significant improvement from the discretized function approximation models as well as MPC. It allows a control problem to be tackled directly without compromising action accuracy, whilst still learning optimal control. This sub-section details how the networks were developed and their corresponding training results as well as evaluation results for the 24 tests.

### 5.7.1 Models

Three different models were trained with the same hyper parameters but different input or network layers;

1. Normalized 6-state input $s_{normalized} = [x, \dot{x}, z, \dot{z}, \theta, \dot{\theta}]$
    - This involved executing a simulation before actual learning or testing commenced, to sample 10000 values for each state element. This was then used to normalize each state during execution by removing the mean and scaling $s$ to have unit variance.
    - This model was trained using a maximum of 500 steps per episode. This was done to force the network to land the rocket quickly but in a controlled manner.

2. Unnormalized 14-state input:

$$s = [\, x, \dot{x}, z, \dot{z}, \theta, \dot{\theta}, fuel\ consumed, mass, \ldots$$
$$\ldots leftmost\ barge\ coordinates\ (2), rightmost\ barge\ coordinates\ (2), \ldots$$
$$\ldots landing\ coordinates(2)]$$

    - The latter coordinates were used in case the barge was moved. The fuel consumption as well as changing mass allow the network to model even $\dot{m}$, a state which was ignored in all previous models.
    - The number of steps was not limited during training.

3. Same as (2) but implemented Batch Normalization across layers as suggested by Lillicrap et al. [43].

Each model required two feed forward neural networks together with two copies of those networks as described in Section 4.5.1, all built using Tensorflow 1.2.0. Note that the networks are not restricted to be feed-forward, in fact, a Recurrent Neural Network (RNN) is worth exploring and is listed in the improvements.

Each actor and critic network consisted of 3 layers; one 400 neuron layer, one 300 neuron layer and the final output equivalent to the required dimension number, 3 for the actor and 1 for the critic. The $relu$ activation function was used in the first layer, followed by the sigmoid function and finally $tanh$ in the case of the actor. The output

of the critic was not passed through an activation function since it is meant to represent $Q(s, a|\theta)$.

The $relu$ function allows for linear parts to be captured, whereas the sigmoid function captures non-linearities. The final $tanh$ output limits the output to be between 1 and -1, serving as a convenient limiter.

### 5.7.2 Learning

Preliminary tests showed that diversifying activation functions allowed the network to learn faster, whereas using a lower number of neurons did not have any noticeable effect unless a number lower than approximately $5\times(Number\ of\ input\ states)$ was used. The Adaptive Moment Estimation (Adam) was used as a gradient optimizer required for backpropagation of the error.

The learning rate was varied starting from a relatively high value of 0.01 for both networks and decreasing by an order of magnitude if training seemed to express divergence properties. Eventually, the actor's learning rate was held at 0.0001, whereas the critic's was 0.001. Low learning rates allow for stable but slower learning, and this proved to be the case. $\tau$, the proportion with which weights of copied networks were moved towards the actual networks was kept at 0.001.

The above justifications allowed the focus to shift on investigating the effects that different inputs, as well as normalization, have on control. All simulation parameters were kept the same as the Q-learning cases, including initial conditions, barge length, and randomness.

The first two models proved to be very effective and learning took place as expected, as shown in Figure 28. However, the third model utilising Batch Normalisation

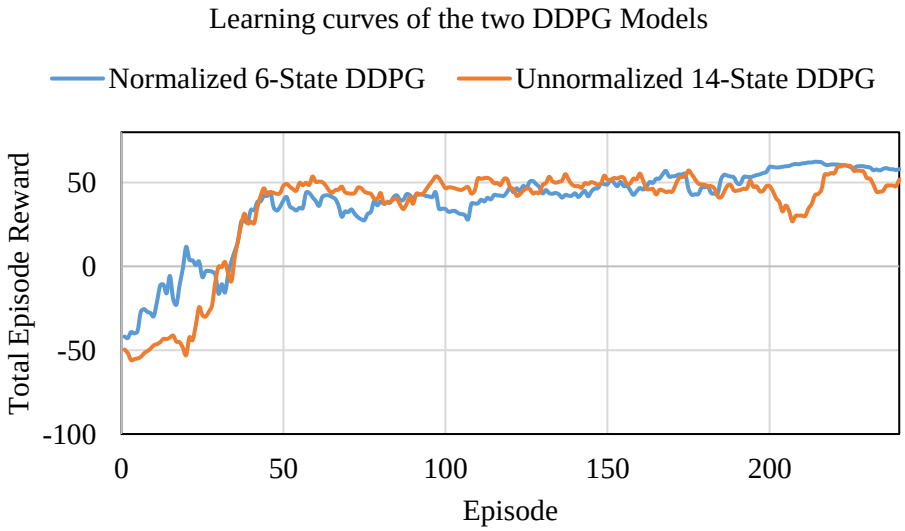



***Figure 28*** Learning curves for the two DDPG models.

slowed down learning significantly, even when higher learning rates were used. Batch Normalisation layers did not seem to benefit the networks in any way, on the contrary, performance degraded. To this end, only the results of the first two networks are presented.

Learning of the first two networks shown in Figure 28 was much quicker and smoother than anticipated, especially when compared to the function approximation models. After just 50 episodes, less than 10 minutes of training time, both DDPG models showed signs of convergence, surpassing the numerical results of the classical PID controller. Fast learning can be attributed back to the Replay Buffer as well as separation of networks.

Consider the results shown in Table 9 and Table 10. The following aspects stand out;

- Relatively low fuel consumption across almost all tests;
- Successful landing consistency;
- Relatively low use of $F_E$;
- Use of other actions when required;
- $|\theta_{average} < 3^o|$ across more than 90% of tests;
- Best numerical values for *Final Cost* across all models as well as *Average Total Reward* for the first model.

Interestingly, both models learn very well as is evident from Figure 28, however, they also perform quite differently. Model 2 learned to use $F_E$ more often (outputting higher values to make $tanh(final\ layer) > 0$) whilst still achieving lower fuel consumption. On top of this, even though the number of successful landings of model 2 was not as much as that of model 1 or the PID, the *Final Cost* and *Total Reward* were higher where it was successful. This suggests that model 2 learnt specific controls quite well for different states, increasing the final efficiency of the controller. Compared to the function approximation models, the DDPG networks did not settle at a local optimum and converged successfully *for the states in which the networks were trained in.*

Even though the use of normalization and 6 states proved to be sufficient in model 1, it is clear that adding additional fuel, mass, and positioning information helps the controller converge to higher efficiency. Manual normalization still proved to be beneficial, even though random state samples were taken before training and were never updated.

Batch Normalization was thought to be an improvement over the manual normalization since it automatically removes any bias from inputs to layers, but this was not the case. An improvement over manual normalization could, therefore, be to update the sample statistics and include the full 14 states.

Note that the barge and landing coordinates did not change in the tests, however, their inclusion in model 2 makes it more robust against disturbances associated with the barge.

***Table 9*** *DDPG Model 1 numerical test results*

| | | Barge length = 46% of width | | | | | Percentage of time that | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | Test Type | Total Reward | Final Cost | Fuel Consumed | Average θ | Landed | Fe | Fl | Fr | \|psi\| > 3 |
| 1 | No Translational Velocity | 25.76 | -6.32 | 1.67 | -0.37 | 1.00 | 0.34 | 0.05 | 0.04 | 0.38 |
| 2 | | 26.25 | -5.91 | 1.62 | -0.03 | 1.00 | 0.42 | 0.14 | 0.01 | 0.46 |
| 3 | | 27.22 | -5.60 | 1.84 | -0.36 | 1.00 | 0.47 | 0.14 | 0.06 | 0.51 |
| 4 | | 27.37 | -5.53 | 1.85 | -0.22 | 1.00 | 0.49 | 0.14 | 0.05 | 0.52 |
| 5 | | 26.80 | -5.80 | 1.85 | -1.02 | 1.00 | 0.40 | 0.11 | 0.07 | 0.38 |
| 6 | x translational velocity varied | 25.13 | -6.14 | 1.63 | -0.17 | 1.00 | 0.45 | 0.10 | 0.11 | 0.45 |
| 7 | | -8.39 | -22.00 | 0.59 | 3.87 | 1.00 | 0.69 | 0.81 | 0.09 | 0.92 |
| 8 | | 24.55 | -6.36 | 1.79 | -0.20 | 1.00 | 0.47 | 0.14 | 0.17 | 0.45 |
| 9 | | 24.83 | -6.06 | 2.37 | -4.98 | 1.00 | 0.42 | 0.17 | 0.03 | 0.43 |
| 10 | | 25.79 | -5.94 | 2.16 | -0.72 | 1.00 | 0.43 | 0.16 | 0.02 | 0.44 |
| 11 | z translational velocity varied | -10.73 | -13.87 | 1.47 | -0.42 | -1.00 | 0.91 | 0.22 | 0.41 | 0.68 |
| 12 | | -21.74 | -36.34 | 0.45 | 4.76 | 1.00 | 0.71 | 0.78 | 0.03 | 0.96 |
| 13 | | -39.75 | -35.96 | 1.65 | -5.28 | -1.00 | 0.99 | 0.50 | 0.36 | 0.85 |
| 14 | | 24.44 | -6.72 | 1.28 | -2.59 | 1.00 | 0.45 | 0.46 | 0.00 | 0.66 |
| 15 | | 26.37 | -5.50 | 1.08 | -1.20 | 1.00 | 0.32 | 0.36 | 0.00 | 0.59 |
| 16 | Initializations with higher theta | 26.26 | -5.43 | 1.42 | 0.87 | 1.00 | 0.32 | 0.16 | 0.01 | 0.38 |
| 17 | | 26.15 | -6.03 | 2.18 | 0.04 | 1.00 | 0.43 | 0.14 | 0.03 | 0.46 |
| 18 | | 25.83 | -6.33 | 1.76 | -2.53 | 1.00 | 0.48 | 0.06 | 0.15 | 0.52 |
| 19 | Added step forces to tests 1-3 | 25.94 | -6.04 | 1.67 | -0.14 | 1.00 | 0.41 | 0.13 | 0.03 | 0.42 |
| 20 | | 26.21 | -6.23 | 1.67 | -0.03 | 1.00 | 0.34 | 0.05 | 0.03 | 0.37 |
| 21 | | 25.70 | -6.39 | 1.70 | -0.49 | 1.00 | 0.34 | 0.03 | 0.05 | 0.35 |
| 22 | Added impulses to tests 1-3 | 26.16 | -6.06 | 1.63 | 0.22 | 1.00 | 0.41 | 0.12 | 0.02 | 0.39 |
| 23 | | 25.61 | -6.06 | 1.68 | 0.03 | 1.00 | 0.38 | 0.13 | 0.02 | 0.40 |
| 24 | | 25.50 | -6.16 | 1.66 | -0.39 | 1.00 | 0.32 | 0.10 | 0.03 | 0.29 |
| **Average** | | **18.22** | **-9.53** | **1.61** | **-0.47** | **0.83** | **0.48** | **0.22** | **0.08** | **0.51** |
| **Std. Dev.** | | **17.90** | **8.75** | **0.42** | **2.07** | **0.55** | **0.17** | **0.21** | **0.10** | **0.18** |

***Table 10*** *DDPG Model 2 numerical test results*

| | | Barge length = 46% of width | | | | | Percentage of time that | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | Test Type | Total Reward | Final Cost | Fuel Consumed | Average θ | Landed | Fe | Fl | Fr | \|psi\| > 3 |
| 1 | No Translational Velocity | 24.88 | -3.49 | 1.05 | -0.26 | 1.00 | 0.15 | 0.00 | 0.04 | 0.11 |
| 2 | | 24.44 | -3.57 | 1.10 | 0.16 | 1.00 | 0.15 | 0.02 | 0.02 | 0.12 |
| 3 | | -10.68 | -14.33 | 1.16 | 0.27 | 0.00 | 0.90 | 0.00 | 0.09 | 0.63 |
| 4 | | -10.91 | -14.38 | 1.17 | 0.44 | 0.00 | 0.90 | 0.07 | 0.11 | 0.68 |
| 5 | | -9.54 | -13.19 | 1.20 | -0.08 | 0.00 | 0.89 | 0.10 | 0.24 | 0.68 |
| 6 | x translational velocity varied | 21.45 | -4.74 | 1.06 | 0.05 | 1.00 | 0.15 | 0.00 | 0.08 | 0.11 |
| 7 | | 29.79 | -5.45 | 0.74 | 2.74 | 1.00 | 0.63 | 0.31 | 0.11 | 0.64 |
| 8 | | 28.66 | -6.50 | 1.19 | -0.17 | 1.00 | 0.82 | 0.00 | 0.40 | 0.64 |
| 9 | | -12.73 | -19.24 | 0.91 | 2.67 | -1.00 | 0.91 | 0.60 | 0.18 | 0.80 |
| 10 | | -8.74 | -16.91 | 1.10 | 0.34 | -1.00 | 0.99 | 0.35 | 0.05 | 0.77 |
| 11 | z translational velocity varied | 31.71 | -4.48 | 1.09 | 0.47 | 1.00 | 0.79 | 0.00 | 0.50 | 0.59 |
| 12 | | 28.06 | -6.16 | 0.82 | 2.10 | 1.00 | 0.67 | 0.26 | 0.12 | 0.59 |
| 13 | | 29.15 | -6.41 | 1.11 | -0.21 | 1.00 | 0.77 | 0.00 | 0.50 | 0.60 |
| 14 | | -13.27 | -19.69 | 0.99 | 2.29 | -1.00 | 0.99 | 0.59 | 0.12 | 0.76 |
| 15 | | -10.00 | -18.21 | 1.01 | 0.28 | -1.00 | 0.99 | 0.38 | 0.10 | 0.78 |
| 16 | Initializations with higher theta | 29.88 | -3.33 | 0.72 | 3.63 | 1.00 | 0.49 | 0.26 | 0.27 | 0.55 |
| 17 | | -10.39 | -21.87 | 0.74 | 3.28 | 0.00 | 0.81 | 0.45 | 0.43 | 0.97 |
| 18 | | 18.58 | -5.92 | 1.11 | 1.66 | 1.00 | 0.13 | 0.04 | 0.08 | 0.13 |
| 19 | Added step forces to tests 1-3 | 26.19 | -3.15 | 1.05 | 0.09 | 1.00 | 0.14 | 0.00 | 0.01 | 0.10 |
| 20 | | 24.67 | -3.15 | 1.10 | 0.16 | 1.00 | 0.15 | 0.02 | 0.05 | 0.13 |
| 21 | | 32.58 | -4.02 | 1.10 | -0.34 | 1.00 | 0.80 | 0.09 | 0.35 | 0.65 |
| 22 | Added impulses to tests 1-3 | 32.49 | -3.26 | 1.07 | -0.23 | 1.00 | 0.68 | 0.00 | 0.03 | 0.50 |
| 23 | | 34.27 | -3.67 | 1.11 | 0.41 | 1.00 | 0.81 | 0.10 | 0.09 | 0.61 |
| 24 | | 33.26 | -3.82 | 1.09 | -0.23 | 1.00 | 0.81 | 0.12 | 0.25 | 0.67 |
| **Average** | | **15.16** | **-8.70** | **1.03** | **0.81** | **0.50** | **0.65** | **0.16** | **0.18** | **0.53** |
| **Std. Dev.** | | **18.70** | **6.33** | **0.14** | **1.23** | **0.76** | **0.31** | **0.19** | **0.15** | **0.26** |

### 5.7.3 Trajectory Comparison between Models 1 and 2

Consider the trajectories in Figure 29 shown for both DDPG models. The chosen tests will also serve as a comparison with the PID controller and cover different tests. The trajectories bring out some very interesting distinctions in the type of control that the models learned.

Starting from Test 2, the initial trajectories were the same, suggesting that some similarities do exist between the networks. However, model 2 made use of large values of $\varphi$ quite sparingly, and $F_E$ only needed to be used 15% of the time, even for a smooth landing. The low final cost reflects such a landing, as well as its accuracy. So why did the networks learn different things in certain states? Recall that the actor's aim is to output an action given a state and the given weights, and a single weight change affects multiple states. Therefore, even initializing the same network with different weights will lead to slightly different behaviours. Moreover, the inputs were different, leading to different converged solutions.

In Test 8, model 1 overshot the target but still landed successfully, exhibiting low $\dot{x}$. In this case, the second model made more use of $F_E$ and $\varphi$ and landed with approximately the same accuracy but on the opposite side of the barge. Considering their control, model 2 is once again preferred due to a more stable trajectory. Together with Test 18, Test 17 suggests that model 2 requires more training when it comes to initializing the rocket at different $x$ coordinates, whereas model 1 should decrease its over-control that causes overshoots. Perhaps the most interesting is Test 23, where an impulse response was applied. Clearly, the impulse affected model 1 more than model 2, however, it still managed to recover soon after, following the

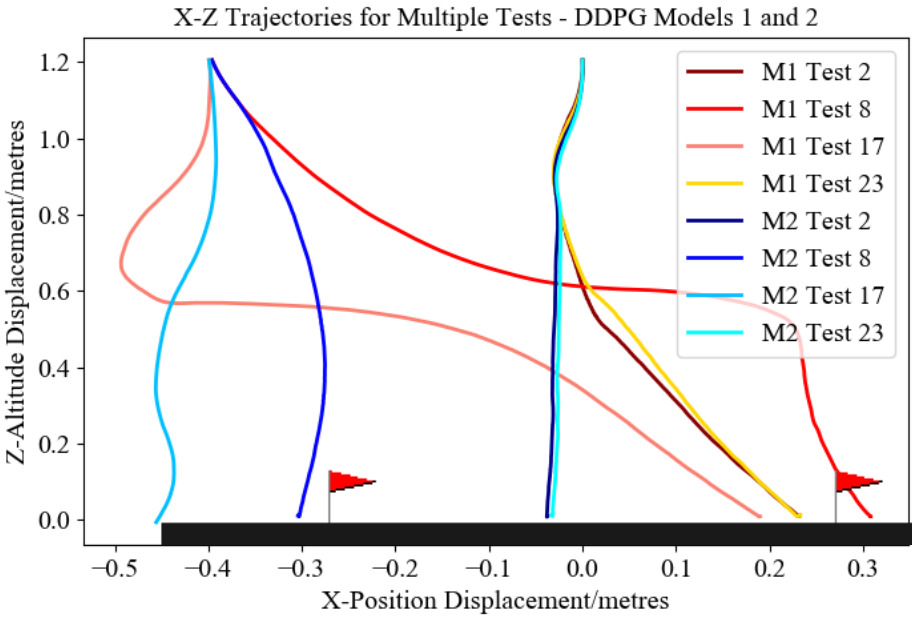



***Figure 29*** Trajectory comparison between the two DDPG models for Tests 2, 8, 17 and 23

trajectory learned in Test 2. Impressively, model 2 remained almost unaffected by the disturbance, suggesting better disturbance rejection properties.

In summary;

- Both model 1 and model 2 learned how to control the rocket in a stable manner similar to the PID. However, some states required more training.
- Both models exhibited rough landings, having higher $\dot{z}$ on touchdown. This can easily be amended by giving a reward to the networks for low values of $\dot{z}$ when close to the barge.
- Model 1 learned how to navigate much better than model 2, however, the controller tended to lower $F_E$ when close to the landing barge, leading to a rougher landing.
- Model 2 seemed to ignore the target when the simulation was started at $x = 0.3$. This is because the model was not trained on enough episodes with $x \neq 0$. The control capability of model 2 surpasses both the PID and model 1.
- Given an infinite amount of fuel, both can hover indefinitely and robustly, surpassing the PIDs capability.

## 5.8 Section Conclusion

This chapter was concerned with the evaluation of 4 different controllers proposed in Section 4; PID, MPC, Q-Learning, and DDPG. The testing conditions, as well as evaluation metrics, were first laid out. The chosen tests represent a variety of conditions that test different aspects of each controller, such as the ability to move whilst still staying upright, the capacity to land and fuel consumption, among others.

Results showed that the PID performed quite well, exhibiting soft and constant landings. However, fuel consumption and the inability to recover from large values of $\theta$ are two major defects. The latter is evident from Test 17 as shown in Figure 30. The fuel consumption problem can be easily adjusted through either $K_D$ of the PID controlling $F_E$, or by offsetting the $z$ target. Nonetheless, in two dimensions, the PID sufficed to be a robust controller.

The PID disregards the state of the system, and only concerns itself with a single output. When multiple SISO systems are put together, controls can conflict. To handle MIMO systems, MPC was introduced. However, results for MPC proved to be inconsistent and divergent, with the controller exhibiting good control at times but very bad in others. This proved to be the case for different initial conditions as well as different hyper parameters. Changing the cost matrices did change the system's behaviour, but the controller failed to control $\varphi$ in the intended manner. The time and control horizons had a drastic effect on the final reward, and surprisingly, increasing the control horizon improved the results. Nonetheless, the surprising failure of MPC to solve non-linearities whilst preserving control served as a stepping stone towards exploring AI techniques.

Function approximation RL was discussed in Section 4.4, where Q-Learning was presented as an episodic alternative to learning control. Two controllers were built, one having low action discretization, whilst the other having higher action discretization and more state information. These controllers were designed to highlight the limitations of the tabular approach to tackle continuous state-action problems. Results showed that neither controller achieved precise, consistent landings with clear executed control.

To tackle the control problem in its entirety, the DDPG algorithm was implemented. The implementation was evaluated with the normalized and unnormalized state cases. Batch Normalization was also used, but surprisingly the networks showed poor performance, possibly due to very slow learning. Although numerical results for model 1 and model 2 were similar, the unnormalized case (model 2) provided softer landings, more control and in some situations, better trajectory choices. However, the first model learned how to navigate from $x \neq 0$ better than the second. Figure 31 shows different successful and unsuccessful landings achieved by model 2.

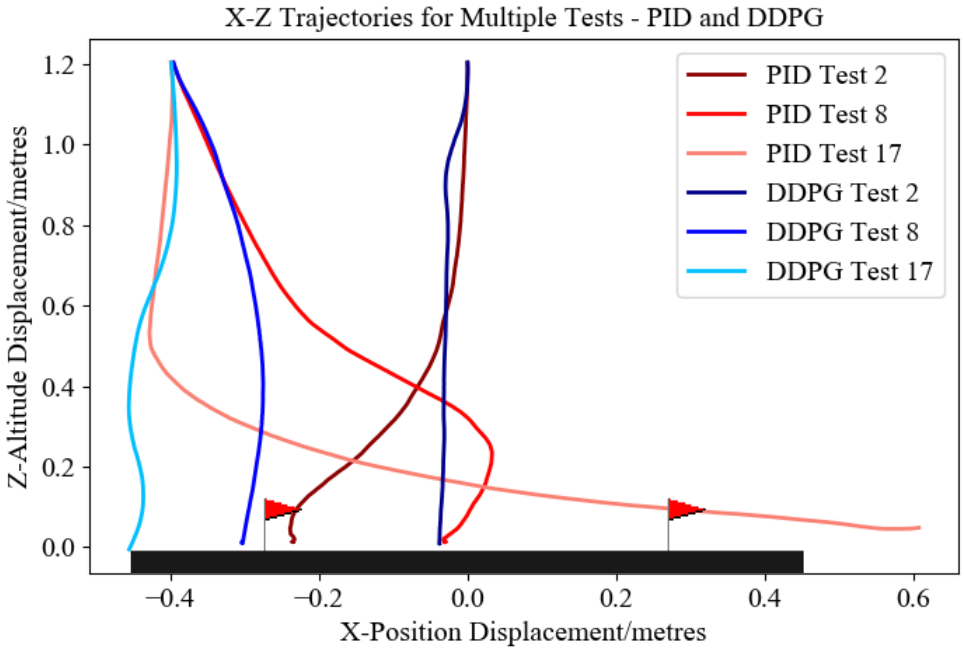



***Figure 30*** Trajectory comparison between the DDPG model and the PID controller for Tests 2, 8 and 17. Together, these tests show a variety of conditions and highlight the differences between controllers.

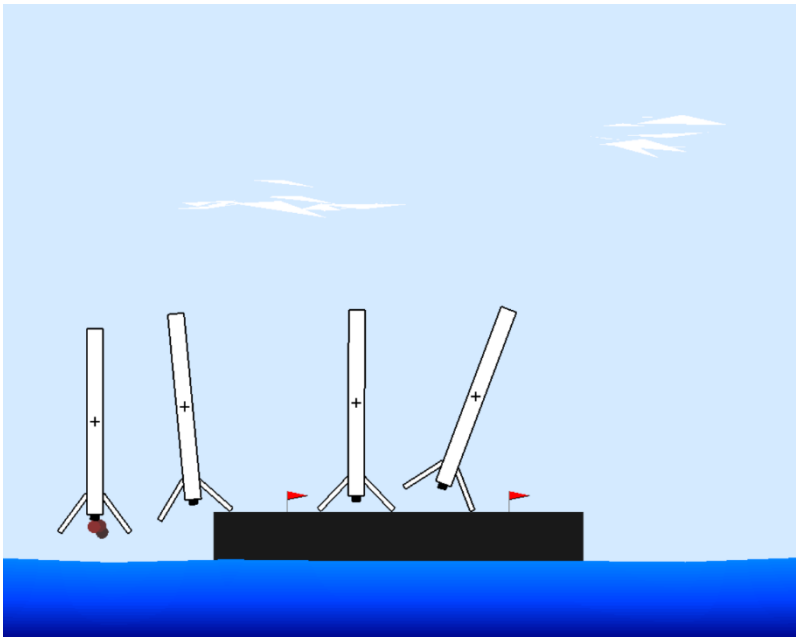


***Figure 31*** Different landings for the unnormalized state DDPG.

Compared to the PID controller, the DDPG technique proved to be very efficient, for instance, the trajectories in Figure 30 show why the DDPG controllers obtained lower fuel consumption. The ability of the DDPG to learn such complex control simply through rewards frees the control designer from explicitly defining the system and requiring the full model. This makes DDPG viable to not just the rocket landing problem, but all continuous control problems, such as the inverted pendulum. The fact that both DDPG models achieved proper control suggests that normalization may only be needed where states have very wide and different ranges. Moreover, the state should include all critical information related to the dynamics of the model.

Despite so many advantages, RL still has its pitfalls. In this case, rewards were explicitly defined, but the input transient responses were not. The response of certain inputs might need to be controlled as shown in the PID root locus design, meeting certain frequency and time criteria. Allocating rewards to control such transients becomes ambiguous and is not scalable. Moreover, the reward is assumed to be provided by the environment, but certain problems might involve having an unobservable reward. Partially observable problems can be defined to solve these problems, but their discussion is out of the scope of this thesis.

The simulation and results are briefly summarized in a short video by following this link: https://goo.gl/tmRwK4. All code associated with the simulation was uploaded to GitHub and can be found at: https://goo.gl/GJpPAh.

# 6 Improvements and Future Work

Continuous control problems are open ended, as such improvements and suggestions will be limited to methods that build on the implemented controllers. The PID serves as a sufficient benchmark and no further development is needed on that front unless it's critical to the subject of study.

- In this thesis, the *LQR* was simply used as a foundation for optimization theory. However, its inability to include constraints led to MPC. On this note, the Riccati equation used for the normal LQR method can be re-derived with the included constraints, giving rise to constrained LQR with integral action. Even though theoretically MPC indirectly includes LQR in its implementation, LQR's algebraic solution makes it an attractive solution over the optimization counterpart. Therefore, several constrained LQRs can be used in different conditions to avoid the use of an optimizer.

- *Optimal control* is a very wide field having many different forms. One such branch lies in the use of different optimizers to solve different problems. Different optimizers differ in the algorithm they use to converge to a solution and their use must be limited to solving that type of problem. Using a convex optimizer to try to solve a non-convex problem leads to instability and divergence. As such, more research should be done on the type of optimizer used. The objective function and the linearized problem were tackled in the correct way, but redefining the problem in a more sophisticated optimization framework can possibly yield much better results. Moreover, hybrid controllers that split the optimization problem into multiple steps should be explored as an improvement over a single MPC controller.

- Given the effectiveness of DDPG and its limitations, more continuous action controllers should be explored. Unlike feedforward NNs, RNNs can consider a sequence of inputs, replicating artificial memory. Such a network might prove useful in both actor and critic networks, especially since the nature of the problem is sequential.

  Besides this, very recent studies on evolutionary networks (ENs) have prompted renewed interest in its applicability to control algorithms [53]. ENs serve as black box optimizers that replace the RL framework such as Q-Learning. The objective is still to maximize a fitness function, however, several simulations are executed in parallel and weights are adjusted by a gradient function weighted by the rewards obtained in those simulations. Thus, small incremental changes are made to the weights. Although convergence is not guaranteed, more advanced evolutionary strategies, such as the Covariance Matrix Adaptation [54], also provide an alternative to non-linear optimization.

# 7 Conclusion

This multi-disciplinary project started with developing a simulation using a physics engine in Python 3.5 to tackle the non-linear problem of VTOL of a rocket using a vectorised nozzle. Vectorised nozzles provide much more control over their static counterparts and have enabled the reusability of first stage rockets, saving millions of dollars in material costs.

The control problem was first presented as a second order differential equation having 6 features; $s = [x, \dot{x}, z, \dot{z}, \theta, \dot{\theta}]$. These fully captured the dynamics of the simulation, with the exception of a few variables, such as wind drag and change in mass. The simulation was built using Box2D with a combination of basic shapes held together by joints. The rocket was modelled after SpaceX's Falcon 9, and attention was given to the accuracy of dimensions and variables, such as mass and scale.

The goal of the project was to design a robust controller for vertical rocket landing. To do so, controllers from different domains were reviewed and 4 controllers were implemented. The classical control approach involved 3 PID controllers tuned independently. These were designed by assuming independence among variables and treating each output as a SISO system, having one input controlling one output. In reality, this isn't true, as coupling exists between multiple inputs and multiple outputs. Root locus was used to design a single PID in the time domain, with subsequent PIDs copied, scaled and adjusted empirically until suitable control was achieved. Moreover, other states were then introduced as feedback in each PID to try to eliminate any decoupling between states.

The PID could be used because the problem is 2-dimensional, simplifying the kinematics significantly. Had the problem been tackled in 3-dimensions, a hybrid controller would have been used as a benchmark, having a PID control a subset of controls, and an MPC controlling others. Nonetheless, the decoupled PID proved to be robust for low angles of $\theta$ and small position offsets. It had the softest landings but highest fuel consumption. However, thanks to the intuitiveness of the algorithm, this can be very easily adjusted. This is in contrast with most AI algorithms, where an entire model needs to be retrained with every change that occurs.

Although classical control is still widely used in the industry, optimal control theory has been successfully adopted due to the increased computational capability of embedded systems. To this end, MPC was explored and a controller was implemented using the CVX framework.

This domain minimizes a utility function given a number of constraints. Constraints include input constraints which help keep the inputs under thresholds. The rest of the constraints involve replicating the state space model in the form $x_{t+1} = Ax_t + Bu_t$.

With a prediction horizon of $T_h$, we would effectively be trying to predict future states and inputs. This is very advantageous over classical control, since consideration is given to future states as well as current states. Although theoretically sound, in practice, MPC experienced convergence issues. This can be due to linearization and non-convexity. Although $F_s$ was being controlled as intended, $\varphi$ wasn't.

Function approximation Q-Learning was used as a bridge between optimal control theory and RL, two fields which derive from the same background. Two controllers were implemented by discretizing the action-space and approximating the state as an algebraic sum of features, each having learnable weights. Controllers managed to exhibit good control in some states, but training led to local optima. Consequently, testing showed that on most conditions, the controllers failed simply because the combination of limited inputs and multiple states was not enough to sustain control.

The final significant contribution made in the thesis was the use of DDPG and the framework required for it. DDPG solves the discrete action space limitation of Q-Learning controllers as well as the state approximation by using two separate NNs to implement the actor-critic architecture. NNs are capable of emulating a state on a continuous level. Although somewhat complex, DDPG managed to obtain the highest efficiency and best overall control, utilising $\varphi$ and $F_S$ in a much better manner than the MPC or Q-Learning. The use of RNNs was proposed as a replacement over feedforward NNs. Theoretically, these would model the sequential inputs, taking into consideration past states.

In summary, all objectives of the thesis were successfully achieved;

- A new, easy-to-use rocket landing simulation framework was developed and contributed as Open Source software on OpenAI.
    - Any new controller can be tested with the use of an Application Programming Interface.
    - A new controller can be evaluated against any developed controller without any changes to the code.
    - Changes to the shape of the rocket, dynamics, environment or even simulation objectives can be easily done.
- The control framework was laid out for both classical control and optimal control, with any controller capable of being benchmarked.
- RL framework was successfully implemented using two controllers, serving as a stepping stone for more advanced algorithms.
- Deep learning framework was successfully implemented and contrasted with traditional control.

# 8 References


[1] J. Gardi and J. Ross, “An Illustrated Guide to SpaceX’s Launch Vehicle Reusability Plans.” [Online]. Available: http://www.justatinker.com/Future/.

[2] K. J. Kloesel, J. B. Pickrel, E. L. Sayles, M. Wright, and D. Marriott, “First Stage of a Highly Reliable Reusable Launch System,” *AIAA Sp. 2009 Conf.*, no. September, pp. 1–17, 2009.

[3] SpaceX, “SES-10 Mission,” no. March, pp. 9–10, 2017.

[4] SpaceX, “Falcon 9.” [Online]. Available: http://www.spacex.com/falcon9.

[5] “Blue Origin | Our Approach to Technology.” [Online]. Available: https://www.blueorigin.com/technology. [Accessed: 07-Apr-2017].

[6] G. Sutton and O. Biblarz, “Thrust Vector Control System,” in *Rocket Propulsion Elements*, 7th ed., vol. 17, no. 5, 1980, pp. 407–412.

[7] K. Lauer, “Box2D.” [Online]. Available: https://pypi.python.org/pypi/Box2D/2.0.2b1.

[8] SpaceX, “Falcon User’s Guide,” p. 69, 2015.

[9] M. M. Ragab *et al.*, “Launch Vehicle Recovery and Reuse,” *Am. Inst. Aeronaut. Astronaut.*, pp. 1–10.

[10] NASA, “Gimbaled Thrust,” *Beginner’s Guide to Rockets, Public Domain Source*. [Online]. Available: https://spaceflightsystems.grc.nasa.gov/education/rocket/gimbaled.html. [Accessed: 07-Apr-2017].

[11] J. J. Isaac and C. Rajashekar, “Fluidic Thrust Vectoring Nozzles,” no. April, 2014.

[12] SpaceX, “Launch Manifest.” [Online]. Available: http://www.spacex.com/missions. [Accessed: 07-Apr-2017].

[13] N. Chris Bergin, “SpaceX’s Autonomous Spaceport Drone Ship ready for action.” [Online]. Available: https://www.nasaspaceflight.com/2014/11/spacex-autonomous-spaceport-drone-ship/. [Accessed: 07-Apr-2017].

[14] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback control of Dynamic Systems (6th Ed.)*. 1994.

[15] K. Ogata, *Modern control engineering*, 5th ed. Prentice Hall, 2010.

[16] K. M. Hangos, J. Bokor, and G. Szederkényi, “State-space Models,” in *Analysis and Control of Nonlinear Process Systems*, 2004.

[17] D. I. Wilson, “Advanced Control using MATLAB or Stabilising the unstabilisable,” p. 576, 2014.

[18] A. Mohammadbagheri, N. Zaeri, and M. Yaghoobi, “Comparison Performance Between PID and LQR Controllers for 4-leg Voltage-Source Inverters,” *2011 Int. Conf. Circuits, Syst. Simul. IPCSIT*, vol. 7, pp. 230–234, 2011.

[19] V. Bobal, J. Machacek, and R. Prokop, “Tuning of digital PID controllers based on Ziegler-Nichols method,” *New Trends Des. Control Syst. 1997*, 1998.

[20] K. Ogata, “PID Controllers and Modified PID Controllers,” in *Modern Control Engineering*, 5th ed., Prentice Hall, pp. 567–572.

[21] Y.-L. Yang and T.-Y. Hsu, “A Dynamic Model for Two-Dimensional Thrust-

Vectoring Nozzles," no. July, 2005.
[22] S. Boyd and L. Vandenberghe, *Convex Optimization*, vol. 25, no. 3. 2010.
[23] R. M. Murray, "LQR control," *Control Dyn. Syst.*, pp. 1–14, 2006.
[24] K. J. Åström and R. M. Murray, "Linear Quadratic Regulators," in *Feedback Systems: An Introduction for Scientists and Engineers*, 2008, pp. 192–196.
[25] M. S. Triantafyllou and F. S. Hover, "Linear Quadratic Regulator," in *Maneuvering and Control of Marine Vehicles*, 2003, pp. 92–98.
[26] E. Vinodh Kumar and J. Jerome, "Robust LQR controller design for stabilizing and trajectory tracking of inverted pendulum," *Procedia Eng.*, vol. 64, pp. 169–178, 2013.
[27] Y. Jung and H. Bang, "Mars precision landing guidance based on model predictive control approach," *Proc. Inst. Mech. Eng. Part G J. Aerosp. Eng.*, vol. 230, no. 11, pp. 2048–2062, 2016.
[28] M. Fawzy, M. A. S. Aboelela, O. A. El Rhman, and H. T. Dorrah, "Design of Missile Control System Using Model Predictive Control," *Technology*, no. 1, pp. 64–70.
[29] S. Boyd, "Model predictive control," vol. 2. editier, 2004.
[30] E. S. Meadows and J. B. Rawlings, "Chapter 5 Model Predictive Control."
[31] C. E. Garcia and M. Morari, "Internal Model Control . 1 . A Unifying Review and Some New Results," *Ind. Eng. Chem. Des. Dev.*, vol. 21, pp. 308–323, 1982.
[32] H. Bryson *et al.*, "Vertical Wind Tunnel for Prediction of Rocket Flight Dynamics," *Aerospace*, vol. 3, no. 2, p. 10, 2016.
[33] G. C. Engineering, "Lecture 15 - Model Predictive Control Part 2 : Industrial MPC," pp. 1–21, 2005.
[34] K. Ogata, "Controllability," in *Modern Control Engineering*, 2010, pp. 675–687.
[35] W. B. Dunbar, M. B. Milam, R. Franz, and R. M. Murray, "Model Predictive Control of a Thrust-vectored Flight Control Experiment," *IFAC Proc. Vol.*, vol. 35, no. 1, pp. 355–360, 2002.
[36] L. Blackmore, B. Acikmese, and D. P. Scharf, "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization," *J. Guid. Control. Dyn.*, vol. 33, no. 4, pp. 1161–1171, 2010.
[37] M. van Otterlo and M. Wiering, "Reinforcement Learning and Markov Decision Processes," Springer, Berlin, Heidelberg, 2012, pp. 3–42.
[38] R. S. Sutton and A. G. Barto, "The Markov Property," in *Reinforcement Learning: An Introduction*, 2013, pp. 61–65.
[39] R. E. Bellman, "The Theory of Dynamic Programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6. pp. 503–515, 1954.
[40] R. S. Sutton and A. G. Barto, "Temporal Differencing," in *Reinforcement Learning: An Introduction*, 2013, pp. 132–163.
[41] R. S. Sutton and A. G. Barto, "Q-Learning: Off-Policy TD Control," in *Reinforcement Learning: An Introduction*, 2013, pp. 148–151.
[42] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning : A Survey," pp. 237–285, 1996.
[43] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015.
[44] E. Catto, "Pybox2d 2.1.0 Manual," 2015. [Online]. Available:

https://github.com/pybox2d/pybox2d/wiki/manual.
[45] An. Packard, K. Poolla, and R. Horowitz, “Jacobian Linearizations, Equilibrium Points,” in *Dynamic Systems and Feedback*, 2002, pp. 169–200.
[46] J. W. Ponton, “Module 6: Multivariable Systems,” 2007. [Online]. Available: http://homepages.ed.ac.uk/jwp/control06/controlcourse/restricted/course/advanced/module6.html. [Accessed: 19-Jul-2017].
[47] M. C. E. Spring, “The Angle and Magnitude Criteria,” vol. 1, pp. 1–2, 2005.
[48] V. Mnih, D. Silver, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” pp. 1–9.
[49] R. S. Sutton and A. G. Barto, “Actor-Critic Methods,” in *Reinforcement Learning: An Introduction*, 2013, pp. 245–255.
[50] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic Policy Gradient Algorithms,” *Proc. 31st Int. Conf. Mach. Learn.*, pp. 387–395, 2014.
[51] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” 2015.
[52] K. Doya, “Reinforcement Learning in Continuous Time and Space,” *Neural Comput.*, vol. 12, no. 1, pp. 219–245, 2000.
[53] T. Salimans, J. Ho, X. Chen, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,” 2017.
[54] N. Hansen, “The CMA evolution strategy: A comparing review,” *Stud. Fuzziness Soft Comput.*, vol. 192, pp. 75–102, 2006.
[55] L. Dai, Y. Xia, M. Fu, and M. S., “Discrete-Time Model Predictive Control,” *Adv. Discret. Time Syst.*, 2012.