Model Free Optimal Control Approach for UAVs

Aditya Deole

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Mechanical Engineering

University of Washington

2020

Committee:

Mehran Mesbahi

Santosh Devasia

Steve Shen

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Model Free Optimal Control Approach for UAVs

Aditya Deole

Chair of the Supervisory Committee:
Professor Mehran Mesbahi
Department of Aeronautics and Astronautics

This thesis discusses use of model free control algorithms for application on an UAV. The work contrasts use of LQR based methods to conventional model free learning based on neural net approximators which do not provide guarantees of optimal solution. The model free control methods discussed here are based on discrete time linear systems with LQR based costs that have proven convergence to the optimal solution. We discuss Q-learning algorithm for LQR and a policy gradient methods with a variation. We see applications in simulations for a noisy measurement case with sub-optimal controller as well as a scenario where the dynamics of system has been altered due to disturbances or manipulation.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ACKNOWLEDGMENTS

I to express sincere appreciation to RAIN Lab members and express my gratitude to my advisor Prof. Mehran Mesbahi for his valuable feedback and support both technical and moral throughout this endeavour. I would also like to thank UW Mechanical Engineering Dept. for providing me with this opportunity.

Furthermore I would especially like to thank Tom Miesen, Siavash Alemzadeh and Jingjing Bu for their valuable input and feedback. Finally I also thank my friends and family for their support.

DEDICATION

to my grandmother Hemlata Chitale and my mother for persisting through the trying times.

Chapter 1

INTRODUCTION

In recent years there has been a constant shift in the control community towards the application of data based learning approaches as opposed to optimal or robust control. Machine learning seems to have an impact in almost all the facets of academia [16]. Robotics and controls are also at the forefront of novel AI applications [17]. There have been numerous AI applications in feedback and sensing. Machine vision is one such application which has developed into a field of its own. Vision based applications like depth perception, classification, and state estimation are used frequently by the industry. The use mathematical tools like linear regression, curve fitting or least square estimation that form the basis of ML have been well established since a long time. The reason for the recent boom in ML applications has been a rise in computational capabilities through unprecedented processing power of graphics processing units and availability of tensor based open-source libraries. The shift to machine learning has come as now we can solve complex approximation problems through use of neural networks. We can fit high dimensional nonlinear functions in differentiable neural networks. They can be seen as weighted sums of simpler non linear functions with known gradients. Depending on the complexity of the original functions we can select number of basis functions in a neural network. This flexibility has made use of machine learning techniques much easier.

Our focus at RAIN lab mainly lies in development of the learning control setups in the field on multi-agent control. Specifically in we use our quadrotor test-bed swarm as an application platform. The idea of this thesis is to look for learning algorithms that could be used to control a UAV with unknown dynamics. Since in aviation and transportation industries we need to provide safety guarantees before introducing any fly-by-wire features we want to focus our applications

with learning approaches where we can provide a convergence guarantee. We will first explore some commonly used ML control approaches and then cite the linear system based methods we use here. The problem we will look at is a setup where we have a UAV with slightly modified dynamics in cases such as object manipulation or battery drain, where we still have control of the vehicle but its not optimal. We want to converge to a optimal controller in those scenarios.

We will talk about the hardware setup in this section before we discuss the system dynamics with LQR and PID control. Then we discuss the Q-learning and gradient based learning methods and lastly solve the problem discussed above using both algorithms.

Let us now take a look at some model free learning algorithms. In general every machine learning problem involves fitting a function to a series of sequential input-output data points [12]. A class of prevalent ML problems is **Supervised learning** [23] where learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training data-set, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. We see this application in machine vision where object identification can used to avoid obstacles in an UAVs field of view. Self driving cars use this type of algorithm to identify lanes and traffic in cluttered space

Another set of applications are **Unsupervised machine learning** algorithms where the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system does not figure out the right output, but it explores the data and can draw inferences from data-sets to describe hidden structures from unlabeled data. We often see this as a state estimation application in machine vision where either the object trajectory or self state estimation needs to be done based on time series image inputs. This is also used as a system identification setup where we try and fit a system to state transition data.

There are also other types of learning algorithms like semi-supervised learning and imitation learning that employ similar tactics.But we are specially interested in a class of learning problems called **reinforcement learning**. This class of problems establish an action-reward based mechanism. The decision maker or the agent interacts with an environment and through a series of actions discovers errors and rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. By designing a reward system we create a specific context through which agents determine an ideal behaviour in the environment.

1.1 Learning Setup

In a general RL setup, the agent observes the state s_t from the environment at time t. The agent interacts with the takes action a_t in state s_t and observes the state transition s_{t+1} while receiving a reward attached to this state given by r_t. This forms a data point called a tuple (s_t, a_t, r_t, s_{t+1}). The goal of this agent is to learn a policy(control strategy) π that maximises the expected return(cumulative). This policy tells us what is the optimal action in any given state. This is the same problem as optimal control problem where the policy is based on minimising the cost through state feedback.

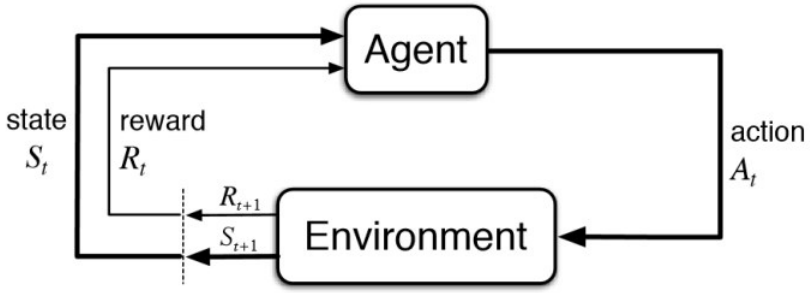

Figure 1.1: Reinforcement learning setup.

This is the type of model free setup we focus on in our application. The system or environment here is unknown where we only receive a reward and can observe a state transition for our actions.

The algorithms we will look at namely Q-Learning and gradient search fit in the reinforcement class. Thus let us look at some basics of RL.

RL setups can be described as *Markov decision processes* which consist of:

- A set of states X, and a distribution of stating states $p(x_0)$.
- A set of actions
- Transition dynamics $T(s_{t+1}|s_T, a_t)$ that map state action pair to next state.
- An instantaneous reward function $R(s_t, a_t, s_{t+1})$
- A discount factor $\gamma \in [0,1]$ to vary emphasis on immediate rewards.

As mentioned earlier the goal of RL is to find optimal policy π^* that gives maximum expected return.

$$\pi^* = arg\max_{\pi} \mathbb{E}[R|\pi] \tag{1.1}$$

The two main approaches to solving RL problems are value based and policy based methods

Value function methods are based on estimating expected return or value of each state. The value function for state s_t while following policy π henceforth is given by:

$$V^*(s) = \mathbb{E}[R|s,\pi] \tag{1.2}$$

Since in the RL setup we do not know the state transition properties T, we define a separate state-action value or quality function $Q^*(s,a)$

This setup is essentially Q-learning, where the best policy given $Q^*(s,a)$, can be found by choosing a greedy action at every step: $arg\max_{\pi} Q^*(s,a)$.

The learning process for this Q-function uses dynamic programming for solving a Bellman equation based on the MDP.An application of this setup is seen in DQN(Deep Q-Networks) algorithms where the Q-function is approximated by a neural network. These algorithms operate under a discrete time and discrete space setting where both state and action space are segregated. For this grid world created the Q-function assigns a Q-value for each state action pair.

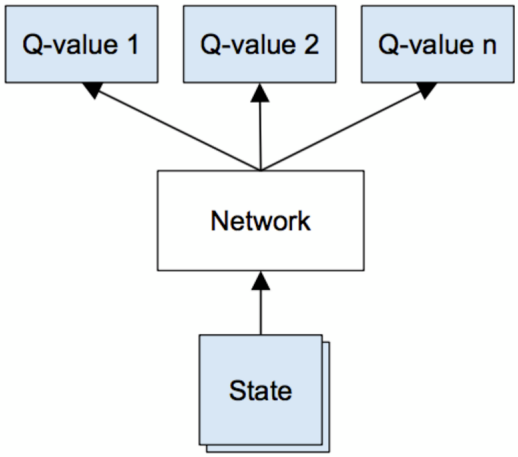

Figure 1.2: DQN setup.

Another class of algorithms are policy search methods that search for an optimal policy. These methods use neural nets to approximate the policies. The parameters of these policies are usually updated by gradient based methods called back propagation. Gradients provide a direction towards improvement of policies. In order to compute the expected return, average over plausible trajectories is picked. Here REINFORCE rule is a commonly used tool to generate the gradient of an expectation over a function f o random variable X wrt parameters θ:

$$\nabla_\theta \mathbb{E}_\mathbb{X}[f(X;\theta)] = \mathbb{E}_\mathbb{X}[f(X;\theta)\nabla_\theta logp(X)] \tag{1.3}$$

The DQN algorithm discussed earlier is very effective for systems with low dimensionality but complexity exponentially increases for the network for higher dimensional systems. Moreover continuous systems cannot be emulated in this architecture. Policy search algorithms also cannot

deal with large systems on their own.Therefore some algorithms to tackle this problem, mix policy search with value function approximation. Essentially it is a approximation of state to action transition defined by policy to a neural network. This is called a actor critic approach which essentially employs an approximator for policy and one for a critic(value function)

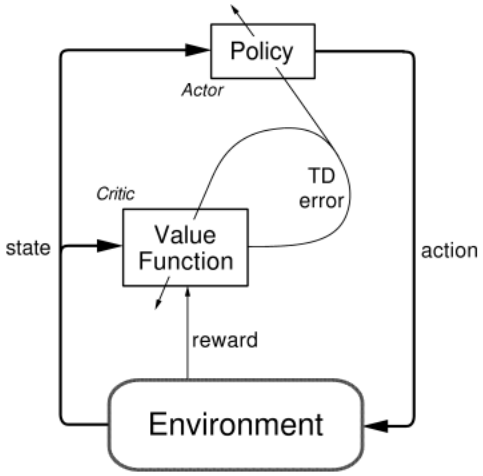

Figure 1.3: Actor Critic setup.

An important aspect of actor-critic algorithms is data generation or exploration. Based on how they generate the data we can segregate the algorithms as policy based or off-policy algorithms. Off-policy algorithms like Deep deterministic policy gradient(DDPG) [13] estimate the total return for the state-action pair assuming a greedy policy were followed despite the fact that it's not following greedy policy. On the other hand algorithms like Trust region policy optimisation estimate their returns for state-action pairs assuming current policy was followed. Both these classes of deep RL(algorithms with more than 3 layers of nn) algorithms [10] are seen to be used in the model free setting in robotics. These algorithms are also preferred over DQN for their ability to handle high dimensional systems due to their depth.

Now we take a look at some of the current applications of deep reinforcement learning algorithms with quadrotor setups. Danial et al. [22] apply DDPG to a spinning UAV. They train the

UAV in a simulator and found the final control to be better performing than a PID controller. A similar application by Hwangbo et al [11] is widely cited uses a policy optimization algorithm that outperforms TRPO and DDPG algorithms for way-point tracking on a UAV. Meanwhile Sampedro et. al. [2] emphasise use a realistic simulation and demonstrate DDPG for landing a UAV on a moving platform. The problem that arises with the use of deep neural networks is that they are nebulous black boxes of sorts. We can not guarantee if they would converge to a optimal solution. Moreover the cost structure is not guaranteed to be convex. We can not be sure if a global minimum for the cost or maximum for the reward function is achieved.

We also looked at system identification as a model free setting [9]. This involves approximating the model of the system based on series of input-output time series data. The data is usually generated by exploring the environment through some randomly generated series of inputs. We can fit the data to a deep neural network or to a linear system. We can then use this derived model to generate an optimal controller. The issues here are high data requirements for this method for getting closer to the actual model. Moreover the system identification usually gives us a possible solution out of the family of systems that could generate the previously generated data set. The other issue arises with slight inaccuracies of system ID for the systems which are close to being unstable(one of the system eigenvalues is close to the unit circle boundary), if the derived model is very close to on the system stability boundary it can easily become unstable. Usually the system ID approaches are data exhaustive but one of the low requirement methods is use of dynamic mode decomposition for control [19] where author discusses use of DMD to extract low-order models from high-dimensional complex systems. An alternative approach that uses system ID is use of adaptive MPC for quadrotor control [18], where the agent solves an optimisation problem in a forward moving window. Although this method is very efficient it is computationally very costly to solve the minimisation problem alongside system ID.

In classical control we have another model free setup of the PID(Proportional, Integral and Derivative) control. Essentially for a PID controller all we need is a time series of feedback error

which we can use to tune the parameters of PID controllers. We can use classical control methods to derive equations for theses parameters but assuming we do not have the access to the model most applications choose to manually tune theses parameters. Some approaches [20] [4] use RLS algorithms to fit the parameters to a previously known value. There are methods that use a parameter based cost and try to solve the optimisation problem to minimise the cost wrt parameters. Some recent studies also work with tuning PID parameters with actor-critic network setups [1].

The model-free setups we choose to explore in this thesis are based on the premise that they can guarantee a fast convergence to an LQR solution. The general LQR problem acts as a rewards based action similar to Q-learning. Moreover the value function in LQR problems is quadratic so we are guaranteed to converge to global minimum or an optimal solution. Thus we choose the LQR cost as our reward system for model-free setup. We assume that having access to observed states and actions gives us the instantaneous cost for the LQR model. We will look at a LQR based Q-learning setup [6] by Bradtke and a LQR based first order gradient search [8].

1.2 Hardware Setup

Figure 1.4: Quadrotor test-bed.

The hardware setup at RAIN Lab consists of swarm of nano-quadrotors fabricated using 3-D printed frames. The quadrotor is assembled with four ESC controlled motors arranged in a cross pattern bisecting the roll and pitch axes. The internal attitude hold requires an hover stabilisation, thus to sense the roll, pitch and yaw along with their respective rates we have an on-board iner-

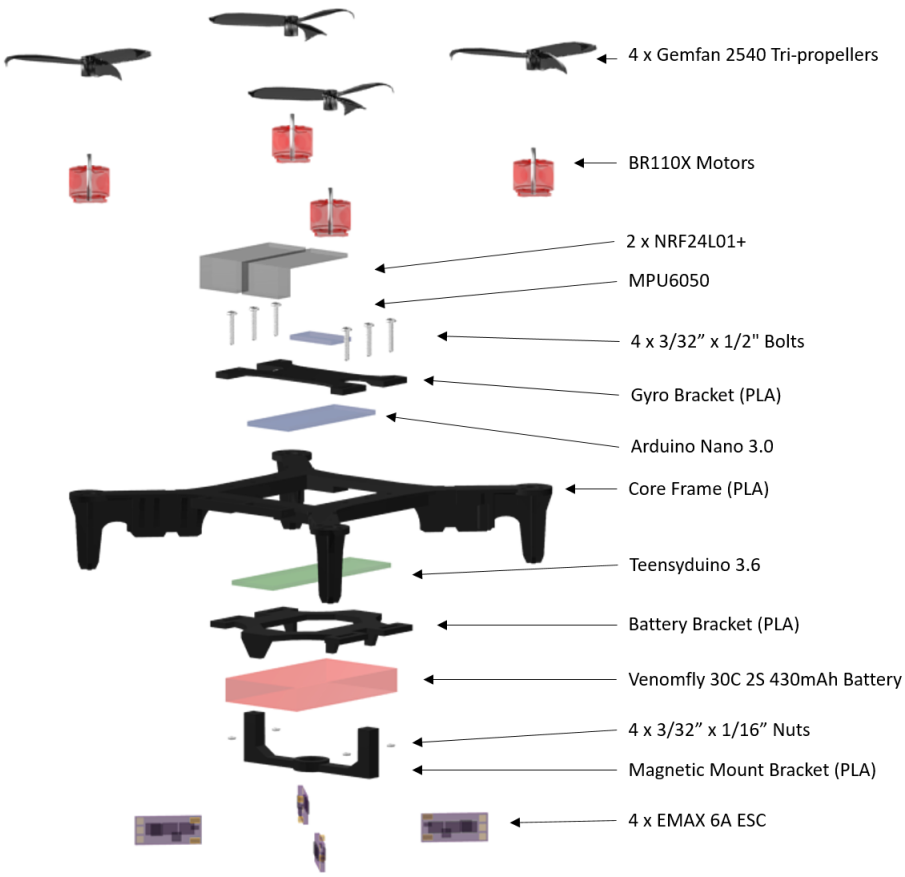

Figure 1.5: Quadrotor Assembly.

tial measurement unit(IMU) which consists of a gyroscope and an accelerometer. We use sensor integration to combine rate updates and force updates from gyroscope and accelerometer respectively to obtain the attitude angles. Since this update requires a faster control cycle we use a pilot controller that handles only the attitude feedback and hover stabilisation through the motor outputs. For slower but complex maneuvers like way-point tracking or trajectory planning we have a separate navigator IC that sends desired attitude to the pilot depending on required action. The navigator still requires a global position feedback to localise itself. This GPS like update is provided by a motion capturing system called VICON. The quadrotor has reflective attachments that feed their position to multiple cameras of the VICON system. An external ground-station is then

required to send this feedback of quadrotor position generated by VICON to the quadrotor. We also need to send desired position to the navigator IC from this ground-station. Thus we have a transmitter-receiver module on-board to handle all the communication. We can send trajectories or desired goal points to the navigator through this ground-station as well.

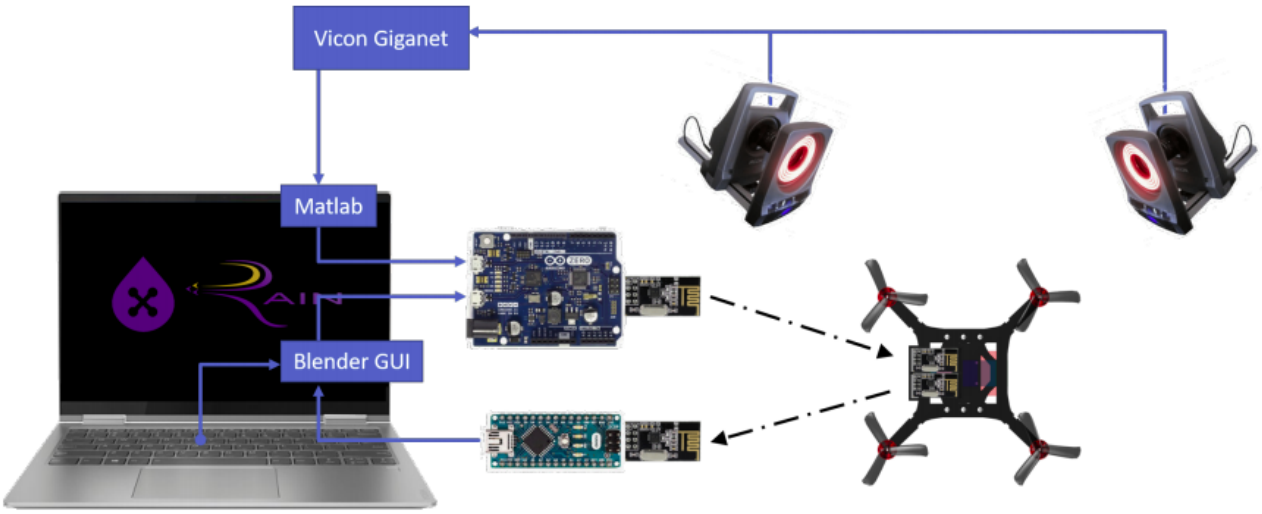

Figure 1.6: Quadrotor communication architecture.

Chapter 2

SYSTEM DYNAMICS

The control system in focus here is an UAV in our case a quadrotor. This analysis looks at dynamics of the non-linear system with 12 states and 4 actions.The states six orientation states and six positional states. The orientation consists of attitude (roll(θ),pitch(ϕ) and yaw(ψ)) of the quadrotor and its corresponding rates or angular velocities. Similarly the positional states are three dimensional (x,y and z) and velocities in all those directions.

2.1 System Dynamics

The non-linear model shown below show that the attitude dynamics are completely independent of positional dynamics. The positional dynamics also can be seen as independent of attitude if we consider the quadrotor near hover and as a point object.

$$\begin{aligned} \dot{\theta} &= \phi\psi\frac{(I_y - I_z)}{I_z} + d\frac{u_2}{I_x} \\ \dot{\phi} &= \theta\psi\frac{(I_z - I_x)}{I_z} + d\frac{u_3}{I_y} \\ \dot{\psi} &= \theta\phi\frac{(I_x - I_y)}{I_z} + a\frac{u_4}{I_z} \end{aligned} \tag{2.1}$$

$$\begin{aligned} \dot{x} &= \frac{u_1}{m}(cos(\theta)sin(\phi)cos(\psi) + sin(\theta)sin(\psi)) \\ \dot{y} &= \frac{u_1}{m}(cos(\theta)sin(\phi)sin(\psi) - sin(\theta)cos(\psi)) \\ \dot{z} &= -g + \frac{u_1}{m}(cos(\phi)cos(\theta)) \end{aligned} \tag{2.2}$$

Here the constants a, d are quadrotor dimensions and I_x, I_y and I_z are moments of inertia along all three axes, also g is the gravitational constant.

The inputs u_1, u_2, u_3 and u_4 are given by:

$$u_1 = m_1 + m_2 + m_3 + m_4 \tag{2.3}$$

$$u_2 = m_1 - m_2 + m_3 - m_4$$

$$u_3 = -m_1 - m_2 + m_3 + m_4$$

$$u_4 = -m_1 + m_2 + m_3 - m_4$$

(2.4)

Where m_1, m_2, m_3 and m_4 are motor thrust values for the quadrotor. Appendix A mentions the linear format of this system

2.2 Control

The control structure employed by us is a cascaded control where we have a inner PID controller for stabilizing the quadrotor. This loop can be independently used for hover application. The inputs are attitude set-points and outputs are the motor corrections.

The outer controllers are position and thrust controllers that are LQR or PID driven. The position controller takes in desired position set-points and returns desired attitude set-points as input for the inner controller. The thrust controller is independently used to output desired thrust based on inputs for elevation.

This system is not inherently stable with just the inner loop. We need a preliminary stabilizing position controller to make the system stable so we can apply learning based algorithms

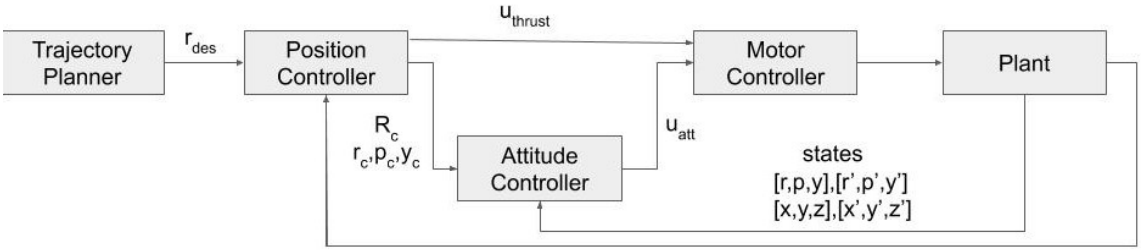

Figure 2.1: control system structure.

2.2.1 Hover control

The hover control implementation used here involves a PD controller for roll, pitch and yaw. The attitude corrections are mapped to the corresponding motor thrust based on the their orientations.

the input at any instance for any PID system is given by:

$$u(t) = u_{ref}(t) - (k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de}{dt}) \tag{2.5}$$

here $u_{ref}(t)$ is the reference input for stabilization, $e(t)$ is given by instantaneous error or

$$e = y - r$$

. The PID parameters for proportional, integral and derivative controller are given by k_p, k_i and k_d.

Similarly the PD controller for attitude control of the quadrotor is given by following control law for output$\Theta = \begin{bmatrix} \theta & \phi & \psi \end{bmatrix}^T$ and the reference $\Theta_{des} = \begin{bmatrix} \theta_{des} & \phi_{des} & \psi_{des} \end{bmatrix}^T$ which is $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ for hover.

$$u(t) = u_{th}(t) - (K_p e(t) + K_d \frac{de}{dt}) \tag{2.6}$$

where $u_{th}(t)$ is the thrust for each motor required for hover $\begin{bmatrix} \frac{mg}{4} & \frac{mg}{4} & \frac{mg}{4} & \frac{mg}{4} \end{bmatrix}$
Here we have three independent controllers for roll, pitch and yaw summed together linearly as all of them are completely independent of each other. Thus simplifying further the controller is a combination of the following:

$$u_\theta = k_{p\theta} e_\theta(t) + k_{d\theta} \frac{de_\theta}{dt}$$

$$u_\phi = k_{p\phi} e_\phi(t) + k_{d\phi} \frac{de_\phi}{dt}$$

$$u_\psi = k_{p\psi} e_\psi(t) + k_{d\psi} \frac{de_\psi}{dt}$$

The linear combination of all the above three controllers gives us:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \frac{mg}{4} \\ \frac{mg}{4} \\ \frac{mg}{4} \\ \frac{mg}{4} \end{bmatrix} u_{th}(t) - (K_p \begin{bmatrix} e_\theta \\ e_\phi \\ e_\psi \end{bmatrix} + K_d \frac{d}{dt} \begin{bmatrix} e_\theta \\ e_\phi \\ e_\psi \end{bmatrix}) \tag{2.7}$$

Since the three dimensional error has to be mapped to each of the four motors that correspond to the correct orientation. Thus the proportional and derivative controllers K_p and K_d have the following form:

$$K_p = \begin{bmatrix} k_{p\theta} & -k_{p\phi} & -k_{p\psi} \\ -k_{p\theta} & -k_{p\phi} & k_{p\psi} \\ k_{p\theta} & k_{p\phi} & k_{p\psi} \\ -k_{p\theta} & k_{p\phi} & -k_{p\psi} \end{bmatrix}, K_d = \begin{bmatrix} k_{d\theta} & -k_{d\phi} & -k_{d\psi} \\ -k_{d\theta} & -k_{d\phi} & k_{d\psi} \\ k_{d\theta} & k_{d\phi} & k_{d\psi} \\ -k_{d\theta} & k_{d\phi} & -k_{d\psi} \end{bmatrix}$$

2.2.2 Thrust control

The attitude control law takes care of the orientation of the quadrotor. One of the important aspects in control of UAVs is its altitude from the ground. In the case of a quadrotor, proving a combined thrust equivalent to the mass through all the motors gives us a steady altitude hold.
Since orientation is independent of position we can add a outer controller over position only.

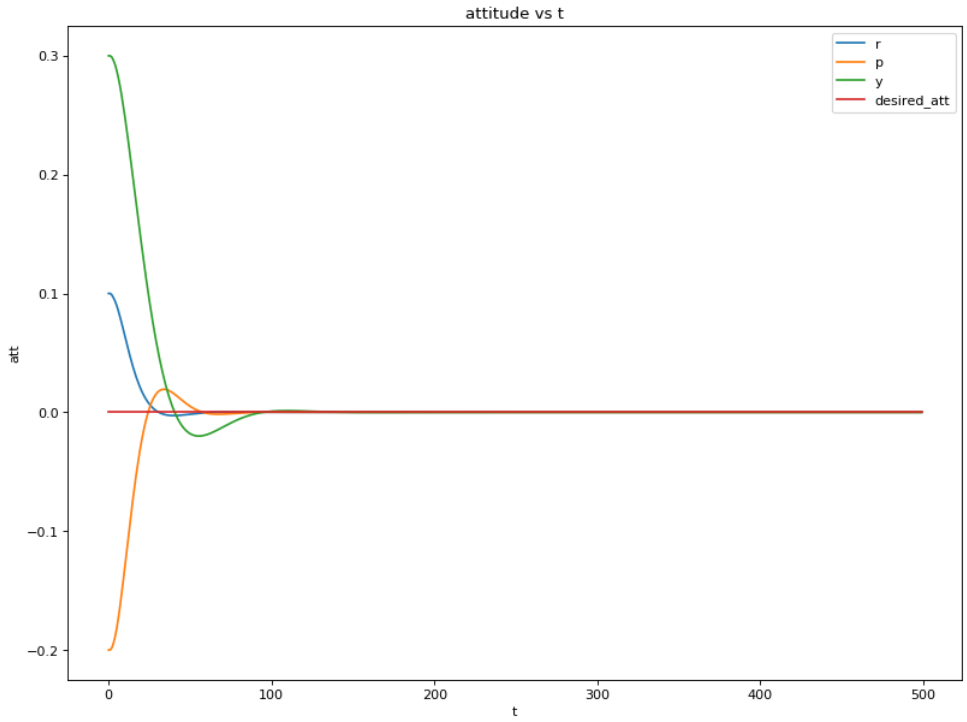

Figure 2.2: Attitude stabilisation.

We can momentarily increase or decrease the thrust through all the motor to effectively create a net force which is not canceled by gravity. Thus we can have a altitude controller by changing motor thrusts.

Here a PID controller for altitude hold or position hold in z direction is used.

So in this case the output is z position and reference is desired altitude.

Since all motors must behave in a congruous manner after the attitude stabilization. The mapping of one dimensional error to four motor inputs can be handled by a scalar controller parameter.

$$u(t) = u_{th}(t) - (Kt_p e(t) + Kt_d \frac{de}{dt}) \tag{2.8}$$

where Kt_p and Kt_d are thrust proportional and derivative controllers.

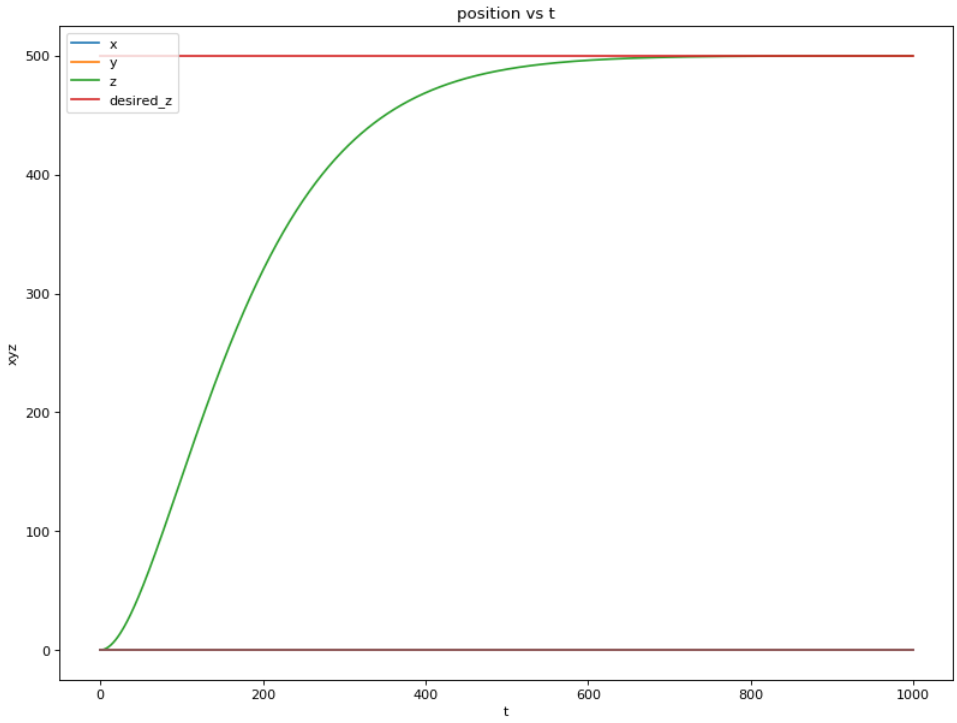

Figure 2.3: Thrust controller achieving desired altitude.

2.2.3 Position controller

The main aspect of control of a quadrotor in a closed environment is tracking a way-point. This can be modified to make a trajectory controller based on desired position velocity and acceleration. This strategy is achieved by making use of the independence of attitude and position dynamics of the UAV. Thus here we have an inner attitude controller that stabilizes the orientation and an outer position controller which feeds desired attitude set-points to the inner attitude controller.

The control structure shown above uses the trajectory planner as the input which feeds desired trajectory ($[x_{des}, y_{des}, z_{des}], [\dot{x}_{des}, \dot{y}_{des}, \dot{z}_{des}]$) to the position controller. The position controller is a PID controller that gives the attitude set-point outputs and also feeds desired altitude to the thrust controller.

The attitude controller as suggested before gives motor thrust as output similar to the thrust controller. The motor controller converts these force inputs to data pulses that the embedded controller feeds to the motor. The inertial measurement sensor and the vicon system on the quadrotor gives us the full state feedback which includes attitude, attitude rates, position and velocities. The state information is fed back to the two controllers ie attitude and position controllers to close the feedback loop.

The position controller works as follows: The current trajectory r_i is given by $r_i = \begin{bmatrix} x & y & z \end{bmatrix}$, similarly the desired trajectory is defined by a combination of r_i and $\dot{r}_i$. We generate $\ddot{r}_i^{des}$ as follows

$$(\ddot{r}_i - \ddot{r}_i^{des}) + k_p(r_i^{des} - r_i) + k_i \int (r_i^{des} - r_i)dt + k_d(\dot{r}_i^{des} - \dot{r}_i)$$

Using the quadrotor dynamics we can generate the attitude set-points from the desired acceleration trajectory points:

$$\begin{aligned} \theta^{des} &= (\ddot{r}_1^{des} sin(\psi) - \ddot{r}_2^{des} cos(\psi))/g \\ \phi^{des} &= (\ddot{r}_1^{des} cos(\psi) - \ddot{r}_2^{des} sin(\psi))/g \end{aligned} \tag{2.9}$$

The desired roll and pitch obtained from the above controller and desired yaw is transferred to the attitude controller.
Another output of this controller is part of the thrust controller. The desired altitude from the trajectory is used as input to thrust controller seen in the previous section.

2.2.4 Velocity Controller

The velocity controller is a simple add-on to the position controller which calculated desired position based on a external trajectory controller.

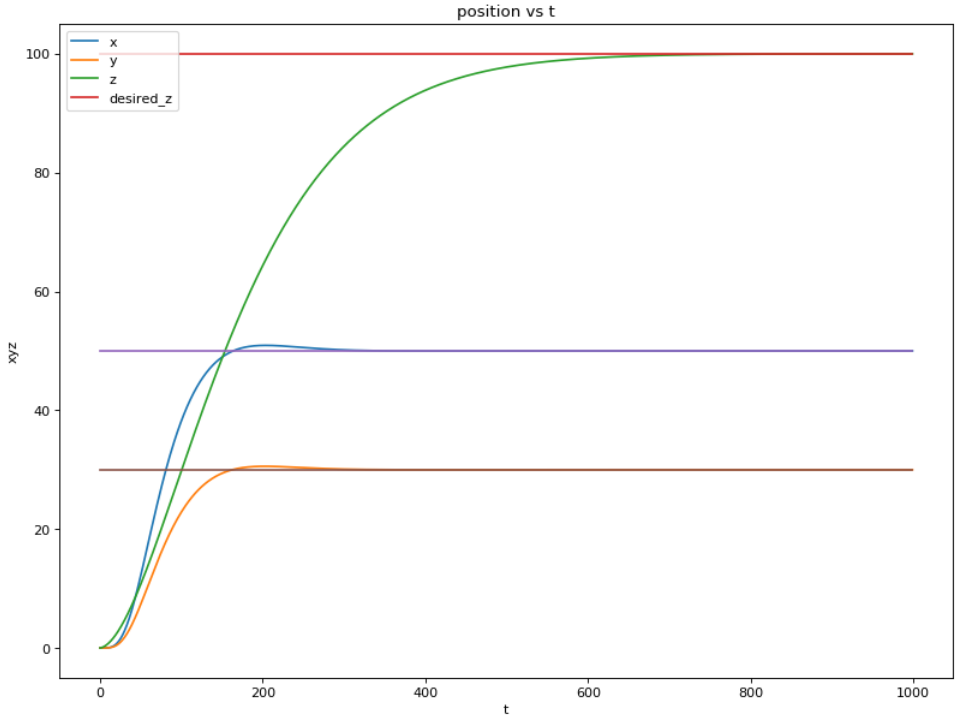

Figure 2.4: Position controller achieving desired positions and altitude.

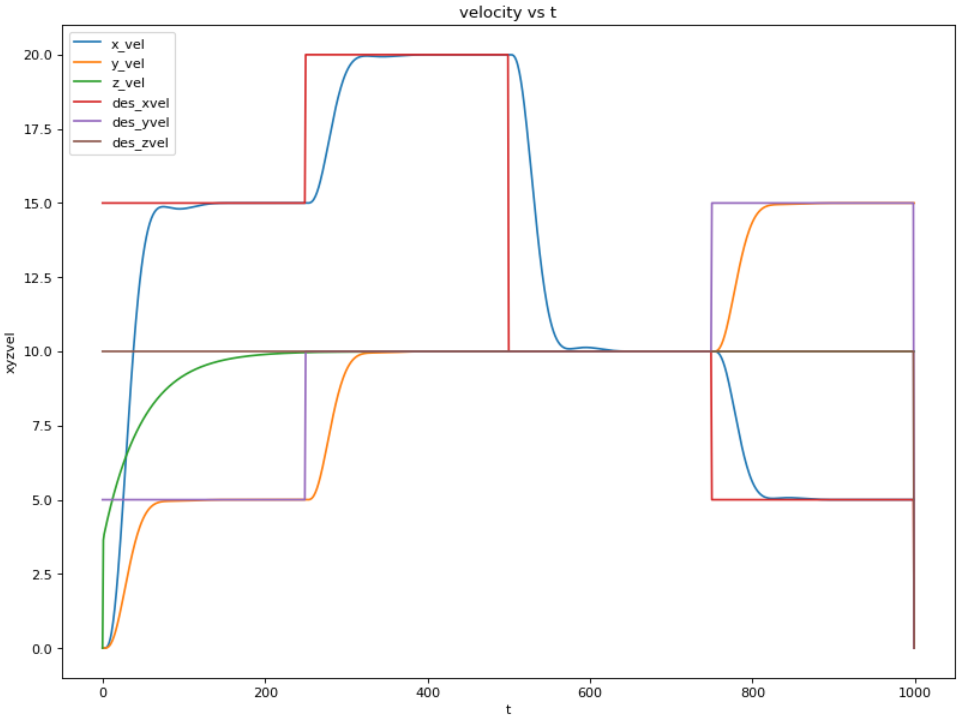

Figure 2.5: Velocity controller achieving desired velocities.

2.3 Linear System

The system presented here can be easily segregated into attitude stabilization system and an independent free body system with three degrees of freedom actuated by roll, pitch and thrust respectively.

We use the outer system as a focus the adaptive algorithms since linearity is an underlying assumption for Q-learning and system ID algorithms and it is easier to linearize this six dimensional system than including the linearity caused by attitude states.

The states for this system are $q = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}$ and input are set as $u = \begin{bmatrix} \phi & \theta & th \end{bmatrix}$.
The th component of the input is the total thrust provided by all the motors which controls z position and velocity.
The system is linearized near the equilibrium stable hover state. We use small angle approximation for input for roll and pitch. Since the roll and pitch angles are also heavily constrained by the inner controller the linearization holds for the whole work space.

$$\dot{q} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} q + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & \frac{1}{m} \end{bmatrix} u \tag{2.10}$$

The LQR control setup for this system is based on a discrete time version of the above system. Using the time interval for position control application we discretize the system and use the following LQR control law.

Consider the discrete-time system:

$$x_{k+1} = A_k x_k + B_k u_k \tag{2.11}$$

The LQR state feedback control to find optimal u_k to minimize the following quadratic value function.

$$V_t(z) = \min_{u_1..u_N} \frac{1}{2} x_N^T Q_N x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_N^T Q_N x_N + u_k^T R_k u_k \tag{2.12}$$

We find that V_t is quadratic ie $V_t(z) = z^T P_t z$, where $P_t = P_t^T \geq 0$

We find P_t recursively, working backwards using the algebraic riccati equation.

Using the Dynamic Programming principle

$$V_t(z) = \min_w (z^T Q z + w^T R w + V_{t+1}(Az + Bw)) \tag{2.13}$$

The above equation is a sum of current cost incurred and min cost-to-go from where you land.// So as stated earlier by assuming $V_{t+1}(z) = Z^T P^{t+1} Z$, with $P_{t+1} = P_{t+1}^T \geq 0$

By DP,

$$V_t(z) = z^T Q z + \min_w (w^T R w + (Az + Bw)^T P_{t+1} (Az + Bw)) \tag{2.14}$$

Taking derivative w.r.t. w to zero:

$$2w^T R + 2(Az + Bw)^T P_{t+1} B = 0 \tag{2.15}$$

hence optimal input is

$$w^* = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \tag{2.16}$$

Solving the DARE we get

$$V_t(z) = z^T P_t z \tag{2.17}$$

where

$$P_t = Q + A^T P_{t+1} A - A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A \tag{2.18}$$

Coclusively we have

$$u = -Kx \tag{2.19}$$

where

$$K = (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A \tag{2.20}$$

Soour quadrotor LQr systme is described by:

$$\begin{aligned} x_{k+1} &= Adx_k + Bdu_k \\ u_k &= -Kx_k \end{aligned} \tag{2.21}$$

where

$$Ad = \begin{bmatrix} 1 & 0 & 0 & 0.0169 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0.0169 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.0169 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, Bd = \begin{bmatrix} 0 & 1.401 & 0 \\ -1.401 & 0 & 0 \\ 0 & 0 & 0.001 \\ 0 & 0.017 & 0 \\ -0.017 & 0 & 0 \\ 0 & 0 & 0.130 \end{bmatrix}$$

$$Q = 10^{-5} \begin{bmatrix} 2.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{bmatrix}, R = 10^3 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^{-5} \end{bmatrix}$$

This helps us achieve similar performance to the PID position and velocity controllers. The only difference is that the desired attitudes and thrust are the direct inputs generated by the following feedback gain matrix.

$$\begin{bmatrix} 0 & -1.554 \times 10^{-4} & 0 & 0 & -2.0334 \times 10^{-4} & 0 \\ 1.554 \times 10^{-4} & 0 & 0 & -2.0334 \times 10^{-4} & 0 & 0 \\ 0 & 0 & 5.6604 & 0 & 0 & 2.8067 \end{bmatrix}$$

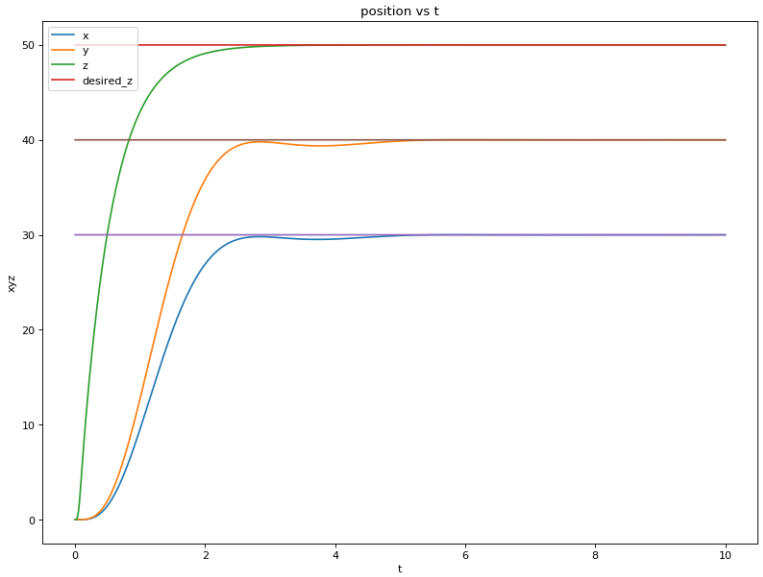

Figure 2.6: LQR Position controller achieving desired positions and altitude.

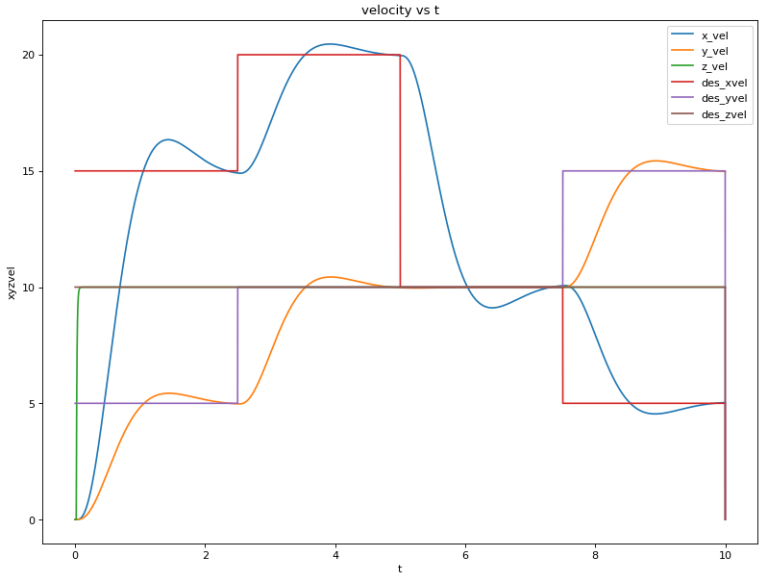

Figure 2.7: LQR Velocity controller achieving desired velocities.

Chapter 3

GRADIENT DESCENT ON LQR

The current reinforcement learning algorithm that are in vogue like DQN, DDPG, TRPO etc can be effectively used with some continuous control problems. These are RL approaches used for uncertain dynamical systems which used sample based optimization. For RL algorithms based on nonlinear neural net approximators we have little theoretical understanding of their efficiency and there is lack of optimal convergence guarantees. But at the core of all these algorithms is an iterative gradient update step that is called back propagation. This is just a gradient of cost or loss function taken wrt parameters layer by layer which is fed back to the function approximators to correct themselves. Simply put it is a step forward in the direction of steepest gradient. In case of non linear functions and non convex costs we can provide no guarantees for convergence to optimal controllers. Conventionally, optimal control techniques like LQR have convex and differentiable cost functions. We can calculate first order gradients and perform a gradient descent on this cost and arrive at an optimal solution. We will look at such an algorithm in this section. In this scenario even if the system is unknown convergence to to lqr solution is guaranteed given the linearity assumptions.[Mesbahi et al. 2019].

3.1 Algorithm

Consider the system:

$$\dot{x} = Ax + Bu \tag{3.1}$$

The following infinite horizon LQR problem:

$$\textit{minimize} \quad \mathbb{E}\left[\sum_{t=0}^{\infty}(x_t^T Q x_t + u_t^T R u_t)\right] \tag{3.2}$$

$$\textit{such that} \quad x_{t+1} = Ax_t + Bu_t, \;\; x_0 \sim D \tag{3.3}$$

where initial state $x_0 \sim D$ is assumed to be randomly distributed within its constraint according to distribution D; $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{n\times m}$ define the system; $Q \in \mathbf{R}^{n\times n}$ and $R \in \mathbf{R}^{m\times m}$ are positive definite matrices that parametrize the LQR.

Here state errors are used in the LQR costs. solving the algebraic riccati equation generated by applying the value function approximation on bellman equation. The input response is obtained as

$$u_t = Kx_t \tag{3.4}$$

$$K = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$$

where P is the solution to

$$P_{t-1} = Q + A^T P_t A - A^T P_t B(R + B^T P_t B)^{-1} B^T P_t A \tag{3.5}$$

until P_t converges

For the model free optimization scenario a sample based policy gradient step can be defined. Using zeroth order optimisation we can define following update steps:

Policy gradient

$$K_{n+1} = K_n - \eta \widehat{\nabla C(K)} \tag{3.6}$$

and natural policy gradient

$$K_{n+1} = K_n - \eta \widehat{\nabla C(K)} \hat{\Sigma}_{K_n}^{-1} \tag{3.7}$$

For the given optimization landscape in the following sets of lemmas can help explain why global convergence is guaranteed. First of all the for the convex costs for systems with dimensions less than two we have convex costs which would provide guaranteed convergence simply from the fact that we travel to the steepest gradient direction.

From **Lemma2** [8] we can see that there are systems with $d \geq 3$ where an LQR optimization problem,$min_K C(K)$ is not convex, quasi-convex or star-convex.

In that case we look at a corollary for **Lemma2** (gradient dominance property)[8]:

Corollary4:(Stationary Point Characterisation) *If* $\nabla C(K) = 0$*, then either K is optimal policy or* Σ_K *is rank deficient*

Note that co-variance $\Sigma_K \succeq \Sigma_0 \mathbb{E}_{x_0 \sim Dx_0 x_0^T}$. Therefore this lemma is important in selecting a distribution over x_0 as opposed to a fixed starting point: $Dx_0 x_0^T$ being full rank guarantees Σ_K being full rank, which implies all stationary points are optimal.

The concept of gradient dominance is important in non-convex optimization. If a function is gradient dominated then it implies that if the magnitude of the gradient is small at some x, then the function value at x will be close to that of the optimal function value.

In general a function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be gradient dominated if there exists some constant λ, such that for all x,

$$f(x) - \min_{x'} f(x') \leq \lambda ||\nabla f(x)||^2$$

Gradient dominance property doesn't immediately imply that gradient descent converges quickly to global optima. The function $C(K)$ must be smooth for that case.

The LQR objective cannot guarantee smoothness condition globally i.e. the objective becomes infinity when matrix $A - BK$ becomes unstable (has an eigenvalue outside of the unit circle in the complex plane). At the boundary between stable and unstable policies, the objective function quickly becomes infinity.

To solve this issue the policy K_0 is selected that is not too close to the boundary or is sub-optimal.

Here for an unknown A, B, Q and R we use the following algorithm gives procedure to find approximate estimates of both $\widehat{\nabla C(K)}$ and $\hat{\Sigma}_{K_n}^{-1}$

Algorithm 1 Model-Free Policy gradient (and Natural Policy Gradient) Estimation

1: Input K, number of trajectories m, roll out length l, smoothing parameter r, dimension d

2: **for** $i = 1, .., m$ **do**

3: a policy $\hat{K}_i = K + U_i$, where U_i is uniformly drawn at random over matrices whose (Frobenius) norm is r

4: Simulate $\hat{K}_i$, for l steps starting from $x_0 \sim D$. Let $\hat{C}_i$ and Σ_i be empirical estimates:

$$\hat{C}_i = \sum_{t=1}^{l} c_t, \;\; \hat{\Sigma}_i = \sum_{t=1}^{l} x_t x_t^T$$

where c_t and x_t are costs and state on this trajectory

5: **end for**

6: Return the estimates

$$\widehat{\nabla C(K)} = \frac{1}{m} \sum_{i=1}^{m} \frac{d}{r^2} \hat{C}_i U_i, \quad \widehat{\Sigma_K} = \frac{1}{m} \sum_{i=1}^{m} \hat{\Sigma}_i$$

In Summary, the algorithm presented uses a underlying assumption of convexity of cost function w.r.t. K. It starts with a random feasible guess of K for which initial infinite horizon cost is finite. Then we pick an allowed error bound ε for concluding the algorithm. A n-dimensional sphere is then created around K and it is scaled according to the learning rate γ. From the surface of this sphere we pick s randomly selected points. Then we calculate costs at each of these points and pick direction of travel based on least cost value. This gives our supposed gradient and we update our K to this point. If this minimum cost is not smaller than current K we reduce our scaling factor γ, as the minimum now lies inside our scaled sphere. The algorithm stops when the difference between the new minimum and current cost goes below ε

The assumptions held here are that the system is a discrete time linear system. Also we assume that initial cost $C(K_0)$ is finite. The initial state x_0 must also belong to a distribution i.e. $x_0 \sim D$ and is bounded by L.

3.2 Case Studies

Stochastic Gradient Descent on a convex function

Taking a look at how the gradient update step works for a given convex function when we randomly sample for gradients is show in the following example.

For a convex function:

$$\frac{-1}{1+x^2+y^2}$$

We have the following cost function The stochastic gradient update step on this function can be

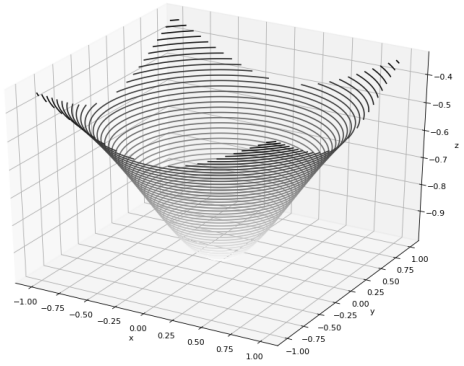

Figure 3.1: two dimensional convex function

easily demonstrated using a contour plot as:

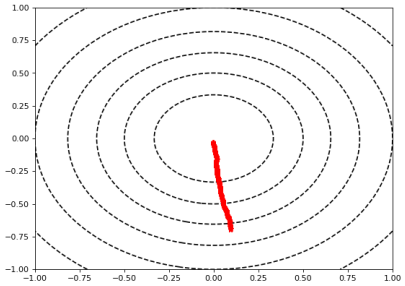

Figure 3.2: Gradient Descent on convex function

Springmass Damper

The double integrator system used here is given by:

$$A = \begin{bmatrix} 0 & 1 \\ \frac{-k}{m} & \frac{-c}{m} \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, Q = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, R = 1$$

where $c = 0.5, k = 1, m = 1$

Since this is a damped system we can see that system is Hurwitz stable without initial controller. So we do not need a sub-optimal controller and initial cost $C(K_0)$ is finite.

As we can see the parameter based cost function $C(K)$ is gradient dominated so we can guarantee a convergence based on gradient update step.

We derive the optimal LQR gain for this system as:

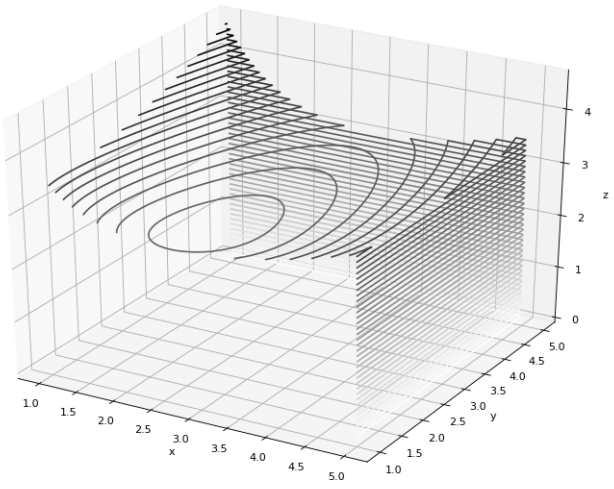

Figure 3.3: cost map on a grid of k parameters.

$$K = \begin{bmatrix} 1.81662479 & 3.35788149 \end{bmatrix}$$

Now applying stochastic gradient descent on the LQR cost within 22 iterations we get the following controller for an error bound of 10^{-9}:

$$\hat{K} = \begin{bmatrix} 1.78465833 & 3.01105708 \end{bmatrix}$$

Chapter 4

Q-LEARNING

We now take a look at reinforcement learning algorithm that uses quadratic function approximator to assign optimal action to a state. Traditionally in Q-learning related algorithms an agent tries to learn optimal policy by training a critic to judge the best action using a sequence of state-action-reward. This is also called an off-policy algorithm as the q-learning function learns from actions that are outside the current policy. Since we require taking random actions a policy is not needed. The agent, or the object making the decision, learns from history of interaction with the environment by maximizing its total reward R by transitioning from state s to s' by taking an action a. The tuple of action a, state transition s - s' and reward r is used to update the Q-function or Q table that defines a value to every possible state action pair. Maximising this Q-value gives you the optimal action for that state. This Q-value is updated using the following temporal difference equation.

$$q(s,a) \leftarrow q(s,a) + \alpha(r + \gamma(q(s',a) - q(s,a)))$$

Here α is the learning rate and γ is the discount factor, we use q notation instead of Q as it conflict with LQR gains.

This algorithm works exceptionally well for smaller dimensional systems that are discretised. We often see demonstrations based on grid worlds with discritised states and actions and objectively designed rewards. For higher dimensional and continuous space systems though the computational requirement on Q-table update grows exponentially. Thus we see use of neural networks as function approximators in algorithms like DQN(Deep Q-Network) or DDQN(Double DQN). These algorithms are seen to work fairly well with high dimensional systems but still require heavy computation for continuous systems. Moreover the main block that keeps us using these algorithms in

real world applications are that we cannot give convergence proofs for problems with non linear function approximators and continuous systems.

To work around this problem we look at a setup provided by Bradtke(insert reference) where he proposes Dynamic Programming based Reinforcement Learning applied to Linear Quadratic Regulators. This algorithm describes an algorithm that has been proven to converge for a large class of LQR problems.

4.1 Algorithm

Consider a deterministic linear, discrete time dynamic system:

$$x_{t+1} = Ax_t + Bu_t \tag{4.1}$$

with feedback control

$$u_t = Kx_t \tag{4.2}$$

Here $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ and $K \in \mathbb{R}^{m\times n}$. The K is chosen such that $A + BK$ has all its eigenvalues inside unit circle or it is Hurwitz stable.

The one-step LQR cost associated with the state is:

$$r_t = x_t^T Q x_t + u_t^T R u_t \tag{4.3}$$

where $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are positive semi-definite matrices.

For the LQR problem we can write the Q function as

$$\begin{aligned} Q_K(x,u) &= R(x,u) + \gamma V_K(f(x,u)) \\ &= x^T Q x + u^T R u + \gamma (Ax+Bu)^T P_K (Ax+Bu) \\ &= \begin{bmatrix} x & u \end{bmatrix}^T \begin{bmatrix} Q+\gamma A^T P_K A & \gamma A^T P_K B \\ \gamma B^T P_K A & R+\gamma B^T P_K B \end{bmatrix} \begin{bmatrix} x & u \end{bmatrix} \\ &= \begin{bmatrix} x & u \end{bmatrix}^T \begin{bmatrix} H_{K11} & H_{K12} \\ H_{K21} & H_{K22} \end{bmatrix} \begin{bmatrix} x & u \end{bmatrix} \\ &= \begin{bmatrix} x & u \end{bmatrix}^T H_K \begin{bmatrix} x & u \end{bmatrix} \end{aligned} \tag{4.4}$$

where $\begin{bmatrix} x & u \end{bmatrix}$ is he column vector concatenation of x and u and H_K is a symmetric positive definite matrix of dimensions $(n+m) \times (n+m)$.

The convergence results for the conventional Q-Learning assumes a discrete time and finite space systems that use look-up tables to represent Q functions. For the LQR domain where the states and actions are continuous and vector spaces, a parameterized representation of Q-function is used. The definition of Q-function shown above has a form of quadratic function of its arguments. Therefore the approximated Q-function used here is also selected to be a linear function of quadratic combinations of vector $[xu]$, $\begin{bmatrix} x & u \end{bmatrix}$.For state $\begin{bmatrix} x_1 & x_2 \end{bmatrix}$ and action $\begin{bmatrix} u_1 \end{bmatrix}$,the Q-function is linear combination of $\begin{bmatrix} x_1^2 & x_2^2 & u_1^2 & x_1x_2 & x_1u_1 & x_2u_1 \end{bmatrix}$. This format allows the usage of linear Recursive Least Square to implement Q-learning in the LQR domain.

The policy based Q-Learning rule is given by:

$$q_{t+1}(x_t,u_t) = q_t(x_t,u_t) + \alpha[r(x_t,u_t) + \gamma q_t(x_{t+1}, Kx_{t+1}) - q_t(x_t,u_t)] \tag{4.5}$$

The policy improvement step for a policy K_k and value function q_K we can find the improved policy K_{k+1}, by defining K_{k+1} as

$$K_{k+1}x = arg\min_u q_K(x,u) \tag{4.6}$$

minimizing by differentiating q_K wrt u yield

$$\begin{aligned} u &= -\gamma(R+\gamma B^T P_K B)^{-1} B^T P_K A x \\ &= K_{k+1} x \end{aligned} \tag{4.7}$$

Using Eq(5.4)

$$K_{k+1} = H_{K(22)}^{-1} H_{K(21)} \tag{4.8}$$

This policy K_{k+1} has no higher cost than K_k.

The RLS update step:

Define function Θ for square matrices. $\Theta(K)$ is the vector obtained from the realization $x^T K x = \bar{x}^T \Theta(K)$.

Now we can write

$$Q_K(x,u) = \begin{bmatrix} x & u \end{bmatrix}^T H_K \begin{bmatrix} x & u \end{bmatrix} = \begin{bmatrix} x & \bar{} & u \end{bmatrix}^T \Theta(H_K)$$

Further skipping some steps we can rearrange the Eq(10) to get:

$$\begin{aligned} r_t &= R(x_t, u_t) \\ &= \phi_t^T \theta_K \end{aligned}$$

where

$$\phi_t = \begin{bmatrix} x_t & \bar{} & u_t \end{bmatrix} - \gamma \begin{bmatrix} x_{t+1} & \bar{} & K x_{t+1} \end{bmatrix} \tag{4.9}$$

Recursive least squares (RLS) can now be used to estimate θ_u. The relations for RLS are given by:

$$\hat{\theta}_k(i) = \hat{\theta}_k(i-1) + \frac{P_k(i-1)\phi_t(r_t - \phi^T \hat{\theta}_k(i-1))}{1+\phi^T P_k(i-1)\phi_t} \tag{4.10}$$

$$\P_k(i) = P_k(i-1) + \frac{P_k(i-1)\phi_t \phi_t^T P_k(i-1)}{1+\phi^T P_k(i-1)\phi_t} \tag{4.11}$$

$$P_k(0) = P_0 \tag{4.12}$$

where $P_0 = \beta I$

The algorithm starts with initializing the randomized RLS parameters based on dimensions of the system. Since the algorithm is model free we only need to know the dimension of the state

Algorithm 2 Q-Learning Algorithm

initialize Q-function parameters $\theta_0, \hat{H}_0, P_0$
Set $t = 0, K_0, \varepsilon$.
for Episode $i = 1$ to M **do**
    Initialize the Recursive Least Square estimator
    for $t = 1$ to N **do**
        $u_t = K_{i-1}x_t + e_t$, where e_t is the 'exploration' component of the control signal
        Apply u_t to the system, resulting in state x_{t+1}
        Update Q-function parameters $\theta_i, \hat{H}_i, P_i$, using RLS implementation.
    end for
    Define $K_i = -\hat{H}_{22}^{-1}\hat{H}_{21}$
    Initialise parameters $\hat{H}_{k+1} = \hat{H}_k$
end for

action pair. The transition state is given by an oracle that is either the system or emulates the system. The underlying system must be linear as it was one of the initial assumptions of the algorithm. Also we making sure that the initial controller K_0 is stabilizing or the system is Hurwitz stable. The data matrix H is used to derive the optimal controller as defined above. The length of roll-out N, is a tuning parameter that must be carefully set. The excitation component e is a normally distributed random value that is added to the action. The higher the bound on e faster is the convergence. This means more we explore the environment faster we can converge. The caveat to this rule is that we can not randomly explore the environment in real life systems, there may be action constraints we need to follow so that induces a bound on excitation that increases number of iterations required for convergence. Moreover randomly picking excitation values may also need be feasible for an actual system as it may lead to system failure due to state constraint violation.

We also use an iterative update on the controller using the Q-value approximation. From the derivation of H function shown above we have a formulation of the LQR controller. This updates

the controller in a single step based on past data, whereas the gradient update step we saw earlier was an iterative update on the controller where we update K based on some learning rate. Here from the knowledge of structure of H we can use a quasi newton update step on the Q-learning controller.

We have the quasi-newton rule as:

$$K_{l+1} = K_l - \eta q_n(K_l) \tag{4.13}$$

This quasi newton rule is given by:

$$q_n(K) = 2(R + B^T P B)^{-1}(RK - BPA_K) \tag{4.14}$$

The formulation in term of our H matrix can be written as:

$$K_{l+1} = K_l - \frac{1}{max(eig(2H_{22}))}((H_{22}K_l) - H_{21}) \tag{4.15}$$

This is a slower update step as compared to directly updating Qlearning controller but the advantage gained in this update step is that we can reduce the magnitude of excitation component for the input. This allows us to approach a practical scenario where we have tighter constraints on the input signal.

We see how the algorithm behaves with certain examples of linear systems.

4.2 Case Studies

Take a look at a two dimensional system:

$$A = \begin{bmatrix} 0.0897 & -0.1133 \\ 0.0318 & 0.1146 \end{bmatrix}, B = \begin{bmatrix} 2.9035 & -0.0734 \\ -0.0734 & 3.0137 \end{bmatrix}$$

Since the system has all its eigenvalues inside the unit circle on omplex plane we can select a null initial controller and it can work with our setup.

For LQR parameters:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

And the LQR controller can be given by:

$$K = \begin{bmatrix} -0.0278 & 0.0342 \\ -0.0100 & -0.0336 \end{bmatrix}$$

From the Q-learning algorithm we get for 25 iteration an l_2 norm on Q-learning controller error wrt LQR controller within bound of 10^{-9}

Progression of error per iterations is seen as: This algorithm is highly susceptible to the pa-

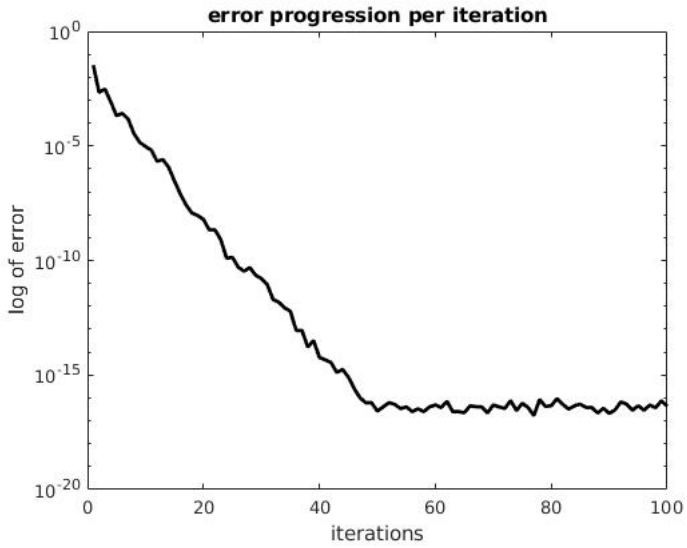

Figure 4.1: l_2 norm error for controller per iteration.

rameters used here especially the excitation constant used with the input. The following result emphasises that fact,higher the excitation lower the number of iterations required:

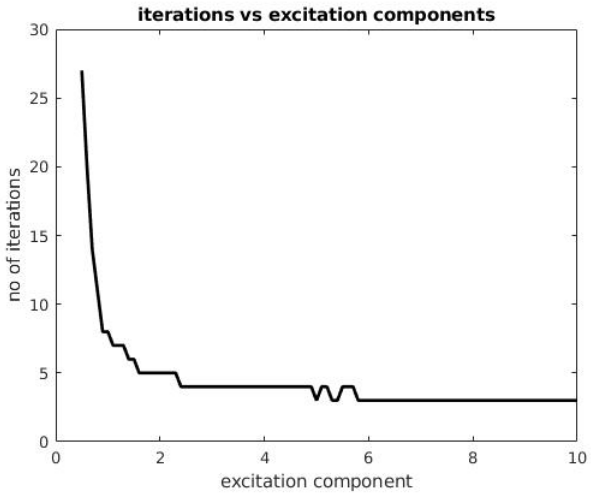

Figure 4.2: Iterations for convergence based as a function of magnitude of excitation component

Looking at the quasi-newton update step version of the algorithm where we slowly update the controller:

The iterative update step we have can be computed as:

$$K = K - \frac{1}{\max eig(2H_{22})}(H_{22}K - H_{21})$$

This is a slower update step as we are doing a step by step trek towards the optimal controller. The plot shows error progression per iteration for the same exploration component as the previous case.

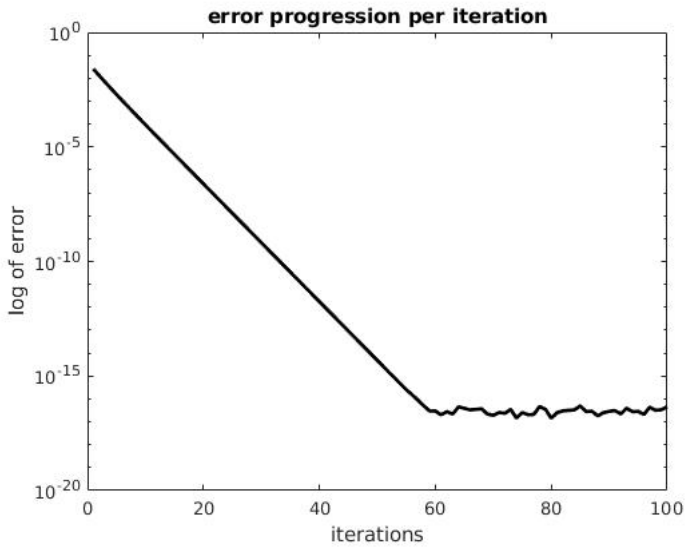

Figure 4.3: l_2 norm error for controller per iteration for quasi-newton update step

As we can see it takes more iterations to update but the step is a smoother approach. Also, since this is a slow update it allows us to not be too aggressive in the exploration step and we can reduce the magnitude of exploration component. This property helps us in the UAV application where we have tough input constraints and cannot afford exploring inputs too divergent from nominal inputs.

Chapter 5

EXPERIMENTAL SETUP

The objective we want to build towards is control of a UAV using model free setups. Imagine a drone control personnel operating a UAV with a remote control. The person has control of roll, pitch and thrust of the UAV to navigate in a three dimensional environment. The system dynamics need not be known to the controller if state feedback is available. Similarly with a manually tunned PID controller system dynamics need not be known if we have feedback.These are the types of model free control approaches we usually see applied in the field. We try and apply the linear LQR based model free techniques in a similar way. First the system is converted to a linear discrete time point mass model where we can control its position states using attitude and thrust adjustment. Then we generate the model free controllers for this setup. We will look at the case where we have a sub-optimal initial controller and noisy state measurement. A problem we look to solve here is generating an optimal controller for a system with altered or changed dynamics without any full knowledge. An example of this is control of a UAV that experiences change in mass as it picks up or drops an object. Another situation could be sudden change in battery level that could lead to variable thrust from the motors. Both of the above situation could put extra strain on the motors. The original LQR controller would try to overcompensate with extra thrust and energy costs would be too high, thus leading to sub-optimal performance if existing controller is maintained.

Our quadrotor system in this setting is a highly non-linear system. Taking a look back at our system definitions we see we have a 12 dimensional non linear system with 4 dimensional action space. As mentioned earlier our system is attitude is PID stabilised. Thus we use the following system:

$$\dot{x} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} q + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & \frac{1}{m} \end{bmatrix} u + w_k \tag{5.1}$$

We assume the system to be discretized version of the above system with Gaussian measurement noise w_k

$$x_{t+1} = A_d x_t + B_d U_t + w_k \tag{5.2}$$

The LQR cost at each time instant is given by:

$$C_t = \sum_{i=0}^{t} x_i^T Q x_i + u_i^T R u_i$$

The problem here is a way-point tracking from point A to B. In LQR reference tracking problem we usually set the error of desired to current state as LQR cost but since we will pick our final position as the origin we can pick the state x instead.Since the system needs to Hurwitz stable and initial cost needs to be finite we take a sub-optimal controller to start with that has a slow performance. The optimal control LQR gain is given by:

$$K = 10^{-3} \begin{bmatrix} 0 & -1.559 & 0 & 0 & -2.038 & 0 \\ 1.559 & 0 & 0 & 2.038 & 0 & 0 \\ 0 & 0 & 0 & 973.365 & 0 & 512.012 \end{bmatrix} \tag{5.3}$$

For the gradient descent problem we need to set up the simulator for exploration. The assumptions for the algorithms included that the initial state is part of a distribution ie $x_0 \sim D$ and initial cost $C(K_0)$ is finite. To satisfy first condition we pick a trust region that does not violate hardware constraints so pick a uniform distribution of states with quadrotor at rest at any position above the ground plane within our work-space. The iterations need to run roll outs until we see a finite cost,

thus roll out length is selected to be long enough to let the dynamics die out. Also number of roll outs per iteration are selected to be enough to gain a stable cost any set of randomly selected points for that controller. Lastly as mentioned above the initial starting point is selected to be just within the stable boundary, thus guaranteeing convergence. A modification we use for gradient update is using the knowledge of structure of the controller. Since we are using a control scheme for a point mass in space that is controlled through attitude and thrust hold we know how the structure of the linear system will look(5.1). This sparse structure also gives rise to the controller in eq[5.3]. To keep with the model free setting we can still use this knowledge of the structure and reduce the learning time by only updating the non-zero parameters for the controller and also exploiting the symmetry. Thus we only need to update four parameters instead of 18 shown in ed[5.3]. Also we can normalise the parameter to keep a uniform learning rate.

The Q-learning algorithm working on the same linear system setup and LQR cost update the controller based on recently collected data-points and is a faster update than the iterative update we saw earlier. We use the same sub-optimal initial controller for this algorithm to satisfy assumptions of the algorithm. The tuning parameters here are the size of data-sets and more importantly the excitation components used for action space. For the same noise characteristics we see the results for Q-learning.

Here for both the cases we will let the controller run until we achieve a satisfactory performance similar in both cases. We cannot ascertain in an actual scenario where our actual LQR controller lies so we would have to rely on empirical data to know how long to run these algorithms

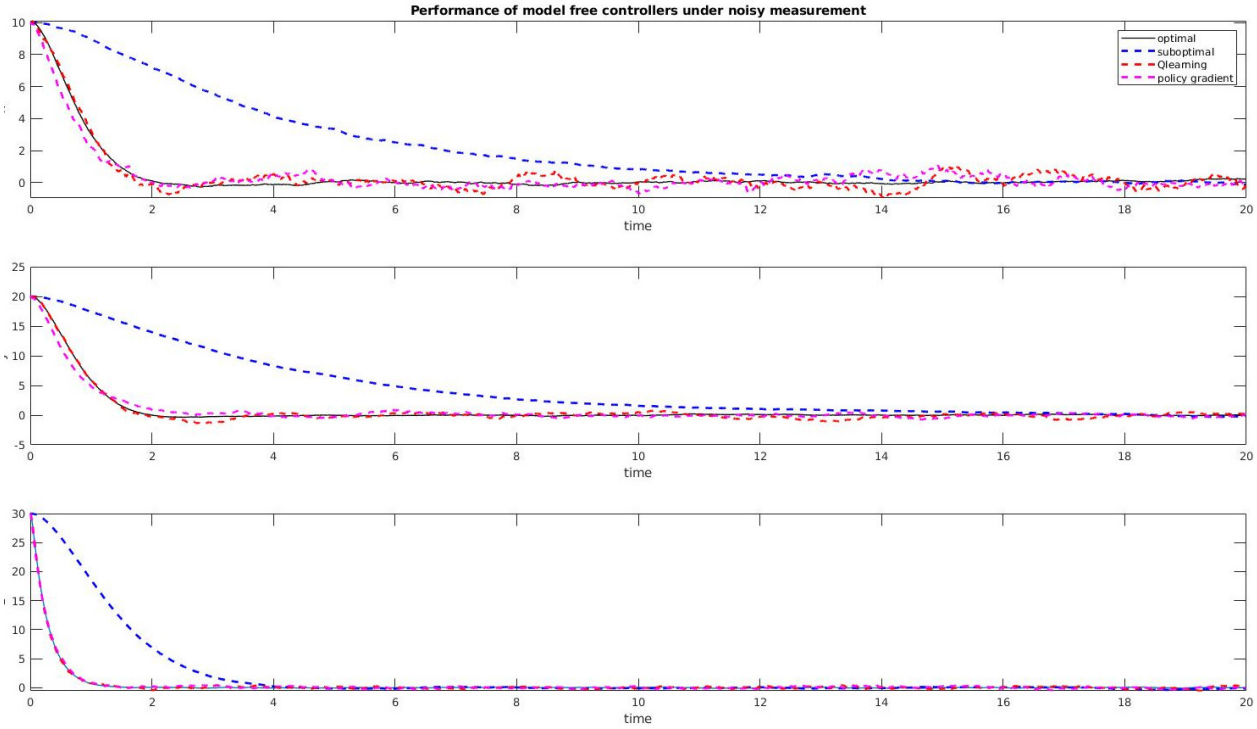

Figure 5.1: Positions states stabilization for sub-optimal and model free controllers

As seen in figure 5.1 The blue dashed line represents the initial controller. We can see slower convergence in this case for all three states (x, y, z) as shown in the figure. The Q-learning solution and policy gradient solution can be seen in red and pink dashed lines respectively. They show much faster convergence even with noisy measurements as shown. The convergence iterations for satisfactory results vary for both these algorithms though. Q-learning can be seen to converge within about 50 iterations of around 200 data-points whereas policy gradient converges in around 150 iterations of about 2500 data point each. They do converge to about 10^{-2} norm error region of the LQR controller. If left to run for more iterations we can be certain of even better results.

The trajectory simulation for both these algorithm shows the real difference we make by upgrading from the sub-optimal controller. We have a LQR trajectory traversing from a point $(100cm, 200cm, 300cm)$ to the origin. In fig 5.2 we show the sub-optimal trajectory in light-gray and we take snapshots at half a second intervals. The Q-learning result are shown in red and policy

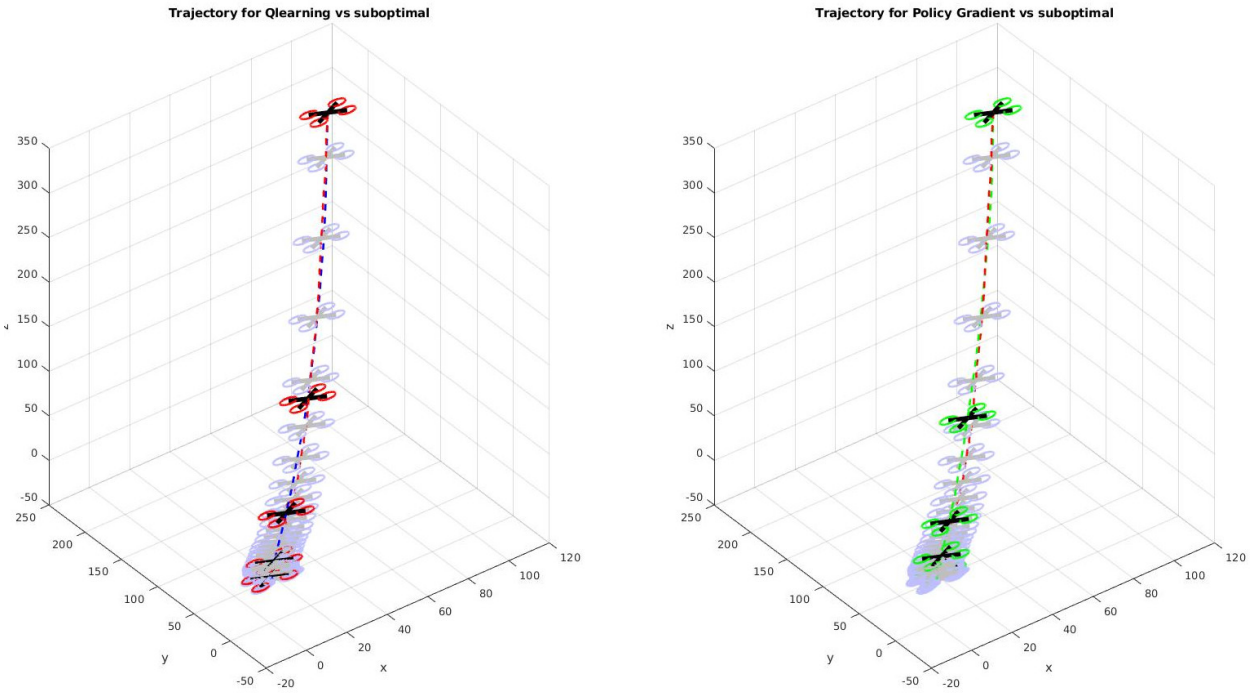

Figure 5.2: Trajectory settling time comparisons for the algorithms. Left: sub-optimal vs Q-learning. Right: sub-optimal vs Policy gradient

gradient in green. We can see for both cases that snapshots for model free algorithms are farther apart meaning the quadrotor travels faster in this instance. Thus starting from same position we can see the model free controllers travelling greater distance than sub-optimal controller while maintaining flight within system constraints. For both the algorithms there is no discernible difference in trajectory paths.

Next we will see the problem we discussed earlier referred to as altered system dynamics for the quadrotor. Since we have established that we have the stability and linearity assumptions for the convergence they must hold true for this problem as well.

For the defined LQR cost we have K as the optimal controller for discrete time linear systems A_d and B_d. Then we can guarantee convergence for all displaced systems $\mathbf{A} = \{A'_d, B'_d\}$ such that $A'_d - B'_d K$ is Hurwitz stable. Thus given that the displaced system can still be stabilised by our previous controller we can guarantee that K as our starting point would converge to LQR solution.

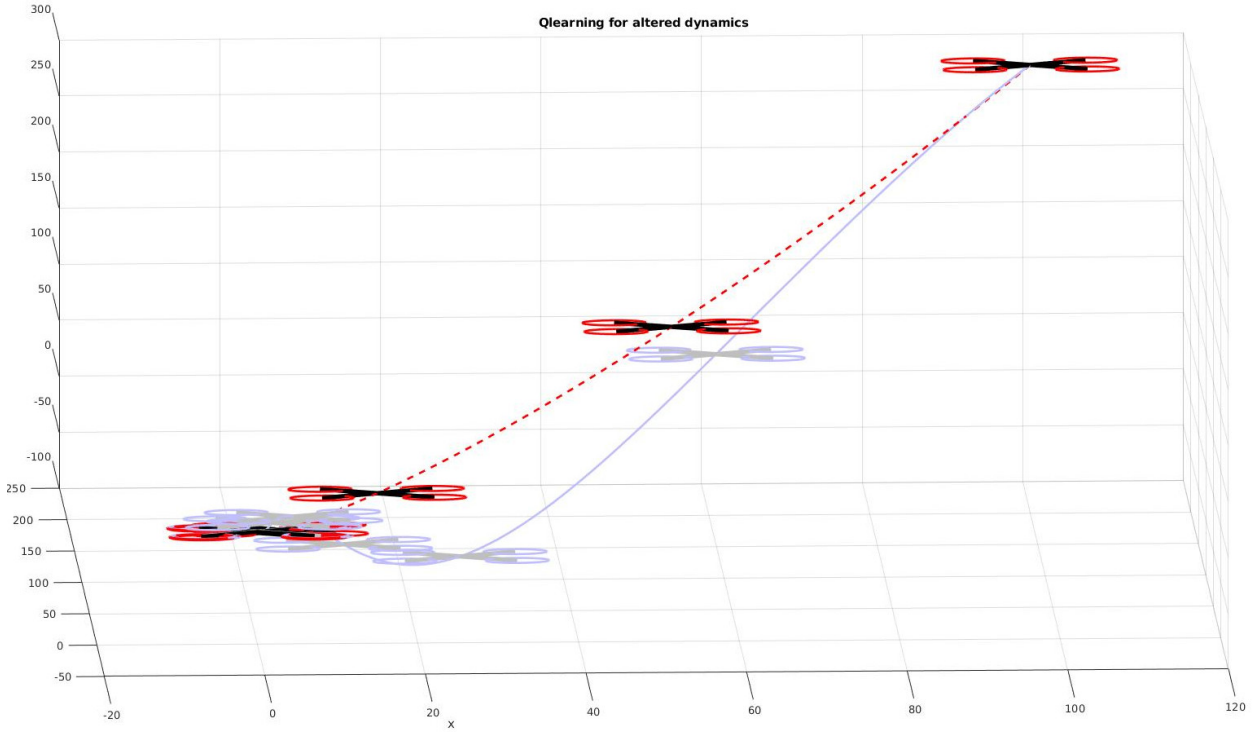

Figure 5.3: Trajectory comparison for original controller(light-gray) and Q-learning controller(red) with altered dynamics

Now for the premise that we have loss of thrust due to a battery drainage and we have addition of mass to the UAV due to object manipulation our LQR controller is performing poorly so thus we can use both these algorithms to improve performance. The results for this case seem to be promising in terms of ease of Q-learning controllers to handle the shift in dynamics. From the figure 5.3 we can see the difference in trajectory for original controller on new system vs Q-learning controller on new system. We can see the original system due to lack of thrust control though being stabilizing violates the planar constraints of dropping below the ground plane. This in a practical scenario would be equivalent to a crash. The Q-learning controller on the contrary can be seen to converge to a optimal controller within 10 iterations without any measurement noise and prevents a crash.

Chapter 6

CONCLUSIONS

Through the survey of current practices in controls and machine learning we have seen a proclivity of robotics community in recent times to use data based methods to solve complex problems. In aeronautics industry though the contemporary deep learning algorithms we see everywhere are hard to spot. Due to higher requirements for safety deep RL methods are rarely used for direct control in aviation of self driving industry. Thus we tried to explore some algorithms that can give us a guarantee of convergence thus building a pathway to use of Deep RL in flight.

We modelled our quadrotor system to fit to the constraints put on the system through the assumptions of linearity and stability put forth by the algorithms. The algorithms by using LQR as the base can guarantee stability and convergence. Another plus point of using this setup is that it can actually be transported to a UAV quiet easily as the algorithms are low-cost in terms of memory exhausted. Since the output of this algorithm from the simulation plane is the controller itself, we can directly swap the solution with on-board controller without changing the embedded system as the UAV already runs on LQR type controllers. This is a huge benefit as compared to deep RL methods like DDPG and TRPO, where the controller is a neural network thus it needs to be first loaded on to high frequency processor with greater computing power, thereby defining control architecture constraints on the UAV itself. Therefore for a setup like our where we have low frequency ICs and have control through a ground station it is much easier to use a algorithm that can be run on the existing platform itself. Another benefit of using the LQR based algorithms is that we can avoid having to designing a discretized reward penalty system where we have to account for all types of scenarios and embed them into the reward function. The caveat in terms of using these algorithms remains in the fact that like deep RL, we have to rely on a simulator for the learning phase.

The thesis presented two model free approaches for UAV control namely Q-learning and Policy gradient method. While both were linear discrete time LQR based methods they differed in the approach as Q-learning is a value function based approach where as gradient method is a policy search. We use the same systems for both approaches and with noise free setting we can benchmark some of their aspects. As mentioned in the previous section, comparing a single update iteration of both algorithms we see that Q-learning uses 10^2 data-points whereas policy gradient update uses around 10^4 order of data-points for a single iteration. The difference arises due to the fact that for each iteration of policy iteration we need explore a batch of controllers around the current one and cycle them through a distribution of initial states. Thus if only a fast convergence is desired one should consider Q-learning. Another aspect for comparison is the excitation component involved in exploring the environment. For policy search we can explore by selecting a random batch of controllers and no extra excitation is needed in the input whereas for Q-learning we see the excitation component is added to the input itself. The magnitude of this excitation in Q-learning algorithms is fairly higher than the acceptable constraint for the original input itself. Therefore for a scenario where the input constraints cannot be violated in the simulator we should consider using the model free approach. The slower iterative update method we saw in Q-learning where we perform a quasi-newton update on the controller based on the Q-function we can reach a middle ground. Here we have same number of data-points as with Q-learning and with a slower update we can afford to reduce the excitation component. The algorithms through the experiments have shown a capability to handle noisy measurements and converge to LQR solution. Some of contemporary deep RL methods avoid addition of noise to data sets during training phase., thus these methods are robust in that aspect. We also saw a setup where we recovered to an optimal solution when the system dynamics had been altered and current controller was no longer optimal.

From the results discussed above the setup seems promising but translating the training phase on-board seems to be next challenging step. For the UAV to train itself mid flight amid an external disturbance or system change would be a appropriate problem to solve stepping forward. Some work is being done in the lab on selecting safe exploration strategies for Q-learning algorithms. So

if we can derive a policy or input sequence for Q-learning that does not violate in flight constraints we can converge to a LQR solution within a few iterations. Applications can also be extended to exploring solution when we do not have a initial stabilising controller for the given system as we may not have such a scenario where system could be stabilised under disturbance. Some work is also done on this problem of finding a initial stabiliser for Q-learning for systems with specific structure. Thus in future, we plan to build towards a system that could let us optimise controls for a UAV on the fly.

BIBLIOGRAPHY

[1] Adel Akbarimajd. Reinforcement learning adaptive pid controller for an under-actuated robot arm.

[2] Hriday Bavle Paloma de la Puente Pascual Campoy Alejandro Rodriguez-Ramos, Carlos Sampedro. A deep reinforcement learning strategy for uav autonomous landing on a moving platform.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017.

[4] Baoxia Tian, Hongye Su, and Jian Chu. An optimal self-tuning pid controller considering parameter estimation uncertainty. In *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393)*, volume 5, pages 3107–3111 vol.5, 2000.

[5] S. Bouabdallah and R. Siegwart. Full control of a quadrotor. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 153–158, 2007.

[6] Steven J. Bradtke. Reinforcement learning applied to linear quadratic regulation.

[7] Jingjing Bu and Mehran Mesbahi. Global convergence of policy gradient algorithms for indefinite least squares stationary optimal control, 2020.

[8] Maryam Fazel, Rong Ge, Sham M. Kakade, and Mehran Mesbahi. Global convergence of policy gradient methods for linearized control problems. *CoRR*, abs/1801.05039, 2018.

[9] Claude-Nicolas Fiechter. Pac adaptive control of linear systems.

[10] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017.

[11] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *CoRR*, abs/1707.05110, 2017.

[12] A. W. Moore L. P. Kaelbling, M. L. Littman. Reinforcement learning: A survey.

[13] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[14] Domagoj Tolić Jens Kober Ivana Palunko Lucian Buşoniua, Tim de Bruin. Reinforcement learning for control: Performance, stability, and deep approximators.

[15] C McDonald. Machine learning: a survey of current techniques. *Artif Intell Rev 3, 243–280*, 1989.

[16] Kalegele K. Mduma, N. and D. Machuve. A survey of machine learning approaches and techniques for student dropout prediction. *Data Science Journal, 18(1), p.14.*, 2009.

[17] Hanssen K.G. Moe S., Rustad A.M. Machine learning in control systems: An overview of the state of the art. *In: Bramer M., Petridis M. (eds) Artificial Intelligence XXXV. SGAI 2018, Lecture Notes in Computer Science, vol 11311. Springer, Cham*, 2018.

[18] Chioniso Dube Piwai N.Chikasha. Adaptive model predictive control of a quadrotor.

[19] Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz. Dynamic mode decomposition with control, 2014.

[20] Ragia I. Badr Rania A. Fahmy and Farouk A. Rahman. Adaptive pid controller using rls for siso stable and unstable systems.

[21] Migdat HODZIC Sevkuthan KURAK. Control and estimation of a quadcopter dynamical model.

[22] D. Sufiyan, L. T. S. Win, S. K. H. Win, G. S. Soh, and S. Foong. A reinforcement learning approach for control of a nature-inspired aerial vehicle. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6030–6036, 2019.

[23] Expert system team. What is machine learning? a definition.

Appendix A

SYSTEM

A.1 Linearization of dynamics

Rearranging the state equations to frame like the state space form showing x as a state with $x = \begin{bmatrix} q & \dot{q} & p & \dot{p} \end{bmatrix}^T$ where q are the attitude states and p are positional states

$$\frac{d}{dt}\begin{bmatrix} \theta \\ \phi \\ \psi \\ \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \\ x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \\ \dot{\phi}\dot{\psi}\frac{(I_y - I_z)}{I_z} + d\frac{u_2}{I_x} \\ \dot{\theta}\dot{\psi}\frac{(I_z - I_x)}{I_z} + d\frac{u_3}{I_y} \\ \dot{\theta}\dot{\phi}\frac{(I_x - I_y)}{I_z} + a\frac{u_4}{I_z} \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \frac{u_1}{m}(cos(\theta)sin(\phi)cos(\psi) + sin(\theta)sin(\psi)) \\ \frac{u_1}{m}(cos(\theta)sin(\phi)sin(\psi) - sin(\theta)cos(\psi)) \\ -g + \frac{u_1}{m}(cos(\phi)cos(\theta)) \end{bmatrix} \tag{A.1}$$

We linearize around an equilibrium for analysis and controller design. This is helpful in a practical scenario as most of the prevalent control methods for quadrotor separate its attitude and position control aspects. This simplifies the quadrotor to a point mass in space with three degrees of freedom.

Linearizing the dynamics around the equilibrium state where $\frac{dx}{dt} = 0$ gives us the conditions

around hover for the quadrotor.

i.e.

$$\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix} = 0 \tag{A.2}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = 0 \tag{A.3}$$

also

$$\begin{bmatrix} \dot{\phi}\dot{\psi}\frac{(I_y - I_z)}{I_z} + d\frac{u_2}{I_x} \\ \dot{\theta}\dot{\psi}\frac{(I_z - I_x)}{I_z} + d\frac{u_3}{I_y} \\ \dot{\theta}\dot{\phi}\frac{(I_x - I_y)}{I_z} + a\frac{u_4}{I_z} \end{bmatrix} = 0 \tag{A.4}$$

$$\begin{bmatrix} \frac{u_1}{m}(cos(\theta)sin(\phi)cos(\psi) + sin(\theta)sin(\psi)) \\ \frac{u_1}{m}(cos(\theta)sin(\phi)sin(\psi) - sin(\theta)cos(\psi)) \\ -g + \frac{u_1}{m}(cos(\phi)cos(\theta)) \end{bmatrix} = 0 \tag{A.5}$$

So we can derive

$$\Rightarrow u_2 = 0, u_3 = 0, u_4 = 0$$

and using small angle approximations we get

$$u_1 = mg, \theta = 0, \phi = 0$$

. lastly the thrust from each motor

$$\Rightarrow m_1 = m_2 = m_3 = m_4$$

The linearised model thus obtained is

$$\frac{d}{dt}\left[X\right] = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} X + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & d/I_x & 0 & 0 \\ 0 & 0 & d/I_y & 0 \\ 0 & 0 & 0 & a/I_z \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 \end{bmatrix} U \quad \text{(A.6)}$$

alternatively we can also use motor thrusts as inputs and use the following B matrix:

$$\frac{d}{dt}\left[X\right] = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} X + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ d/I_x & -d/I_x & d/I_x & -d/I_x \\ -d/I_y & -d/I_y & d/I_y & d/I_y \\ -a/I_z & a/I_z & a/I_z & -a/I_z \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/m & 1/m & 1/m & 1/m \end{bmatrix} U \quad \text{(A.7)}$$

Appendix B

HARDWARE SETUP

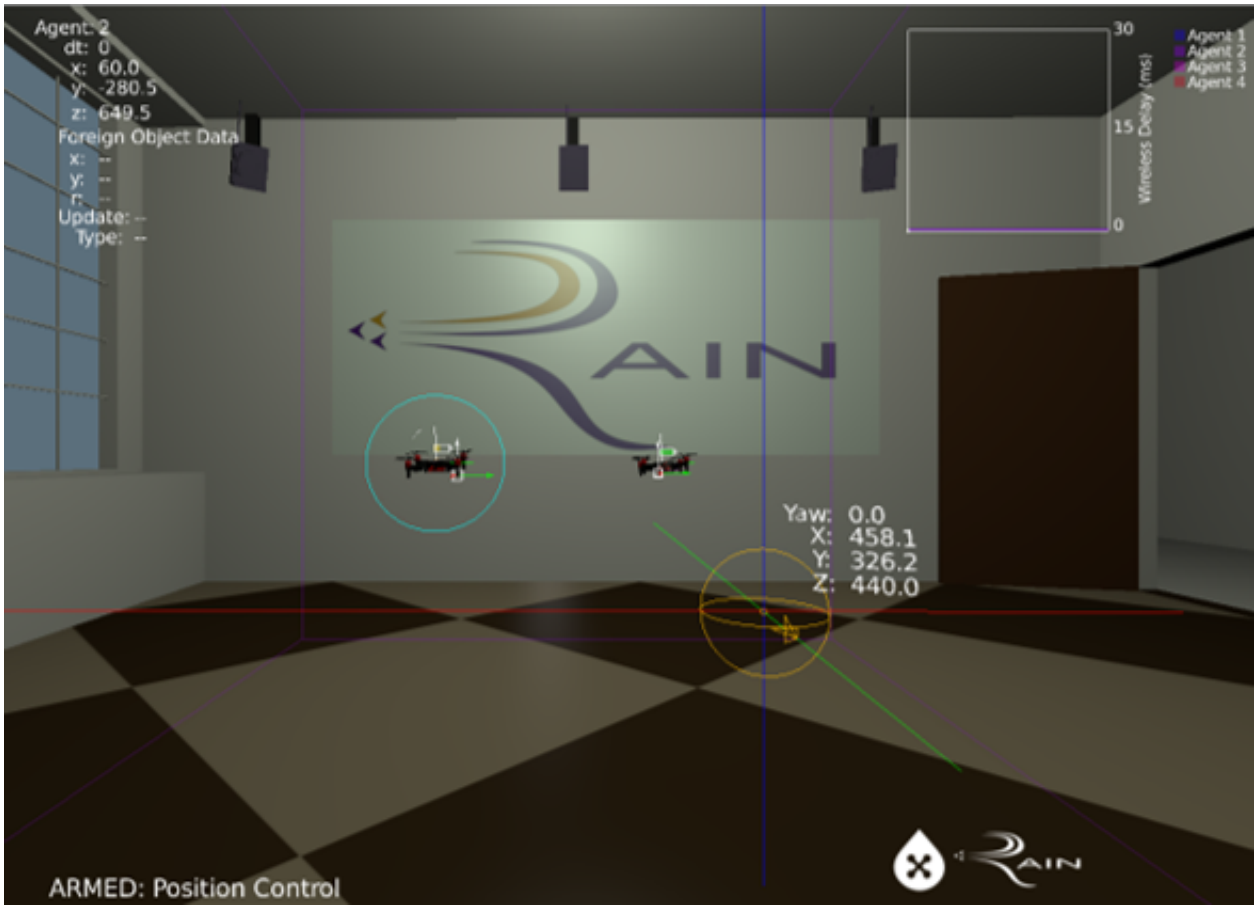

Figure B.1: Quadrotor Blender Simulator

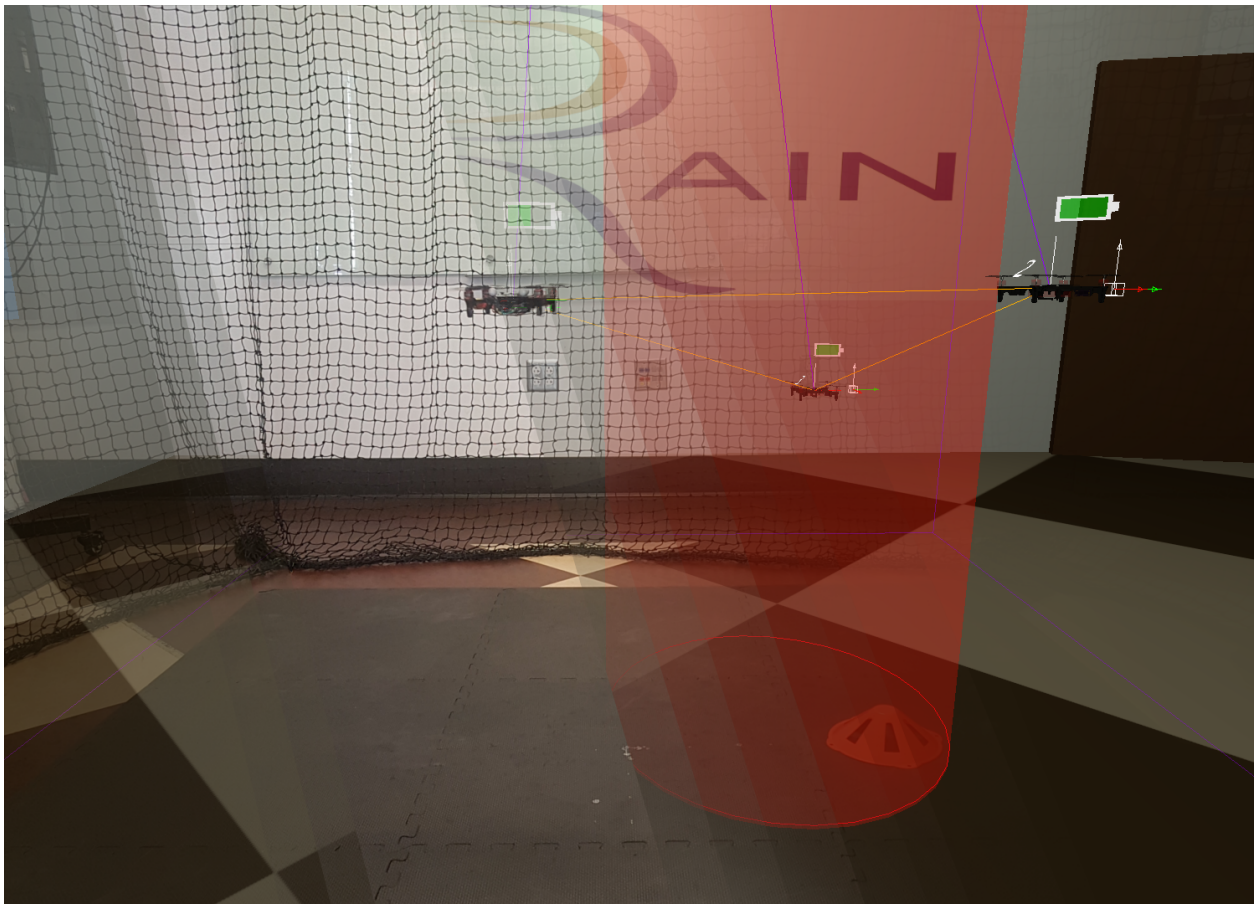

Figure B.2: Obstacle Avoidance in simulator and actual workspace

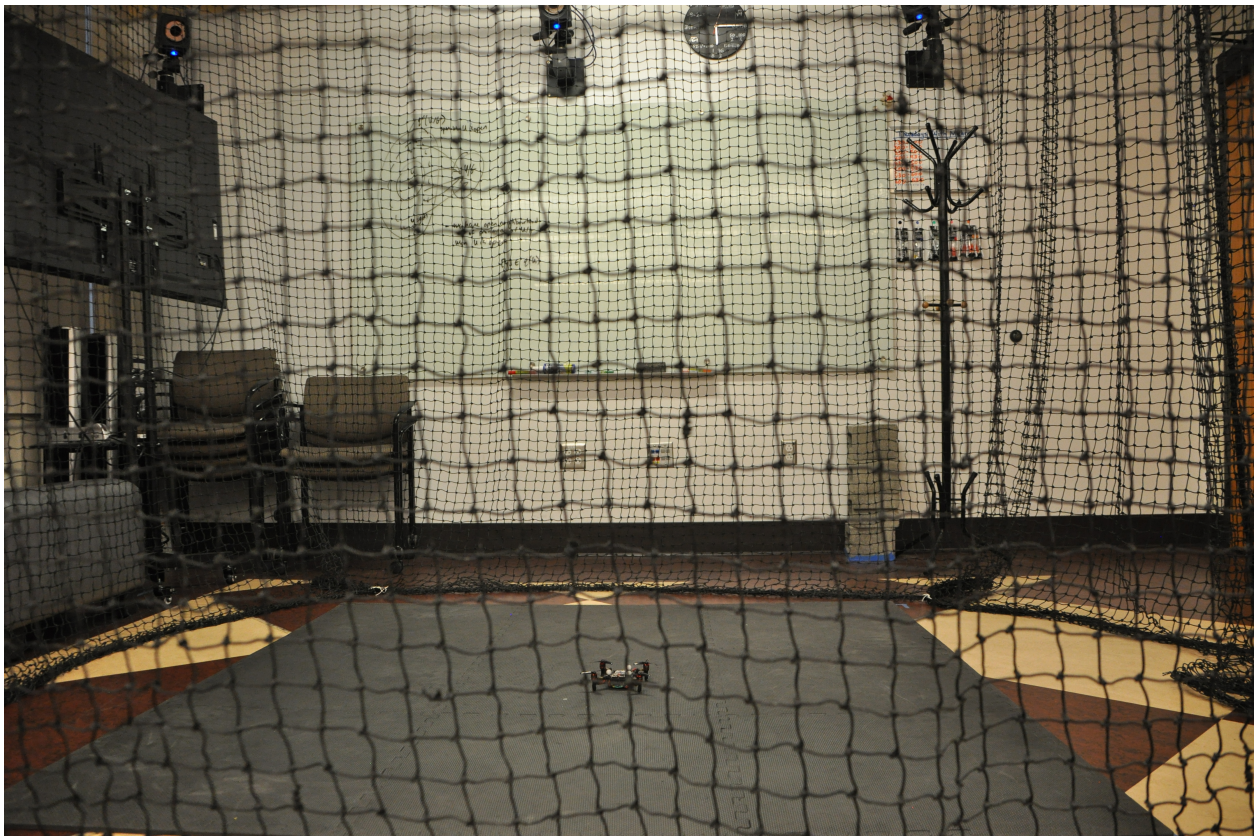

Figure B.3: Lab workspace

Figure B.4: Detailed Quadrotor view

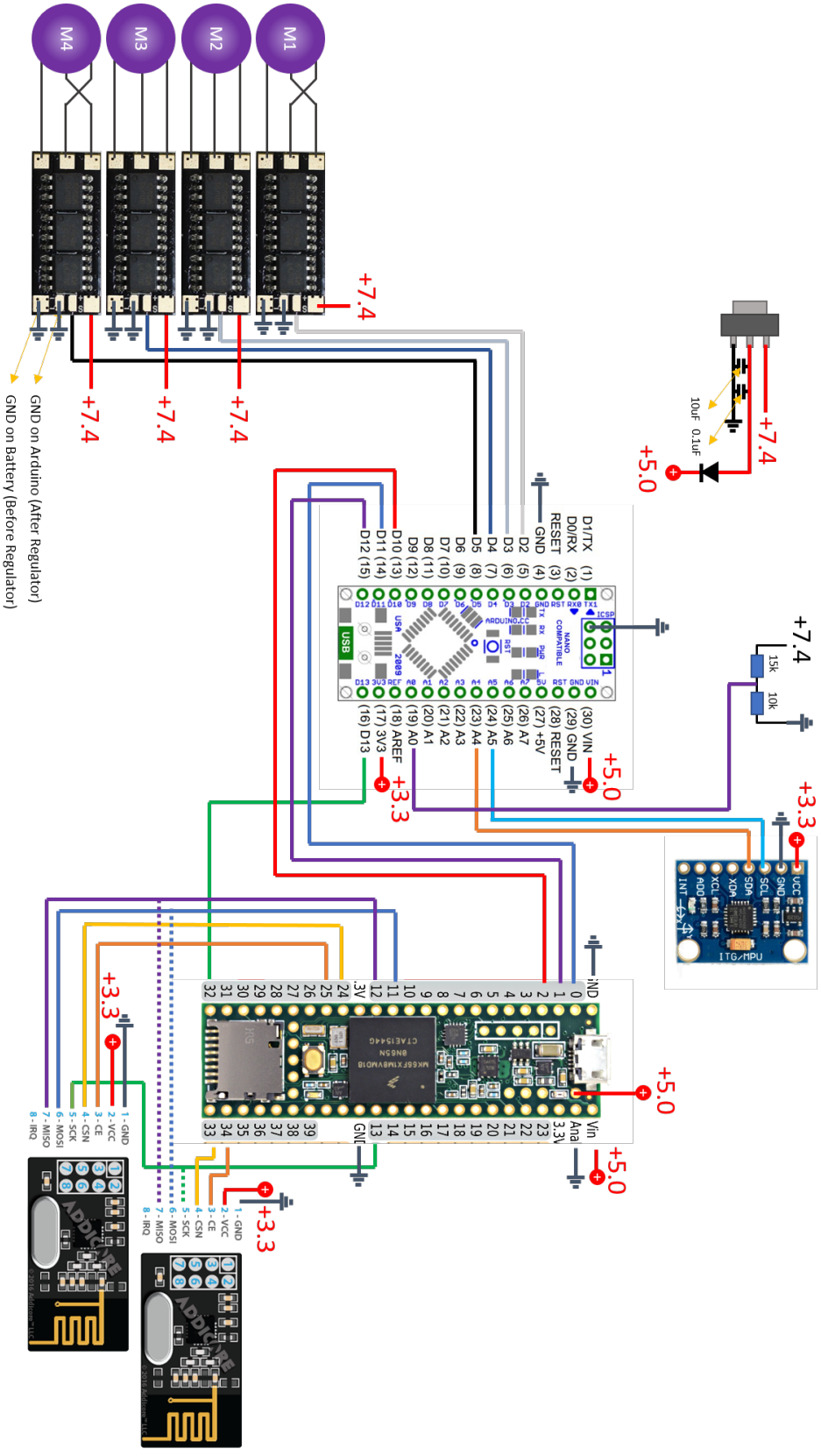

Figure B.5: Control architucture