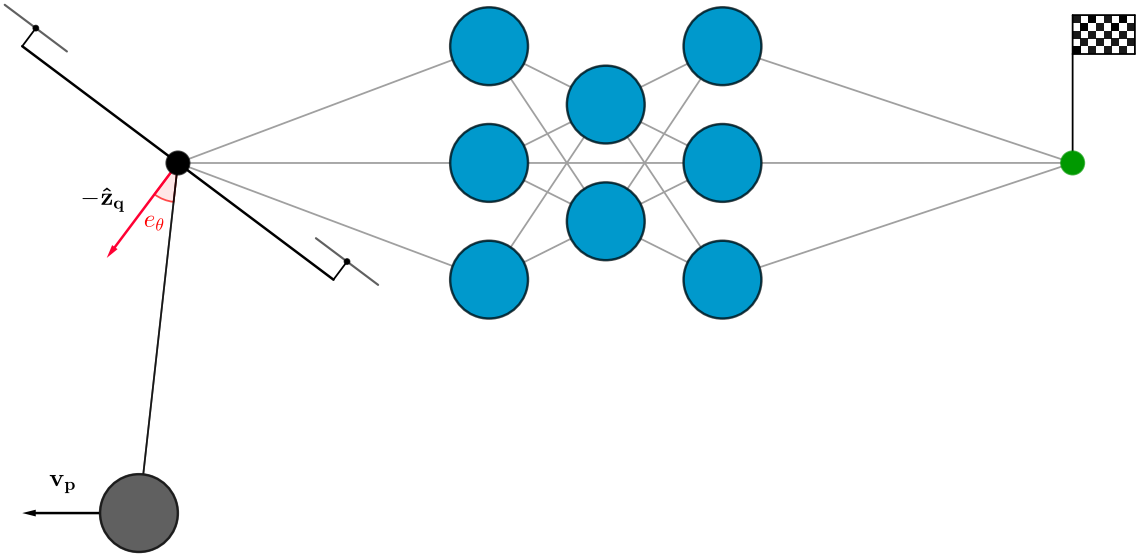
Quadrotor payload carrying using Model-Free Reinforcement Learning

by

David Felsager

AU-ID: au649158



Bachelor's Thesis

in

Electrical Engineering

Supervisor: Erdal Kayacan

Co-Supervisor: Halil Ibrahim Ugurlu

Aarhus University, Department of Electrical and Computer Engineering

15 June 2022

AARHUS UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Abstract

A recent developing approach in the domain of control is the use of Model-Free Reinforcement Learning (RL) when an accurate model of the system cannot be obtained. In this study, Model-Free RL is used to control a quadrotor carrying a slung payload. The viability and performance of this method is then compared to a Proportional Derivative (PD) controller. Training of the RL-model and evaluation of both controllers are carried out in the physics simulator Gazebo with a custom-built environment. The RL-model/controller is trained using the Model-Free RL algorithm "Soft Actor Critic", which tries to maximize reward while being as random as possible. The ambition is that Reinforcement Learning can provide a means of controlling the system in a way that reduces payload swing while keeping the agility of the quadrotor dynamics.

Contents

Acknowledgements

I want to thank Air Lab at Aarhus University and Erdal Kayacan for providing the facilities and opportunity to carry out this project. And a special thanks to Halil Ibrahim Ugurlu for supervising and supporting me throughout the whole project.

Chapter 1

Introduction

The quadrotor is an agile aerial vehicle used in a wide range of applications. One potential application is slung payload carrying, where the quadrotor is connected with a payload through a rope. In disaster-stricken areas, the ground can be hard to traverse, and quadrotors can be used to provide medical supplies in the immediate response. When a slung payload is introduced to the quadrotor's already highly non-linear dynamics, it results in a system with complex controller design. Possible ways of reducing this complexity must therefore be explored.

In this study, I assess the viability of Model-Free Reinforcement Learning (RL) to control a quadrotor carrying a slung payload. Model-Free RL is an approach chosen for control problems when an accurate model of the system is complex and therefore hard to design conventional controllers for. The RL algorithm "Soft Actor Critic" is chosen as the brain of the RL-controller because it suits the problem constraints. A custom environment is made to carry out the simulation and training of the RL-controller. After training the RL-controller, it is compared to a classical Proportional Derivative (PD) controller. Finally, the viability of the studied method is discussed, and its weak points covered.

First the problem formulation is laid out in Chapter 2, where the scope of the project, important definitions and assumptions are given. These definitions and assumptions are then later used in building the RL training environment. To build a RL model that can control a quadrotor, the basics of attitude/thrust quadrotor control are necessary to understand and are therefore given in Chapter 3. An intuitive explanation and a general overview of RL is covered in Chapter 4. I will then in Chapter 5 describe the methodology used in building and setting up the simulation model in Gazebo, setting up the RL model using Gym and Stable-Baselines3 and then the procedure for training it. The results from both the PD- and RL-controller are then shown and visualized using appropriate plots in Chapter 6. The viability of RL as a means to control a quadrotor carrying a slung payload are then discussed in Chapter 7. Lastly, concluding remarks about the use of RL as a means to control systems that are difficult to model are given in Chapter 8.

Chapter 2

Problem Formulation

2.1 Background

Slung payload carrying using an unmanned aerial vehicle (UAV) have applications in areas such as delivery of medical supplies in places with inadequate health infrastructure [1]. In such applications, the payload can be fragile and care must be taken when designing the controller, so large changes in the payloads relative position to the quadrotor are minimized.

The dynamics of systems such as a quadrotor carrying a slung payload can be difficult to model accurately. The quadrotor itself can be modelled as a rigid body, while the cable attaching it to the payload cannot because it is flexible. When the system model cannot be obtained, approximations of it can instead be used, as in [2], where the cable is modelled as a massless constant length rigid body that is attached to the drone's center of gravity with an ideal spherical joint.

Controlling a system which is hard or impossible to model accurately can be done in two different ways; using model-based control with an approximate model of the system, or using a model-free approach. The problem of finding good approximations of the system and controlling it has been studied using approaches such as modelling the cable as a spring-damper [3]. However, the use of model-free learning is most often done in simulation, where it is inevitable to compromise on the accuracy of the physics model. Therefore, effort is required to make the simulation model accurate enough that the controller could be transferred to an equivalent real-time system.

Making a simulation model which approximates the real-world equivalent can be easier than to obtain an accurate mathematical model. This is because the models in the simulation software, can be built in detail using joints and links, whereas it can be complicated to derive an accurate mathematical model for systems like the flexible rope that connects the quadrotor and payload.

Model-Free Reinforcement Learning (RL) for control is a recently developing approach compared to classical linear methods like Proportional, Integral, Derivative (PID) and Linear Quadratic Regulator (LQR) controllers. I have therefore chosen to study the viability and performance of Model-Free RL as a means to control the position of a quadrotor carrying a slung payload while also comparing it to a PD-controller.

2.2 Assumptions

The scope of this study is to use Model-Free RL to control a single quadrotor carrying a payload with known mass. Some simplifications and assumptions are necessary to achieve this, given the constraints of the study.

2.2.1 Simplifications

The dimensionality of the problem is reduced from 3-dimensional space to the 2-dimensional plane. 3D space consists of 3 orthogonal coordinate axes xyz. The xy axes make up the horizontal plane and z makes up the vertical axis. When simplifying to 2D, the 2. axis (y) is omitted, and thus, the system is only controlled in the xz-plane. This simplification can be justified by breaking down quadrotor attitude/thrust control to its simplest form; thrust is used to control the z-position, while pitch and roll angle respectively control the x- and y-position. Pitch and roll angle controllers are usually tuned to behave identically, such that the xy position control is independent of the quadrotors yaw angle.

In simplifying the problem to 2D, the thrust and pitch angle control will remain the same, and the pitch angle controller should be generalizable to a roll angle controller. An upside to this simplification is a decrease in dimensionality of the observation and action space, which most often result in an easier to train model.

A separate PD-controller controls the roll to stabilize the quadrotor in the y-axis.

2.2.2 Assumptions

- The payload mass m_p is known and has a constant value of:

$$m_p = 0.5kg \tag{2.1}$$

- Payload and quadrotor state can be obtained.
- Lower level control than the chosen attitude and thrust is provided by the Pixhawk 4 (PX4) Flight Controller Unit (FCU).
- The rope connecting quadrotor and payload can be modelled in simulation as many small links connected with revolute joints, with alternating axis of rotation.

2.3 Definitions

The following definitions are used when creating the classical PD- and the RL-controller.

2.3.1 Observation space

The observation space is the input to the model free learning and is information about the current environment state the agent is acting in. In the following, I list the different states used for defining the observation space, and visualize the payload swing angle in Fig. 2.1.:

- Simplified Quadrotor position:

$$\mathbf{p_q} = \begin{bmatrix} x_q \\ z_q \end{bmatrix} \tag{2.2}$$

- Simplified Quadrotor velocity:

$$\mathbf{v_q} = \begin{bmatrix} v_x \\ v_z \end{bmatrix} \tag{2.3}$$

- Quadrotor pitch angle (in world frame $\{\mathbf{w}\}$): ϕ
- Simplified Payload position:

$$\mathbf{p_p} = \begin{bmatrix} x_p \\ z_p \end{bmatrix} \tag{2.4}$$

- Payload swing angle:

$$\theta = -\operatorname{atan2}(z_q - z_p, x_q - x_p) + \frac{\pi}{2} \tag{2.5}$$

- Quadrotor position error in the x-axis:

$$e_x = x_{des} - x_q \tag{2.6}$$

- Quadrotor position error in the z-axis:

$$e_z = z_{des} - z_q \tag{2.7}$$

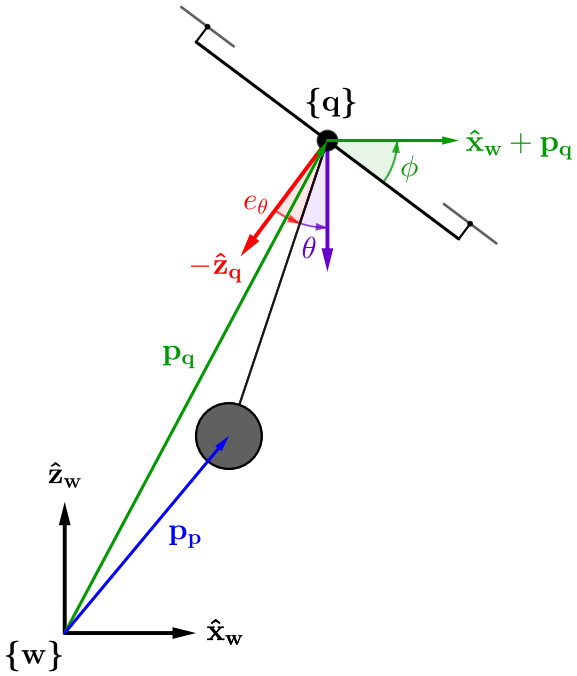


Figure 2.1: Visualization of the payload swing angle θ, quadrotor pitch angle ϕ and payload swing angle error e_θ.

- Payload swing angle error:

$$e_\theta = \phi - \theta \tag{2.8}$$

  - The payload swing angle error is the angle between the vector going from the quadrotor to the payload and the negative z unit vector $-\hat{\mathbf{z}}_\mathbf{q}$ in the quadrotor body frame $\{\mathbf{q}\}$. It is 0 when the vector going from the quadrotor to the payload is orthogonal with the x unit vector in the quadrotor body frame $\hat{\mathbf{x}}_\mathbf{q}$.

I define an observation in the environment $\mathbf{o}$ as:

$$\mathbf{o} = \begin{bmatrix} e_x & e_z & v_x & v_z & e_\theta & \dot{e}_\theta & \phi \end{bmatrix} \tag{2.9}$$

2.3.2 Action space

The action space is the output of the model free learning and is the action taken in the environment:

- Control input for PX4 thrust of the quadrotor control: u_T
- Control input for PX4 pitch control of the quadrotor: u_ϕ

I define an action taken by the agent as:

$$\mathbf{a} = \begin{bmatrix} u_T & u_\phi \end{bmatrix} \tag{2.10}$$

2.3.3 Performance and evaluation

I define the benchmark of good performance of the controller to be:

- Aggressive maneuvering
  - Reaching the desired position in the smallest amount of time.
- Minimum swing
  - The payload is directly under the quadrotor (in the quadrotor body frame) for the most amount of time, while payload movement is minimized.

These benchmarks for performance have been chosen for the following reasons: An important performance criteria for position control of quadrotor's is the time it takes to move between desired positions. In the earlier mentioned application, the need for medical supplies might be urgent, and therefore it is important that the drone can move aggressively through its environment. A medical supply payload may also be fragile, and the swing must therefore be minimized in order to avoid breaking the payloads contents.

As already hinted at, the controller will be trained and evaluated in a position control task. Evaluation is based on position errors e_x and e_z, payload swing angle error e_θ and the time taken t_f to reach the desired position.

Chapter 3

Quadrotor control

In the following, the simplified dynamics of a quadrotor, and a means to control it, is covered.

3.1 Dynamics

A quadrotor is an aerial vehicle consisting of a rigid cross frame with a motor and propeller attached to each of the four corners. Its pitch and roll angles are controlled by differentially adjusting the motor speeds, and its total thrust is given by the sum of the thrust from each motor. It is typically modelled as a rigid body in 3D space. That means it has 3 linear- and 3 angular degrees of freedom (DOF). Thus, the system has 6 DOF in total while having 4 actuators, making it underactuated. [4].

In this work, only 2 out of 3 dimensions are considered. The quadrotor can then be described with 2 linear and 1 angular DOF; the xz-position and the pitch angle.

3.2 Proportional derivative control

Considering the simplified system in 2D, position control can be accomplished by controlling the thrust and pitch angle of the quadrotor. The thrust is controlled about the equilibrium point where the quadrotor hovers i.e., the value of total thrust that directly opposes the gravitational force Mg, where M is the total mass of the system:

$$T^* = -Mg \tag{3.1}$$

A simple yet effective way to control a quadrotor is the classic Proportional Derivative (PD) controller. A PD positional controller can be designed to drive the system to a desired position using the positional errors and the derivative of the errors.

When the desired position is constant, the derivative of the position error is just the negative velocity. I derive it for $\dot{e}_x$:

$$\dot{e}_x = \frac{d(e_x)}{dt} = \frac{d(x_{des} - x_q)}{dt} = \frac{d(0 - x_q)}{dt} = -\frac{d(x_q)}{dt} = -v_x \tag{3.2}$$

It naturally follows that:

$$\dot{e}_z = -v_z \tag{3.3}$$

The pitch control input (3.5) is designed such that the quadrotor tilts and moves toward the desired x-position, while the thrust control input (3.6) is designed to drive the quadrotor to the desired z-position and hover at that position. When the quadrotor is tilted, the thrust of the quadrotor will have to be compensated because it won't be directly aligned with the world frame z-axis. It is done by projecting the thrust unto the world frame z-axis by multiplying with the factor (3.4).

$$n_T = \frac{1}{|\cos(\phi)|} \tag{3.4}$$

$$u_\phi = k_{px} e_x - k_{vx} v_x \tag{3.5}$$

$$u_T = n_T(T^* + k_{pz} e_z - k_{vz} v_z) \tag{3.6}$$

The coefficients $\mathbf{k}$ are the tunable parameters of the controllers:

$$\mathbf{k} = \begin{bmatrix} k_{px} \\ k_{pz} \\ k_{vx} \\ k_{vz} \end{bmatrix} \tag{3.7}$$

Intuitively, the proportional part of the controllers drives the positions toward the desired position, while the velocity decreases overshoot when the quadrotor gets close to the desired position. This can be thought of as the velocity overtaking the position error.

It can be necessary to saturate the pitch angle of the quadrotor when using a linear controller like PD. Otherwise, the system can move into highly non-linear regions of its state-space where the linear controller most likely will not work properly. Intuitively, a linear controller increases the control input linear with the input error. So the linear controller "thinks" that it can just keep increasing the pitch to move faster to the desired x-position, when in reality is will tip over at some larger pitch angle and crash.

Chapter 4

Model-Free Reinforcement Learning

When a child is learning to walk, it knows nothing about how to at the start. It learns by trying different actions. At first, it begins to crawl, then slowly it progresses by standing up and falling many times. The child feels pain when it falls, and begins to associate a negative feeling with actions that leads to this outcome. When it tries to walk and gets closer to succeeding, its parents will typically cheer it on, and the child gets a positive association with that action. In time, the child learns how to walk by repeating the process many times.

Reinforcement Learning (RL) uses these general principles to accomplish many incredible tasks, such as controlling robots both in simulation and in the real world. A block diagram of the general RL scheme is given in Fig. 4.1 as an overview of what will be covered in the following. The two major elements of RL are the agent and the environment. In the environment, the agent acts and thereby changes the environment. Learning a desired behavior is mapping the states (or partial states) of the environment to the actions that lead to the desired change in the environment. This mapping is called the agent's policy and can, for example, be a deep neural network. The agent learns by a measure called the reward signal. A reward signal needs to be shaped in a manner such that it reinforces desired actions and punishes undesired actions. Then the goal of the agent is to act in the environment such that it maximizes accumulated reward, also called return [5].

To summarize, an agent acts according to the policy, which will change the state of the environment. The action is then fed back into the RL algorithm. The new state of the environment is fed back into the policy and the algorithm, and also signals a reward back to the algorithm. The algorithm then updates the policy according to the action taken by the agent, the new state of the environment and the resulting reward.

Returning to the child learning how to walk analogy, the parallels to RL is as follows:

- The child's brain is the agent
- The neural pathways that are reinforced is its policy
- The sensation of pain or cheering on is the reward signal

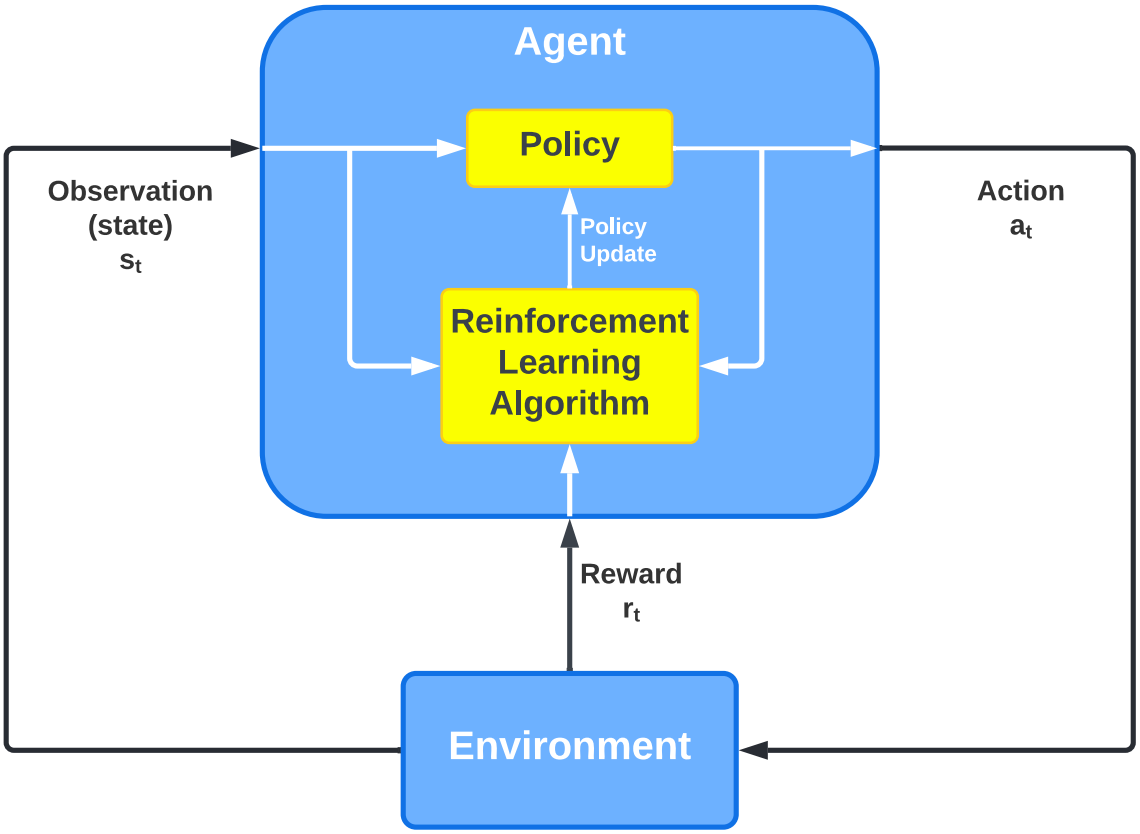


Figure 4.1: Block diagram of the general Reinforcement Learning structure. Inspired by diagram in [7].

- The child itself and the place it is moving in is the environment
- Sensation of touch and sight is the observation of the environment (partial state)
- Movement by the child is the action.

There are two main approaches when creating a RL algorithm; model-based and model-free. In model-based RL, the agent has access to or learns a model of the environment, while in model-free learning a model of the environment is not considered. The advantage of using model-based RL is that it can make predictions of state and reward, while the downside is that obtaining an accurate model can be difficult [6]. And even if an accurate model exists, the complexity of some systems makes the controller design costly. For this reason, only model-free RL is considered in this study because it has been shown to generalize to a lot of different use cases, ranging from advanced control tasks to playing strategy games like Go [5].

4.1 Agent and environment

In RL, the two major parts are agent and environment. The agent is trained in the environment using an RL algorithm to make it act in a desired way. When the agent acts in a desired way, it is rewarded, and oppositely it is punished when it

acts in an undesired way. In Model-Free RL, no prior information is known about the environment. This is favorable in scenarios where a model of the environment is either very difficult or impossible to obtain. The goal of the agent is to act in a way that maximizes the accumulated reward, called the return R [8].

4.2 Observation- and Action space

All information needed to fully describe the environment is called its state $\mathbf{s}$. If state transitions of the system only depends on the most recent state and action, the system is said to obey the Markov property [5]. Often only partial knowledge about the state is available to the agent. An observation $\mathbf{o}$ is instead used interchangeable to describe both the full or partial state. Observations are used as the input to an RL algorithm and is the data it uses to determine an action.

The observation space is all possible observations that can occur and is usually normalized and clipped to $[-1, 1]$. The output of an RL algorithm is the action $\mathbf{a}$ that the agent should take given an observation $\mathbf{o}$. All possible control inputs of the system is called the action space. The action space is also usually normalized and clipped.

4.3 Measuring Performance

4.3.1 Reward signal

The reward signal is a scalar r that is used as a measure of how desired a current state of the environment is. It should be positive when the agent is acting in a desired way and negative when it is not [5].

4.3.2 Reward Shaping

The reward can be shaped using some of the same state variables that are measured in an observation. Typically, it is normalized to be in the interval $[-1, 1]$.

One way of doing this is shaping the reward in the following way, where it is assumed that the entire observation space is used to shape the reward:

$$r = 1 - \mathbf{k} \cdot |\mathbf{o}| \tag{4.1}$$

The coefficient vector $\mathbf{k}$ are designed to exactly add up to 2 and the observations are normalized and then taken the absolute value of to keep them in the interval $[0, 1]$. Then the minimum value of the reward will occur if $\mathbf{o} = \mathbf{1}$:

$$r_{min} = 1 - \mathbf{k} \cdot \mathbf{1} = 1 - 2 = -1 \tag{4.2}$$

The maximum value will occur if $\mathbf{o} = \mathbf{0}$:

$$r_{max} = 1 - \mathbf{k} \cdot \mathbf{0} = 1 - 0 = 1 \tag{4.3}$$

For the assumption of being able to use the absolute observations to shape the reward, it is necessary that the observations are measures of something undesired, such as errors.

4.3.3 Trajectories

The subscript t of an action a_t or state s_t is the time step at which the action is taken, and the state is measured. A time step in the environment is described as follows:

$$\mathbf{s_t} \rightarrow \mathbf{s_{t+1}} \tag{4.4}$$

$$\mathbf{a_t} \rightarrow \mathbf{a_{t+1}} \tag{4.5}$$

A trajectory is defined as a sequence of states and actions in the environment:

$$\tau = \{\mathbf{s_0}, \mathbf{a_0}, \mathbf{s_1}, \mathbf{a_1}, \ldots\} \tag{4.6}$$

4.3.4 Return

The return $R(\tau)$ is a measure of cumulative reward over a trajectory τ [5]. There are two different approaches to return. One is finite-horizon undiscounted return where a recent time window of the reward history is used:

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{4.7}$$

The other is infinite-horizon discounted return where the entire reward history is used, with most recent in time weighted the heaviest and the least recent the lightest:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{4.8}$$

4.4 Policy and learning

4.4.1 Policy types

Mapping observations to states are done according to a policy. In principle, the mapping can be done in any sort of way where the policy consist of a rule set.

In deep RL, a parametrized policy is used by means of a Multi Layer Perceptron (MLP) as the policy of the agent [5]. The input of the MLP is an observation from the environment, and the output is an action for the agent to take. The model parameters in an MLP are its weights and biases.

A policy can be either deterministic or stochastic. In a deterministic policy μ, the output action is a function of an observation:

$$\mathbf{a} = \mu(\mathbf{o}) \tag{4.9}$$

A stochastic policy π is instead a distribution from which actions are sampled given an observation:

$$\mathbf{a} \sim \pi(\cdot|\mathbf{o}) \tag{4.10}$$

4.4.2 Value functions

A value function models the expected return given a specific trajectory starting in $\mathbf{s_0} = \mathbf{s}$. The on-policy value function the agent acts according to its policy π:

$$V^{\pi}(\mathbf{s}) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau) | \mathbf{s_0} = \mathbf{s}] \tag{4.11}$$

The on policy action-value function also starts by taking an arbitrary action $\mathbf{a_0} = \mathbf{a}$ and then acts according to policy:

$$Q^{\pi}(\mathbf{s}) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau) | \mathbf{s_0} = \mathbf{s}, \mathbf{a_0} = \mathbf{a}] \tag{4.12}$$

The value and action-value functions for a RL model is usually not known and are a big part of the learning.

4.4.3 Updating a policy

In Model-Free RL there exists two different approaches to updating a parametrized policy. The first one is policy optimization, where the policy parameters are updated by maximizing an objective function J. Policy optimization is typically done on-policy, which means that it only uses the data collected while on the current policy parameters. The other approach is Q-learning, where the optimal action-value function is approximated. Q-learning is typically done off-policy, which means that all data collected during training regardless of changes in policy is used. The two approaches are not irreconcilable and can be used together, to exploit the strengths of both [6]. One example of such an algorithm is "Soft Actor Critic" (SAC) which will be used in this study. SAC tries to maximize reward while also being as random as possible. It is off-policy and thus sample-efficient, while also remaining very consistent [9]. The Stable-Baselines3 implementation of SAC, which is chosen for this study, uses continuous action and observation spaces, which is well suited for continuous control.

4.5 Exploration vs Exploitation

An agent can choose to either use what it already knows about how its actions affect the environment to maximize return according to that policy. Or, it can try new actions to gain information about potential new ways of acting that are more rewarding. The tradeoff between the two is called exploration versus exploitation. When the agent explores, it tries new actions to find new possible ways of gaining reward, while when it exploits, it uses the knowledge it already has to maximize reward according to its policy [5].

Chapter 5

Methodology

In the following, an overview and a description of all the individual parts that make up the simulation environment is given. Necessary properties of the simulated flight controller are given, and the models used in the simulation environment are described. Design and tuning of a PD-controller for the quadrotor carrying a slung payload is then given, with relevant plots. Lastly, a complete overview of the RL environment creation and the considerations that went in to making it is given, and the procedure of training the agent is described.

5.1 Simulation environment

5.1.1 Overview

The Gazebo physics simulator was chosen for this study because it interfaces with the Robot Operating System (ROS), it is supported by Pixhawk 4 (PX4) Software In the Loop (SITL) simulation of a Flight Controller Unit (FCU) and it supports modelling with the Simulation Description Format (SDF). ROS is primarily used for communication between the simulator, the SITL FCU and the active controller. Furthermore, it is used to manage the software package that contains the controllers, RL environment and utilities. PX4 is used for its simulated FCU, which provides high level control of the quadrotor's attitude and thrust. It should be noted that even though the problem is simplified to 2D, the simulation is still carried out in 3D. The y-axis control is excluded from the RL controller.

5.1.2 Using the Pixhawk 4 flight controller

The hover equilibrium of the PX4 FCU is found to have the following value:

$$T^*_{FCU} = 0.83 \tag{5.1}$$

Also the range of thrust values for the FCU is the interval (5.2), where 0 is no thrust and 1 is max thrust.

$$T_{FCU} \in [0, 1] \tag{5.2}$$

When the pitch angle is input to the PX4 FCU it has to be converted to a quaternion. This is done using a built-in ROS function, where roll and yaw are given together with pitch. The yaw angle will be commanded to 0, while roll is regulated using the same PD-controller as in the pitch for the classical approach.

5.1.3 Quadrotor model

The 3DR Iris quadrotor model [10] is used because it is designed to be used with the PX4 Gazebo SITL simulation.

5.1.4 Rope model

As mentioned in Chapter 2 a crude approach to modelling the rope is a single rigid mass-less link with one end fixed to the payload and the other connected to the quadrotor with a spherical joint. A natural extension of this model is to increase the number of links and connect them with spherical joints and attributing the links mass. Theoretically, the number of links (and thereby joints) can be increased to infinity to reach a very accurate rope model. But in simulation, the practicality of the model must also be considered. Using too many links will result in a very slow simulation, which is highly problematic in RL where simulation speed is important to train the model. Therefore, I created a rope model consisting of 20 very small links connected with revolute joints having alternating axes of rotation. One end of the rope is then connected to the quadrotor model and the other the payload also using revolute joints. The joints are damped to mimic a physical rope. In Fig. 5.1 the rope model can be seen with every joint visualized by a coordinate frame with the axis of rotation circled. Each of the links is given a mass of $0.0005kg$ which in

Figure 5.1: Full simulation model with joints visualized by circles around axes of rotation.

total amount to the cable having a mass of:

$$m_c = 0.0005 \times 20 = 0.01kg \tag{5.3}$$

The length of each individual length is $0.025m$, so the total length of the rope measures up to:

$$l_{c,max} = 20 \times 0.025 = 0.5m \tag{5.4}$$

5.1.5 Payload model

The payload is modelled as a sphere with mass m_p and radius r_p:

$$m_p = 0.5kg \tag{5.5}$$

$$r_p = 0.05m \tag{5.6}$$

5.1.6 Visualization of desired position

The desired position of the quadrotor is visualized in the simulation environment by a transparent green sphere as shown in Fig. 5.2.

5.2 Proportional Derivative controller

5.2.1 Design

A PD-controller is designed to control the thrust and pitch of the quadrotor. The control inputs are designed as described in (3.6) and (3.5).

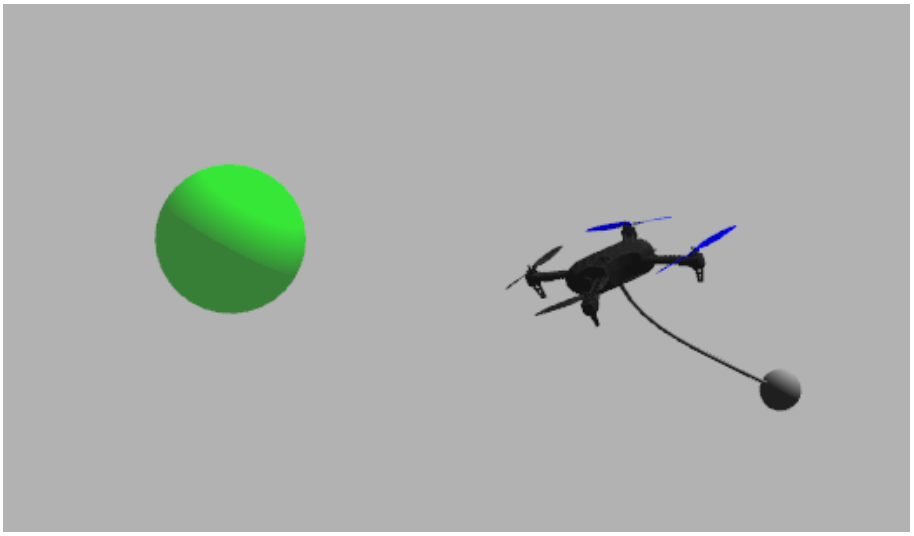


Figure 5.2: Quadrotor, rope, payload and desired position in simulation.

5.2.2 Tuning

The PD-controller tuning is carried out in two scenarios: thrust is tuned using a unit-step input, where it is made to stabilize. Then the pitch angle is tuned by making the quadrotor fly to $+10$ in the x-axis to simulate the scenario wherein it will be compared to the RL-controller. Both the thrust and pitch controller are first tuned to have a fast rise time without steady-state oscillations using the proportional error, and then the velocity is tuned to dampen overshoot. The thrust controller that controls z-movement ends up being critically damped as seen in Fig. 5.3, which means that it has a fast rise time but no overshoot. The pitch controller that controls x-movement ends up having a small overshoot as seen in Fig. 5.4, which is found acceptable as a tradeoff to being aggressive.

Tuning the controllers too aggressively means the quadrotor will accelerate too fast and the payload swing angle error will become too great and the quadrotor drops to the ground. The pitch angle is saturated to the range $[-0.31, 0.31]$ to avoid the quadrotor from tipping over with the payload.

5.3 Reinforcement Learning Controller

To train a RL model using the "Soft Actor Critic" algorithm, an interface between the physics simulator and the algorithm has to be set up. Then using this interface called a "custom Gym environment" the model training scenario is created.

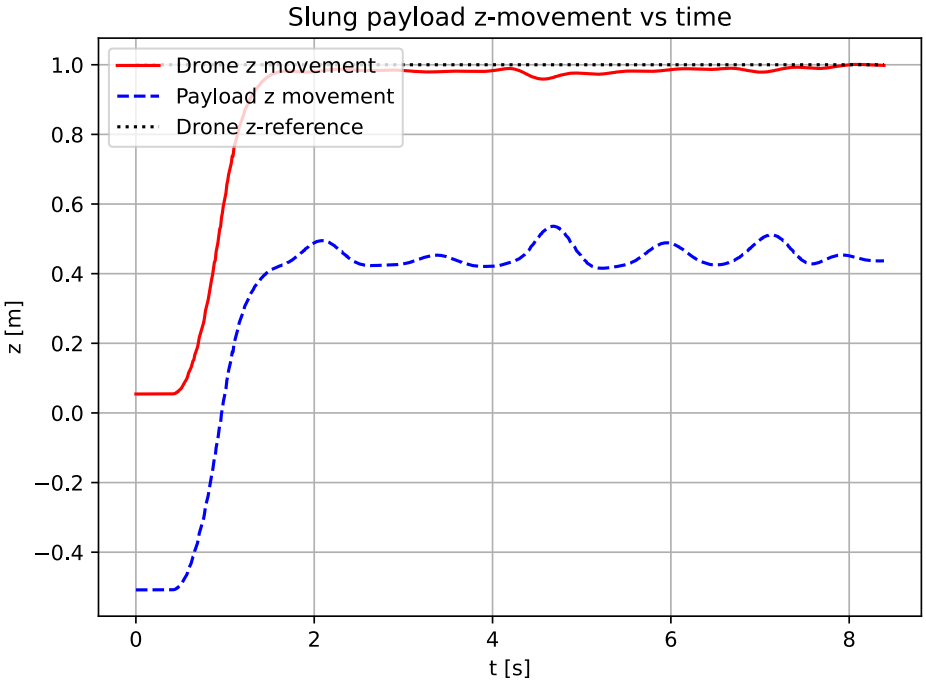


Figure 5.3: Plot of z-movement as a function of time.

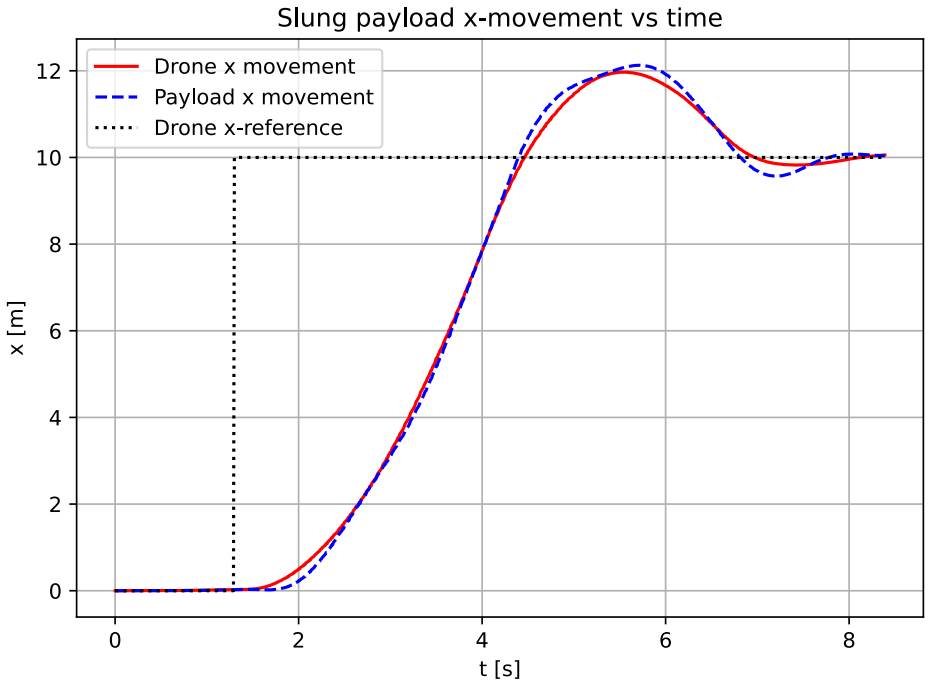


Figure 5.4: Plot of x-movement as a function of time.

5.3.1 Gym and Stable-Baselines3

For building and training the Reinforcement Learning controller, the two Open Source Python packages "Gym" [11] and "Stable-Baselines3" (SB3) [12] are used. Gym is used to build the RL environment such that the physics simulator Gazebo can be used to train the RL agent. SB3 provides the algorithm "Soft Actor Critic" [9] for the RL agent's "brain".

5.3.2 Custom Gym environment

To use Gym and SB3 with the Gazebo simulation environment, it is necessary to create an interface between the two. This is done using the template environment class from Gym. This class needs the two following functions: step and reset.

Step

The "step" function is called to take a time step in the environment. The input to a time step is an action to be taken in the environment, and it returns a new observation, a reward and a done variable that tells the RL environment if it needs to reset.

Reset

The reset function resets the state of the environment to an initial state, so the agent can start anew, and generates a new desired position. Resetting the simulation environment consists of making ROS service calls to different Gazebo services that reset the position and joint angles of the quadrotor, rope and payload.

Taking an action

The raw action $\mathbf{a}$ is a 2D vector with values in the interval $[-1, 1]$. The first entry in the vector a_1 is used as the thrust input command to the quadrotor and is scaled and offset around the hover equilibrium:

$$u_T = T^*_{FCU} + 0.17a_1 = 0.83 + 0.17a_1 \tag{5.7}$$

This is done so thrust actions are taken symmetrically around the hover equilibrium, such that $u_T \in [0.66, 1]$. The second entry a_2 is used as the pitch angle input command by scaling it to the interval $[-0.31, 0.31]$. It is scaled to the same range as the PD-controller is saturated such that they are equally constrained:

$$u_\phi = 0.31a_2 \tag{5.8}$$

The roll angle is controlled using the PD-controller, to stabilize the quadrotor in the y-axis.

Reward function

The reward function is calculated according to (4.1). The position errors are weighted the highest, with the swing angle error weighted the second highest. This is because they always are indicators of an undesired state. The velocity and pitch angle of the quadrotor, together with the derivative of the swing angle error, is weighted about the same at very low values. Velocity is desired when the system is moving towards the desired position, and it should only have any significance when stabilizing at the desired position. The same goes for the pitch angle, and the swing angle error should also not be punished too hard, because it is sometimes desired to have the payload moving to the 0 swing angle error position. The final reward coefficient vector used in training is as follows (subscripts are according to the definitions given in 2.3.1):

$$\mathbf{k_f} = \begin{bmatrix} k_{e_x} \\ k_{e_z} \\ k_{v_x} \\ k_{v_z} \\ k_{e_\theta} \\ k_{\dot{e}_\theta} \\ k_\phi \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.7 \\ 0.05 \\ 0.05 \\ 0.2 \\ 0.05 \\ 0.05 \end{bmatrix} \tag{5.9}$$

Calculating an observation

The ground truth of the system is obtained using the Gazebo ROS topic "link_states". Some of the variables used for an observation (2.9) are errors that must be calculated. They are calculated according to the equations covered in subsection 2.3.1.

Termination condition

A set of criteria for when the agent should terminate and reset is chosen based on practical considerations. Firstly, the velocity of the quadrotor in the negative

z-axis is used as a parameter to reset the training environment. If the negative velocity is great enough, it means the quadrotor is just dropping to the ground. But if the condition is tuned with a threshold that is too low, the agent won't have time to learn and will just reset before doing anything. This condition is also in place because of constraints coming from the PX4 SITL. If the velocity increases too fast, it messes with its simulated extended Kalman filter (EKF2) and a longer reset would be needed.

It is also reset if the quadrotor falls below a $z-$-threshold, so it doesn't hit the ground. The agent also resets after a certain time limit, so the quadrotor doesn't end up in situations where it keeps going while accumulating a large penalty. These termination conditions are designed to incur a large reward penalty.

The environment also needs to terminate when the quadrotor is very close to the desired position while maintaining a low velocity. When reaching the desired position, a reset should incur a large reward. Though, in order to favor reaching the desired position quicker, a penalty is subtracted from the otherwise large reward. I define this as a coefficient times the time taken to reach the desired position divided by the distance from the starting position to the desired position:

$$r_{pen} = k_{pen} \frac{t_f}{||\mathbf{p_{des}} - \mathbf{p_{start}}||} \tag{5.10}$$

The termination rewards and penalties are also parameters to be tuned when training the model.

5.3.3 Training

The training scenario for the agent is set up such that every time the simulation environment is reset, the position of the quadrotor is set to:

$$\mathbf{p_{start}} = \begin{bmatrix} 0 \\ 20 \end{bmatrix} \tag{5.11}$$

Then a desired x-position is sampled from the uniform distribution:

$$X \sim U(-10, 10) \tag{5.12}$$

The desired z-position is chosen as the starting position plus a the uniformed distributed variable Z:

$$Z \sim U(-1, 1) \tag{5.13}$$

It results in the desired position taking the form:

$$\mathbf{p_{des}} = \begin{bmatrix} X \\ 20 + Z \end{bmatrix} \tag{5.14}$$

Training of the RL model is carried out in a separate program where the custom environment is imported and SB3 is used to create the RL-model. The training is monitored by saving the return after every episode. An episode starts after a reset of the Gym environment and ends at its termination. The reward history is then smoothed using a moving average filter to filter out high frequency components, such that the general tendency of the training can be seen. Over time, return is expected to increase. If it doesn't the reward function or other parameters in the custom Gym environment needs to be tuned.

Chapter 6

Results

Both controllers are tested on their ability to minimize swing angle error while reaching the desired position in the shortest amount of time. The accumulated reward (return) gathered with the reward function used to train the final RL controller is used as the first benchmark. As the second benchmark, the absolute average swing angle error and time taken to reach the desired position is used. Using the reward function to measure performance is expected to favor the RL controller, since it is literally built for it. Therefore, it is mostly used to support the avg. swing angle error and time taken.

The test scenario consists of the quadrotor carrying the payload, staying at the same height while moving $+10m$ in x. It starts in the position:

$$\mathbf{p_{start}} = \begin{bmatrix} 0 \\ 20 \end{bmatrix} \tag{6.1}$$

And flies to the desired position:

$$\mathbf{p_{des}} = \begin{bmatrix} 10 \\ 20 \end{bmatrix} \tag{6.2}$$

6.1 Training the model

In the following, I will cover selected training's of the RL-model that ends up being the final RL-controller. This is also the RL-controller that will be compared with the PD-controller. The shown learning curves are smoothed with a moving average filter with appropriate filter sizes. The filter sizes are chosen to make the learning curve plots show the tendency of return over time.

In total, the model is trained over 10 different training scenarios where reward, termination conditions, position variance and normalization factors are adjusted. The RL-model is trained on the scenario described in subsection 5.3.3 and variations of it. In the first trainings, variance in z is set to zero such that $Z = 0$. The reward coefficients for position errors start out being weighted the same. Termination reward and penalty are tuned to be lower such that they don't play too big a role in the beginning of the trainings, because the agent won't be able to reach the desired position consistently and will fail a lot.

After the first training, the model is tested in the deterministic mode, such that the most probable action for a given observation is taken by the agent. The many first episodes of the training ends up with a very low return until the agent learns to get closer to the desired position. It is then found that the quadrotor stops short of the desired position to stay there and accumulate reward until time runs out. This tactic of gaining reward makes the learning plateau, as can be seen in the end of the graph of Fig. 6.1, so the training is stopped.

Before the next training, the reward coefficients are adjusted to weight position error higher and swing angle error less. The position threshold for reaching the desired position was also increased to allow the agent to have more positive terminations and thereby learning that it can gain a larger return by doing this.

This did not achieve the desired effect, and the quadrotor kept stopping short to stay and accumulate reward. So, over the next training's the inverse velocity penalty (5.10) is implemented to incentivize reaching the desired position faster. The normalization factor n_x of observations e_x are changed from being dependent on the current value of the desired x-position to always being the maximum absolute value of the desired x-position:

$$n_x = |x_{des,max}| = 10 \tag{6.3}$$

In the later trainings Z variance are added. Through all the training, the normalization factor for e_z is set to a constant value of $n_z = 1$.

These changes improve the RL-controller somewhat, and it got better at reaching the desired position, though still "preferring" accumulating reward close to the desired position. More drastic measures are therefore needed to make reaching the desired position the most rewarding option for the agent. The time before terminating is decreased from $40s$ to $25s$. I assume that this will make the time limit termination occur more frequent. The penalty for this termination condition is therefore decreased. Reaching the desired position is rewarded more, while the inverse velocity penalty is increased to (6.4) (the subscript f is used to signify the final value used).

$$r_{pen,f} = k_{pen,f} \frac{t_f}{||\mathbf{p_{des}} - \mathbf{p_{start}}||} = 400 \times \frac{t_f}{||\mathbf{p_{des}}||} \tag{6.4}$$

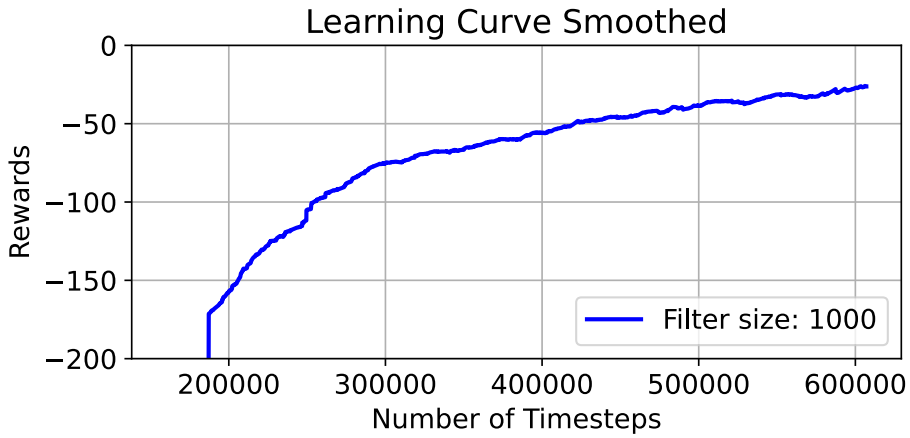


Figure 6.1: Learning curve of the first training.

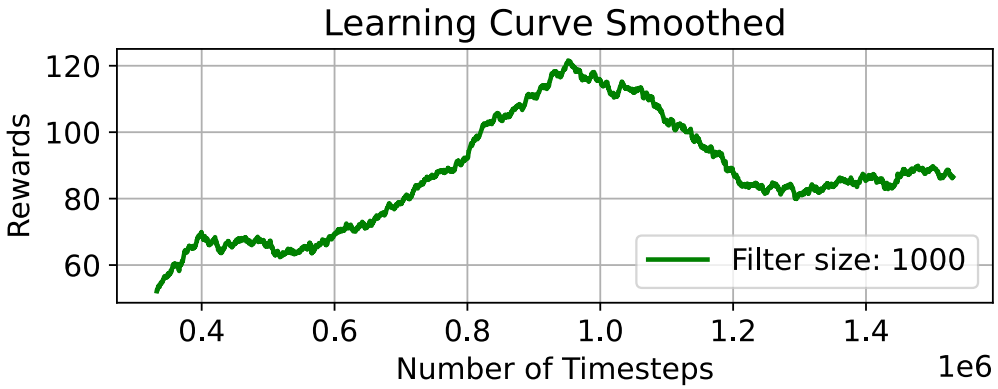


Figure 6.2: Learning curve of the third last training.

The result of what ended up being the third last training was very promising. The reward increased steeply to then reach a peak, fall a significant amount to then plateau as seen in Fig. 6.2. Early stopping is utilized, and training is continued on the model saved around the large peak. At this point, the model has learned to go all the way to the desired position, so stabilizing the quadrotor at the desired position became the main concern.

In the last trainings, both the distance and velocity thresholds are lowered to a value of 0.05. A threshold of the quadrotor's pitch angle ϕ is also added, such that the quadrotor has to be very still for termination to occur. It also has a value of 0.05. With these changes in place, the second last training is carried out. It is stopped when the learning curve plateaus.

To even the odds with the PD-controller which was tuned on the evaluation task, the desired x-position in the last training is changed to be equiprobable ± 10. The last training is done over two sessions, where nothing is changed between the two. They are split to test the deterministic mode performance of the model. The last of the two trainings are stopped when the learning curve plateaus and is shown in Fig. 6.3. The final model that is chosen to be compared with the PD-controller is the one saved at time step 1e6. The early stop is chosen at this time step because of the "flat" peak that occurs in the learning curve at that time.

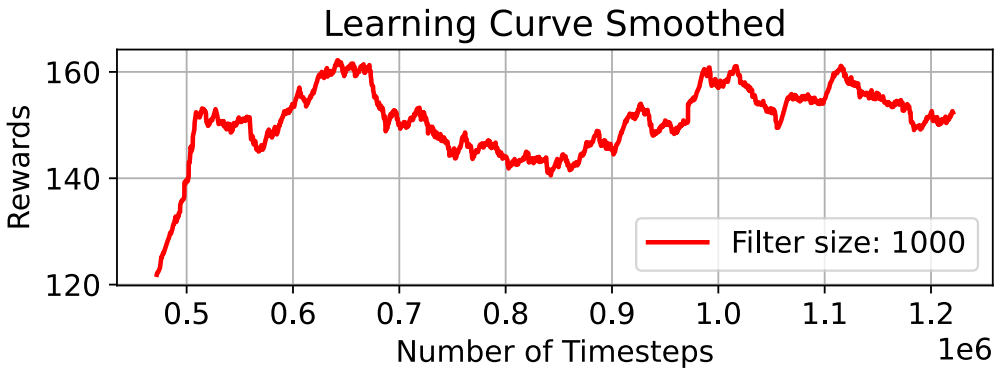


Figure 6.3: Learning curve of the last training.

6.2 Evaluation

To evaluate the performance of the PD- and RL controller, 10 trials are carried out with each. Simulation speed is reduced to 0.2x because it increases the visual quality of the plots, and should have no effect on performance, when using simulation time with ROS. The average reward accumulated over these trials are used as the first performance benchmark. While carrying out the 10 trials, the absolute average swing angle error and the time taken to reach the desired position is tabulated. A sample plot of the errors for each controller is also shown for a qualitative comparison.

The evaluation task is first run 10 times with the PD-controller. The chosen measures of performance are tabulated in Table 6.1 and the errors of a sample trial are plotted in Fig. 6.4. The 10 trials are then averaged in the bottom row of the table. The same procedure is repeated with the RL-controller. The corresponding data is shown in Table 6.2 and the errors of a sample trial are plotted in Fig. 6.5.

Trial no:	Accumulated reward:	Time taken [s]:	Avg. e_θ [rad]:
1	474.07	6.84	1.78
2	532.2 0	7.82	1.77
3	489.85	6.82	1.77
4	496.35	7.18	1.78
5	493.55	7.18	1.78
6	492.51	7.08	1.79
7	493.61	7.02	1.79
8	492.93	7.12	1.79
9	487.42	6.90	1.79
10	491.50	6.92	1.79
Avg.	494.40	7.09	1.78

Table 6.1: Benchmark of PD controller.

Trial no:	Accumulated reward:	Time taken [s]:	Avg. e_θ [rad]:
1	800.17	12.46	0.043
2	795.24	12.66	0.047
3	774.17	12.02	0.049
4	808.26	12.84	0.045
5	782.54	11.88	0.051
6	808.68	13.08	0.047
7	795.74	12.60	0.046
8	787.35	12.06	0.050
9	799.31	12.20	0.045
10	778.13	11.88	0.051
Avg.	792.96	12.37	0.047

Table 6.2: Benchmark of Model-Free RL controller.

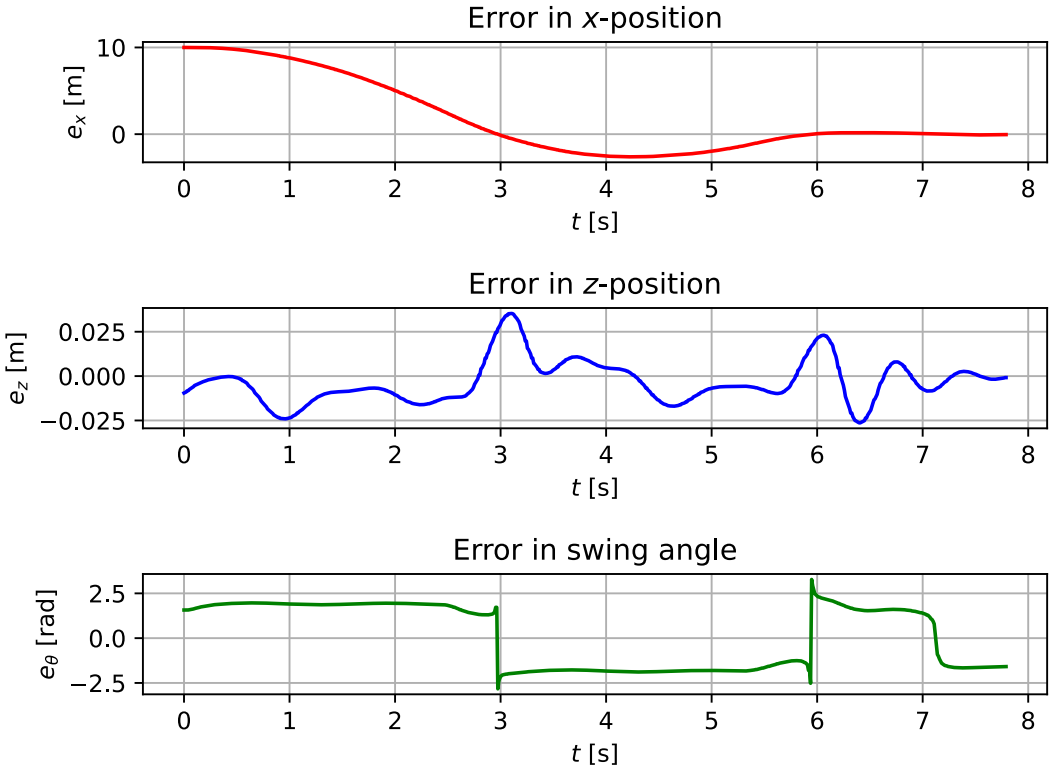


Figure 6.4: Errors of PD controller performing evaluation task.

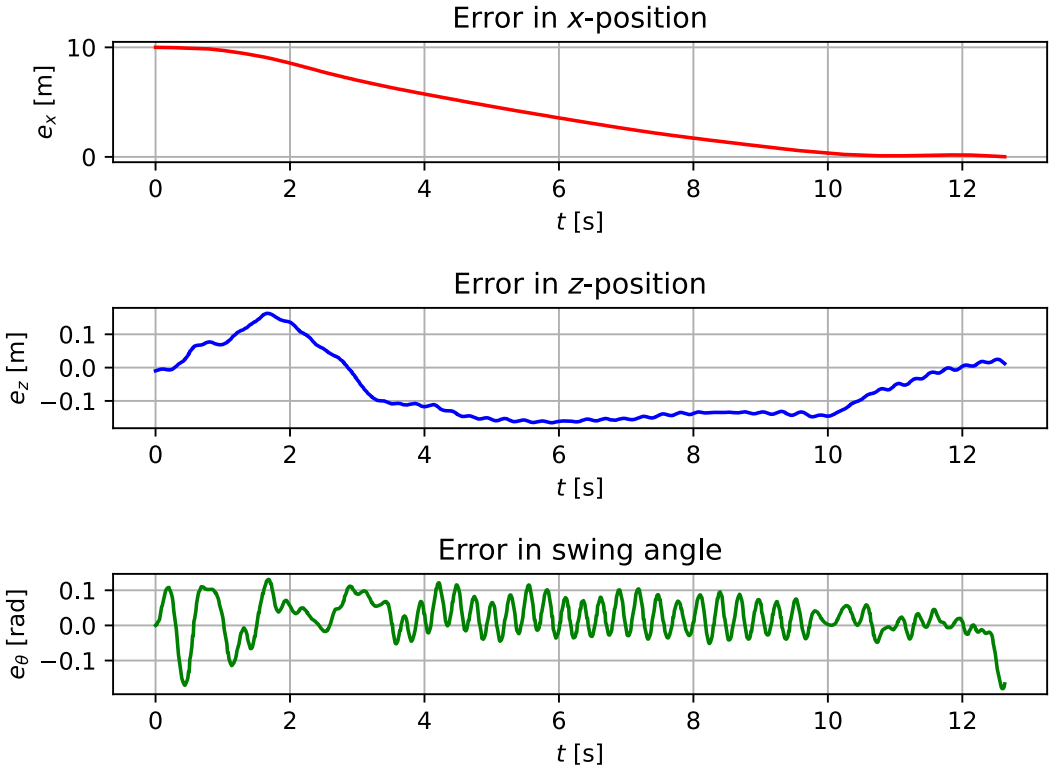


Figure 6.5: Errors of RL controller performing evaluation task.

Chapter 7

Discussion

In the following, I will first discuss the performance of the PD- and RL-controller seen in the results, and highlight major differences. Then the problem of model explainability when using Model-Free RL is discussed. The viability of Model-Free RL as a means to control a quadrotor carrying a slung payload is then assessed. Lastly, potential extensions and changes to this study are covered.

7.1 Controller Performance

The controller performance from the trial results covered in section 6.2 are discussed in the following sections. Emphasis is put on the performance criteria listed in subsection 2.3.3.

7.1.1 Proportional Derivative Controller

In Fig. 6.4 a slightly underdamped response is observed in the x-position error e_x, while very small changes in the z-position error e_z are seen. A large constant error of about ± 1.8 is seen in the payload swing angle error e_θ, with huge changes in the error around the desired position. The result is in an aggressive response with a large payload swing angle error, which is to be expected when only the position errors and velocity of the quadrotor are input to the controller. Though it should be noted that the pitch angle ϕ is indirectly input when compensating the thrust according to (3.4).

From table 6.1 very consistent values of the performance criteria are observed. The tabulated data supports what is observed in the plot, showing an average completion time of 7.09s. Assuming direct flight from start to desired position, the resulting average speed is:

$$\bar{v}_{pd} = \frac{10}{7.09} = 1.41 \frac{m}{s} \tag{7.1}$$

7.1.2 Model-Free Reinforcement Learning Controller

The behavior of the RL-controller after its final training is best described as slow and steady. It moves with about constant velocity most of the way in x as seen in Fig. 6.5, while it dips in z (corresponds to positive e_z) to then rise above the desired

z-position, to lastly converge to the desired z-position. Again assuming direct flight from start to desired position, the resulting average speed is:

$$\bar{v}_{rl} = \frac{10}{12.37} = 0.81\frac{m}{s} \tag{7.2}$$

The payload swing angle error is kept below 0.1 most of the time, while exhibiting small oscillations. In the simulation, it is observed that the quadrotor pitch angle changes back and forth the closer it gets to the desired position. This behavior looks to stabilize the payload around $e_\theta = 0$, but the changing thrust dynamics creates oscillations in z, which results in the small oscillations of the payload swing angle error. The average payload swing angle error end up being very low at a value of:

$$\bar{e}_{\theta,rl} = 0.047 \tag{7.3}$$

The RL-controller manages to reach the desired position while keeping the payload swing angle error low, and thereby succeeds in the second evaluation criteria; minimum swing. But the low speed means that it fails at the first criteria; aggressive maneuvering.

Therefore, when shaping the reward and choosing the observation space of an RL-model, great care must be taken to make it act in a desired way. Often there are ways the agent can exploit the reward function and/or environment which has not been considered. This was the case in the training of the RL-controller, where there was not enough incentive to reach the desired position fast.

7.1.3 Differences and tendencies

There are two obvious differences between the controllers; the PD is a lot faster than the RL, but with some overshoot, and while the RL keeps the payload swing angle error low with oscillations, the PD has a large constant absolute error.

The reward accumulated by the RL-controller is significant higher than the PD-controller, as seen in the tables. But there seems to be a positive correlation between reward accumulated and time taken. This highlights a problem with the reward penalty for going slow. It was intended that the reward should be higher for faster completion to favor that behavior. The reward penalty should therefore be harsher. Another factor in the lower accumulated reward by the PD-controller is the high error of the payload swing angle, which should result in a general lower reward.

7.2 Model explainability and stability of model-free learning

Model-Free Reinforcement Learning, and neural networks, are sometimes described as black box models. This is because there is no clear connection to be seen between input and output of the model. What happens in between is a black box. In other words, the model is not explainable. When there is no analytic function that describes the input/output relationship of a system, it is not possible to carry out stability analysis. This means that nothing can be said about the stability of

the system. And when no stability guarantees can be given, the safety of the system can also not be guaranteed. When testing such systems in a laboratory, this can be fine, because safety measures can be taken to avoid accidents with humans. But such a system cannot be deployed in the public, where such safety measure are not in place.

7.3 Viability of Model-Free Reinforcement Learning for control

When using Model-Free RL to make a controller, there are several aspects to consider when assessing its viability. Foremost there is the controller's raw performance which was the focus of the evaluation criteria; how does it perform in the specific task of moving $+10$ in x, while staying at the same z?

When it comes to speed, the performance of the RL-controller was lacking. Speed could potentially be improved with further changes to the RL-model, and more training. One obvious change is to scale the pitch angle to a larger range, such that the agent can learn potential new ways of manipulating the payload to achieve more aggressive maneuvering. The goal of minimizing the payload swing angle error is accomplished by the controller, but with constant small oscillations. One way to potentially improve this behavior is to weight $\dot{e}_\theta$ heavier, such that the agent is disincentivized to make rapid changes in the payload swing angle. A constant error in z is also observed, which is undesired.

Extending the capability of the existing controller to reach any desired x-position is trivial. It is accomplished by having the desired position being "pushed" $10m$ in front of the quadrotor, until the pushed desired position is equal to the final desired position.

Implementing these improvements to the model should result in satisfactory raw performance. Thus, Model-Free RL is viable in terms of making the system accomplish a position control task. But the problem then is the aspect of model explainability. There can be states of the system which are either rarely or never seen in training, where the RL-model performs significantly worse, so-called "edge cases".

As long as the RL-controller's inner workings are not fully understood, it is not viable from a safety perspective. This is a major shortcoming of Model-Free vs Model-Based RL, where the workings of a Model-Based RL model is easier to interpret, because it is based on a model of the system. But again, when modelling the system is out of the question, it is necessary to come up with alternatives. And that is exactly the strength of Model-Free RL; the capability of learning an input/output mapping without any knowledge of the system model, such that the system exhibits a desired behavior according to a reward function. Even though stability guarantees cannot be given, the issue can partially be overcome with throughout training and testing of the controller, so the behavior of the model is known for as large a part of the observation space as possible. This can be in the form of increasing variance of observations seen by the agent and by changing conditions under training, such as starting state, desired position etc.

7.4 Future work

The learned behavior of the RL-controller exhibited the desired low payload swing angle error, but did not achieve the desired aggressive movement. It was also seen that the inverse velocity penalty was not significant enough to reward faster finishing times more than slower. Tuning reward and termination conditions further is therefore a natural first step in getting a more aggressive RL-controller. An addition that could help stabilize the controller in z, is compensating its thrust according to the same factor used with the PD-controller (3.4). It is a non-linear dynamic, which can be hard for the agent to learn properly.

Another obvious extension is to also control the roll angle of the quadrotor using Reinforcement Learning, opening up for a desired position with a y-component; extending it to 3 dimensions. This increases the action space dimensionality by 1. If the observation space is directly extended, it will mean adding the y positional error, velocity, payload swing angle error, derivative of payload swing angle error and roll angle of the quadrotor. Thus, the dimensionality of the observation space is increased to:

$$D_{\mathbf{o},3D} = D_{\mathbf{o},2D} + 5 = 7 + 5 = 12 \tag{7.4}$$

This is a significant increase in the complexity of the learning problem, which often means a lot more training will be needed to get good results. A possible way to get around an increase in complexity when extending to 3D is to exploit the symmetry of the pitch/roll controller mentioned in subsection 2.2.1. This is done by feeding the corresponding y-axis observations into the RL-controller, and then using that output for roll control.

A more involved extension is transferring the Reinforcement Learning model to a real-time system and testing out the viability of the model in real-time. If the model does not prove accurate enough to transfer it to real-time, another simulator that is tailored to RL can be used instead, for example NVIDIA's Isaac Gym [13].

Chapter 8

Conclusion

In this study, Model-Free RL was used to create a controller for a quadrotor carrying a slung payload. Evaluation criteria of the controller were set to be minimum swing of the carried payload while reaching a desired position in the shortest amount of time. It was tested and evaluated in simulation side by side with a PD-controller tuned for the same task. Using data collected in the evaluation task, I found that the RL-controller performed well in regard to the payload swing, while being slow in the position control task. In contrast, the PD-controller performed the position control task fairly quick, while having a large constant absolute swing angle error.

The PD-controller behavior was expected because it was tuned to be as aggressive as possible without being too affected by the payload dynamics. But the RL-controller was trained with the goal in mind of being aggressive while also keeping the payload swing angle error at a minimum. It did not learn the desired aggressive maneuvering, and instead learned to keep the payload swing angle error low by moving slowly at an almost constant low velocity. Reshaping of the reward function and changes to the termination conditions are therefore necessary steps to incentivize more aggressive behavior. Increasing the pitch angle range could also allow the agent to learn new ways of manipulating the payload where it maneuvers more aggressively.

Model-Free RL has one already well known disadvantage when used in control, and that is the problem of model explainability and stability analysis. When the controller is a black box, with no clear input/output connection, stability analysis of the system can not be carried out. Thereby, no safety guarantees can be given. Using this method of control should therefore be saved for systems which models are too complex to obtain. But with throughout training and testing of the system, some edge cases can be rooted out, and safety of the controller increased.

Because Model-Free RL shows promise in terms of raw performance for control problems, more research into a RL stability analogue and thereby safety guarantees are needed.

Bibliography

[1] E. Snouffer, "Six places where drones are delivering medicines," eng, *Nature medicine*, 2022, ISSN: 1546-170X.

[2] Y. Feng, C. A. Rabbath, and C.-Y. Su, "Modeling of the dynamics of a micro uav with a single slung load," in *Handbook of Unmanned Aerial Vehicles*, K. P. Valavanis and G. J. Vachtsevanos, Eds. Cham: Springer International Publishing, 2018, pp. 1–19, ISBN: 978-3-319-32193-6. DOI: 10.1007/978-3-319-32193-6_108-2. [Online]. Available: https://doi.org/10.1007/978-3-319-32193-6_108-2.

[3] P. Kotaru, G. Wu, and K. Sreenath, "Dynamics and control of a quadrotor with a payload suspended through an elastic cable," eng, in *2017 American Control Conference (ACC)*, AACC, 2017, pp. 3906–3913, ISBN: 9781509059928.

[4] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012. DOI: 10.1109/MRA.2012.2206474.

[5] *Openai spinning up - part 1: Key concepts in rl*, https://spinningup.openai.com/en/latest/spinningup/rl_intro.html, Accessed: 24-05-2022.

[6] *Openai spinning up - part 2: Kinds of rl algorithms*, https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html, Accessed: 24-05-2022.

[7] *Mathworks - reinforcement learning using deep neural networks*, https://se.mathworks.com/help/deeplearning/ug/reinforcement-learning-using-deep-neural-networks.html, Accessed: 15-05-2022.

[8] *Reinforcement learning: an introduction*, eng, Second edition., ser. Adaptive computation and machine learning series. Cambridge, MA: The MIT Press, 2018, ISBN: 9780262039246.

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," eng, 2018.

[10] T. G. Lorenz Meier, *3dr iris quadrotor sitl model*, https://github.com/PX4/PX4-SITL_gazebo/tree/master/models/iris, Accessed: 30-05-2022.

[11] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *Openai gym*, 2016. eprint: arXiv:1606.01540.

[12] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.

[13] V. Makoviychuk, L. Wawrzyniak, Y. Guo, *et al.*, *Isaac gym: High performance gpu-based physics simulation for robot learning*, 2021.

[14] Q. Huang, "Model-based or model-free, a review of approaches in reinforcement learning," in *2020 International Conference on Computing and Data Science (CDS)*, 2020, pp. 219–221. DOI: 10.1109/CDS49703.2020.00051.

[15] *Openai spinning up - part 3: Intro to policy optimization*, https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html, Accessed: 29-05-2022.