Faculty of Aerospace Engineering

Intelligent Controller Selection for Aggressive Quadrotor Manoeuvring

A reinforcement learning approach

D. Molenkamp

April 21, 2016

TU Delft Delft University of Technology

Challenge the future

Intelligent Controller Selection for Aggressive Quadrotor Manoeuvring

A reinforcement learning approach

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

D. Molenkamp

April 21, 2016

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Intelligent Controller Selection for Aggressive Quadrotor Manoeuvring"** by **D. Molenkamp** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: April 21, 2016

Readers:

Dr. Q. P. Chu

Dr. ir. E. van Kampen

Dr. ir. C. C. de Visser

Ir. R. Noomen

Preface

This thesis is the final product of my research performed at the Control and Simulation Division of Aerospace Engineering at Delft University of Technology, and is in partial fulfilment of the Master of Science in Aerospace Engineering degree.

The subject of this thesis is intelligent controller selection for aggressive quadrotor manoeuvring and the main contribution is the paper that can be found in Part I. This paper is a stand-alone document that contains the most important findings of my graduation project. A big part of research into Fault Detection and Isolation (FDI) is eventually not included in this thesis report, since the research direction had shifted away from this field after the literature study.

I would like to express my gratitude to my daily supervisor Erik-Jan van Kampen for his continued support and feedback throughout my research. My thanks also go to my fellow students in the graduation room for their advice, discussions during group lunch and all the coffee breaks. Finally I would like to thank my family and friends for giving me the motivation that kept me going.

Djim Molenkamp
April 21, 2016
Delft

Contents

List of Figures

Chapter 1

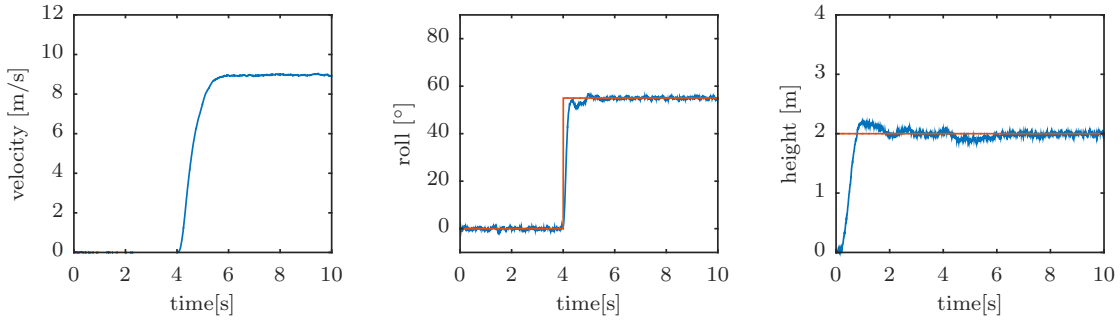
Introduction

Quadrotor helicopters are becoming increasingly popular as a rotorcraft concept for Unmanned Aerial Vehicle (UAV) platforms. They use two pairs of counter-rotating fixed-pitch rotors located at the four corners of the rotorcraft. The utilization of these machines can already be seen in a wide range of applications, such as search and rescue, cinematography, surveillance and as sensor platform (Hoffmann, Waslander, & Tomlin, 2006). According to a reliability study by the US Office of the Secretary of Defense (Marzat, 2012), about 80 percent of flight accidents concerning UAVs are due to faults affecting either propulsion, flight control surfaces or sensors. To allow UAVs to continue their mission, it is necessary to maintain performance in different regions of the flight regime under varying conditions. Classical quadrotor flight controllers can not handle with aggressive manoeuvres sufficiently and experience actuator saturation leading to unwanted behaviour and potential crashes. Adding more performance in more extreme areas of the flight envelope will further increase the possible applications of quadrotors

After an extensive investigation into state-of-the-art research in Fault Detection and Isolation (FDI), Fault Tolerant Control (FTC) or Reconfigurable Control Techniques and Reinforcement Learning (RL), it was found most interesting to implement reinforcement learning to complement a multiple modal flight controller in order to achieve a higher performance and simplify controller design. Tuning a multiple modal flight controller manually had been proven difficult in the past, the switching point location had been chosen arbitrarily. By using reinforcement learning to tune controller selection and find this switching point in an intelligent manner the performance of the quadrotor is likely to be increased.

Using a set of control strategies designed for specific situations quadrotors are able to learn when to switch between these strategies to maintain performance. The performance can be specified in speed, reference tracking or differently depending on the mission characteristics. The contribution of this thesis report is an intelligent controller selection using reinforcement learning. Aggressive horizontal manoeuvres are used to prove this concept and are characterised by fast and large changes in pitch and roll angles. When a conventional controller strategy is tasked to perform an aggressive horizontal manoeuvre the requested velocity translates into a very high commanded tilt angle. If this commanded tilt angle is higher than a

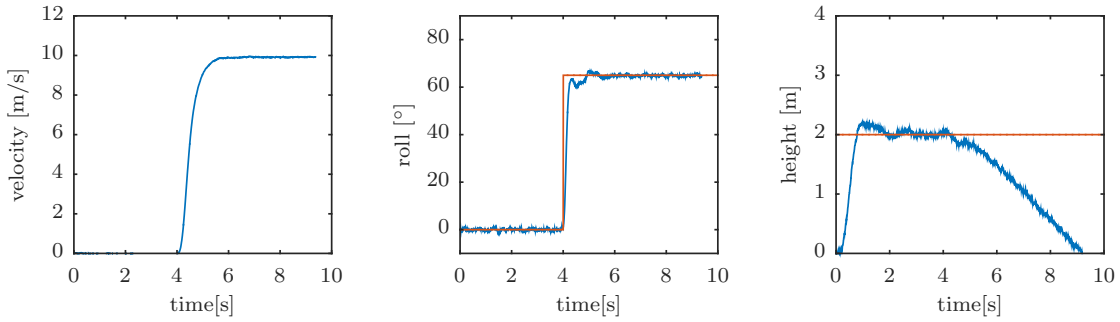
certain limit, the quadrotor will lose altitude due to actuator saturation which is unwanted. Hence performance in aggressive horizontal manoeuvres is insufficient for the conventional control strategy and will result in altitude loss and potentially a crash. In Figure 1-1 a roll command of 55 degrees, which is under the limit, is given:



(a) Velocity **(b)** Roll ϕ **(c)** Height

Figure 1-1: Roll command under the limit

As can be seen the quadrotor is able to maintain the commanded roll angle and accelerate quickly to a horizontal velocity of around 9 m/s. At $t = 4$ s the command is given and a slight decrease in altitude is shown, this is however quickly recovered and the quadrotor is able to maintain its height. However if the commanded tilt angle is too high this will result in altitude loss as is this case in Figure 1-2:



(a) Velocity **(b)** Roll ϕ **(c)** Height

Figure 1-2: Roll command above the limit

A horizontal speed of 10 m/s is not sustainable, since it requires a roll angle that is over the limit. As can be seen the quadrotor quickly loses altitude until it crashes at $t = 9$ s.

In order to guarantee performance and safety an adaptive controller is implemented to intelligently switch between a conventional controller for ordinary situations and a specialized controller for extreme angles.

1-1 Research question, aims and objectives

Reinforcement learning is a self-learning control method that is inspired by behaviourist psychology, it learns by interacting with the environment and receives a reward depending on the success of the action that has been taken. A challenge that appears with reinforcement learning and not in other kinds of learning is the balance between exploration and exploitation. To maximise reward the agent must choose actions that it knows will return a high reward. However to discover these actions the agent has to try actions it has not selected before. The agent should exploit current knowledge to receive reward, but explore to find better alternatives to the current best.

Using reinforcement learning for controller selection has the benefit that it has the capability to find the most optimal switching moment through learning. Investigating how reinforcement learning can be applied both off-line and on-line and how this impacts quadrotor performance is the goal of this research.

This leads to the following a research question:

How can a multiple modal flight controller architecture for a quadrotor using reinforcement learning keep performance at a high level in varying flight conditions?

The research objective is to determine whether the flight envelope of UAVs can be extended and tuning flight parameters simplified by using a multiple modal approach augmented with reinforcement learning. The concept will be able to choose from different control strategies in varying flight conditions, gradually learning which controller is best to use in each situation. In order to learn which controller has better performance a particular situation has to be visited multiple times, to appropriately know the controller performance in that situation.

To give a definite answer to the research question several sub-questions have been formulated:

- How much off-line learning is necessary for guaranteed on-line stability?
- What external disturbances can be dealt with by using this controller architecture?
- Can on-line implementation correct for simulation model differences in a safe and fast manner?
- How much is the performance increased as opposed to single controller usage?
- Is it possible to visit all states to determine appropriate controller selection in each scenario?

By taking a look at the possible regimes of the flight envelope that can occur in the system an estimate of the amount of controllers can be made. In this thesis two controllers are implemented, one for hover and ordinary manoeuvres and another to keep performance high for aggressive horizontal manoeuvres. The performance of each controller in different regions of the flight envelope will determine when to switch between them. Answering the above sub-questions will give an answer to the main research question.

1-2 Outline of this report

This report is structured as follows: first the scientific paper is presented in Part I and is considered to be a standalone document. Part II contains additional results and a walk-through of the simulation model. Finally Part III presents the literature study that has been done in the preliminary phase of this thesis project and contains a more comprehensive introduction into reinforcement learning theory.

Part I

Paper

Intelligent Controller Selection for Aggressive Quadrotor Manoeuvring

D. Molenkamp*, E. van Kampen†, C.C. de Visser‡ and Q.P. Chu§
Delft University of Technology, Delft, 2600 GB, The Netherlands

A novel intelligent controller selection method for quadrotor attitude and altitude control is presented that maintains performance in different regimes of the flight envelope. Conventional quadrotor controllers can behave insufficiently during aggressive manoeuvring, in extreme angles the quadrotor is unable to maintain height which may result in loss of performance. By implementing several controllers designed specifically for these more extreme manoeuvres it is possible to maintain performance while expanding the flight envelope beyond conventional limitations.

The method proposed uses Q-Learning to learn which controller performs best in different scenarios. The controllers that can be used consist of a low angle Nonlinear Dynamic Inversion (NDI) controller and a specialised high angle NDI controller that tries to prevent actuator saturation by controlling height through the pitch and roll angles.

The algorithm is split into 2 parts. During the off-line training phase the quadrotor learns a baseline policy that can be used on-line to skip the initial exploration phase. The resulting policy is a clear representation of controller selection throughout the flight envelope. The success of on-line implementation is highly dependent on the quality of the simulation model. In the on-line phase the policy is mostly exploited and learning is done at a much lower rate.

Finally it is shown that reinforcement learning is a very effective way to tune complex systems. The complex system dynamics do not need to be known, only important performance parameters need to be correctly formulated in a reward function.

Nomenclature

s	State
a	Action
F	Force [N]
I	Moment of inertia [kgm^2]
L, M, N	Moments along the body x,y or z-axis [Nm]
m	Mass [kg]
p, q, r	Angular rates along the body x,y or z-axis [rad/s]
T	Thrust [N]
Q	State-action pair value
q	Quaternion
R	Rotation matrix
r	Reward
V	Velocity [m/s]
W'	Quaternion rate matrix
x, y, z	Position along the x,y or z-axis [m]

*MSc Student, Control and Simulation Division, Faculty of Aerospace Engineering, d.molenkamp@student.tudelft.nl
†Assistant Professor, Control and Simulation Division, Faculty of Aerospace Engineering; e.vankampen@tudelft.nl, AIAA Member.
‡Assistant Professor, Control and Simulation Division, Faculty of Aerospace Engineering; c.c.devisser@tudelft.nl, AIAA Member.
§Associate Professor, Control and Simulation Division, Faculty of Aerospace Engineering; q.p.chu@tudelft.nl, AIAA Member.

α	Learning rate
γ	Discount factor
τ	Torque [Nm]
ϕ, θ, ψ	Rotation along the x,y or z-axis [rad]
ω	Angular rates [rad/s]

I. Introduction

Quadrotor helicopters are becoming increasingly popular as a rotorcraft concept for Unmanned Aerial Vehicle (UAV) platforms.[1] Quadrotors use two pairs of counter-rotating fixed-pitch rotors located at the four corners of the frame.[2] The utilization of these machines can already be seen in a wide range of applications, such as search and rescue, cinematography, surveillance and as sensor platform.[3] According to a reliability study by the US Office of the Secretary of Defense,[4] about 80 percent of flight accidents concerning UAVs are due to faults affecting either propulsion, flight control or sensors. To allow UAVs to continue their mission, it is necessary to maintain performance in different regions of the flight regime under varying conditions. Classical quadrotor flight controllers can not handle with aggressive manoeuvres sufficiently and experience actuator saturation leading to unwanted behaviour and potential crashes. Adding good performance in more extreme areas of the flight envelope will further increase the possible applications of quadrotors.

Using a set of control strategies designed for specific situations quadrotors are able to learn when to switch between these strategies to maintain performance. The performance can be specified in speed, reference tracking or differently depending on the mission characteristics. The contribution of this paper is an intelligent controller selection using reinforcement learning. Aggressive horizontal manoeuvres are used to prove this concept and are characterised by fast and large changes in pitch and roll angles. When a conventional controller strategy is tasked to perform an aggressive horizontal manoeuvre the requested velocity translates into a very high commanded tilt angle. If this commanded tilt angle is higher than a certain limit, the quadrotor will lose altitude due to actuator saturation which is unwanted. Hence performance in aggressive horizontal manoeuvres is insufficient for the conventional control strategy and will result in altitude loss and potentially a crash. In order to guarantee performance and safety an adaptive controller is implemented to intelligently switch between a conventional controller for ordinary situations and a specialized controller for extreme angles.

Various control strategies will provide distinct control inputs during flight, depending on each controllers strengths and weaknesses, while the performance of each controller will vary throughout the flight envelope. The intelligent controller selection algorithm starts without knowledge of controller performance and has to be trained off-line before it can be used on-line. During simulation the controller selection slowly learns which control inputs in each state amount to the highest performance. This is done by either choosing a control strategy randomly or by choosing the best known strategy up to that point. This policy is known as ϵ-greedy and is applied to Q-Learning, a popular method used in reinforcement learning. The off-line simulation forms a baseline policy matrix which ensures an acceptable starting level of performance on-line.

The policy matrix is also updated on-line, albeit it at a much lower rate. During simulation the focus was on exploration, to determine the performance throughout the flight envelope. This balance is shifted to mostly exploitation during on-line usage. By constantly assessing the performance on-line various deviations from the simulated environment can be tackled and correct for as well as a range of unexpected changes to the system.

This paper continues by presenting the quadrotor dynamics in Section II. The reinforcement learning principles that have been implemented are described in Section III. In Section IV the intelligent controller selection algorithm is explained in detail. Then, in Section V the simulation setup is described and in Section VI the results are presented and discussed. Finally, Section VII gives the conclusion and Section VIII some recommendations for future work.

II. Quadrotor Dynamics

A quadrotor has four engines symmetrically placed around its center of gravity, the location where the main flight control board is situated. The basic working principle is that all control is done by changing the thrust of two sets of counter-rotating propellers. A schematic view of a quadrotor system with body reference frame B and the forces and moments can be found in Figure 1. As can be seen the system is

under-actuated with input moments around each axis, but input forces only in the negative z-direction.

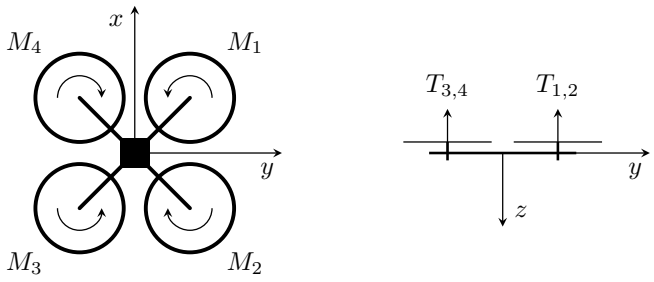


Figure 1: Schematic view of the quadrotor configuration, left a top view and right an intersection through the y-z plane seen from behind.

Another reference frame E is defined as North-East-Down Earth-Fixed with the x_E and y_E-axis pointing North and East and the z_E-axis pointing down. The rotation of the body frame B in the Earth-Fixed frame E is defined using the unit quaternion $\mathbf{q} = [q_0 \;\; q_1 \;\; q_2 \;\; q_3]^T$. Quaternions are used, because of the limitations of using Euler angles, such as singularities.[5] The kinematic relations are found in Equations 1 and 2:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_E = R(\mathbf{q})^T \mathbf{V}_B \tag{1}$$

$$\dot{\mathbf{q}}(\mathbf{q}, \boldsymbol{\omega}_B) = \frac{1}{2} W'(\mathbf{q})^T \boldsymbol{\omega}_B \tag{2}$$

with $\mathbf{V}_B$ the velocity in the body frame and $\boldsymbol{\omega}_B = [p \;\; q \;\; r]^T$ the angular rates in the body frame. The rotation matrix $R(\mathbf{q})$ and quaternion rate matrix $W'(\mathbf{q})$ are defined as:[5]

$$R(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3}$$

$$W'(\mathbf{q}) = \begin{bmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \tag{4}$$

The equations of motion of the quadrotor are derived from Newton's Second Law,[6] and their general form is:

$$\mathbf{F} = m\ddot{\mathbf{x}} \tag{5}$$

$$\mathbf{M} = I\dot{\boldsymbol{\omega}}_B + \boldsymbol{\omega}_B \times I\boldsymbol{\omega}_B \tag{6}$$

with $\mathbf{F}$ the force vector along the body x-, y-, z-axis, m the mass, $\mathbf{M} = [L \;\; M \;\; N]^T$ the moments acting around the body x-, y-, z-axis and I the inertia matrix given by Equation 7:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \tag{7}$$

Now that equations of motion have been given, the force vector $\mathbf{F}_B$ and moment vector $\mathbf{M}_B$ and their components will be discussed. The force vector $\mathbf{F}_B$ consists of 3 contributions, which can be seen in Equation 8:

$$\mathbf{F}_B = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \mathbf{F}_{fus} + \mathbf{F}_{rot} \tag{8}$$

The first contribution is due to gravity acting along the z-axis with m the total mass and g the gravitational acceleration. The main part of the fuselage force $\mathbf{F}_{fus}$ is the drag force D in opposite direction to the airspeed, the fuselage also produces a lift force L perpendicular to the airspeed depending on the frame geometry.[7] Finally the rotor force $\mathbf{F}_{rot}$ acting along the negative z-axis, which is the sum of thrust T_i generated by each rotor. The aerodynamic model is reduced to a static relation between the thrust T, torque τ and rotor rotational velocity ω as suggested in previous work[8] based on the momentum theory:

$$\mathbf{F}_{rot} = \sum_{i=1}^{4} T_i = \sum_{i=1}^{4} C_T \omega_i^2 \tag{9}$$

with C_T a positive constant. The moment vector $\mathbf{M}_B$ consists of three contributions and is found in Equation 10:

$$\mathbf{M}_B = \mathbf{M}_{fus} + \mathbf{M}_{rot} + \mathbf{M}_G \tag{10}$$

The fuselage moment $\mathbf{M}_{fus}$ caused by the frame geometry is only a minor contribution. The moment vector due to rotor forces $\mathbf{M}_{rot}$ is caused by the difference in engine thrust T_i, difference in engine torque τ_i and blade flapping effects. The difference in engine thrust causes moments about the x- and y-axis, while difference in engine torque causes a moment about the z-axis as can be seen in Equation 11:

$$\mathbf{M}_{rot} = \begin{bmatrix} (T_3 + T_4)l - (T_1 + T_2)l \\ (T_1 + T_4)l - (T_2 + T_3)l \\ \tau_1 - \tau_2 + \tau_3 - \tau_4 \end{bmatrix} \tag{11}$$

$$\tau_i = C_\tau T_i \tag{12}$$

with T_i the thrust generated by motor i = 1,2,3,4, l the motor distance to the symmetry-axis, τ_i the generated torque and C_τ a positive constant. The gyroscopic moment $\mathbf{M}_G$ is the sum of the gyroscopic moments occurring in each rotor given by:[8]

$$\mathbf{M}_G = -\sum_{i=1}^{4} (-1)^{i+1} \omega_i I_r \omega_i \tag{13}$$

with ω_i the rotor rotational speed and I_r the rotor moment of inertia. Finally the lateral effects that usually occur in helicopters in horizontal flight are cancelled due to the symmetry of the four rotors.[7]

III. Reinforcement Learning

Reinforcement learning is a type of machine learning that is concerned with how an agent should choose its actions in an environment to maximize its cumulative reward.[9] The agent is not told what to do, but must discover which actions yield the maximum reward by trying them. The actions taken may not only affect the immediate reward but also subsequent rewards. After taking an action the agent finds itself in a new state and the process is repeated. This concept is visualized in Figure 2, with state $s \in \mathcal{S}$ where $\mathcal{S}$ is the set of possible states, action $a \in \mathcal{A}$ where $\mathcal{A}$ is the set of available actions and reward r.

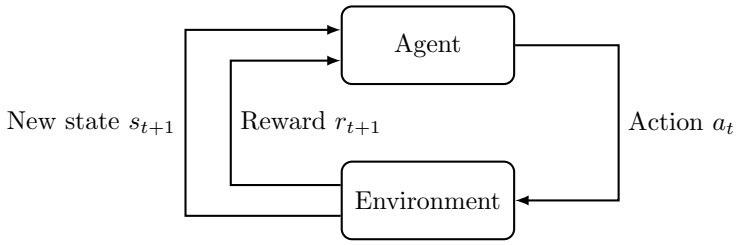


Figure 2: Interaction between agent and the environment in reinforcement learning

One of the most important breakthroughs in the field of reinforcement learning was the development of Q-Learning, an off-policy TD (Temporal Difference) control algorithm.[10] It is found to be a valuable tool

for solving optimal control problems on-line, subject to partial knowledge of the system state and models.[11] Every state-action pair is assigned an estimated value, a Q-value, which is a measure of performance. The main advantage of Q-learning over other methods is that Q-learning directly approximates the optimal action-value function, independent of the policy that is being followed. The policy will still affect which state-action pairs are visited and updated, but as long as all pairs continue to be updated convergence will take place.[12] The basic update rule for a state-action pair can be seen in Equations 14:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\substack{\text{learning}\\ \text{rate}}} \Big[\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q(s_{t+1}, a)}_{\substack{\text{estimate of}\\ \text{optimal future value}}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big] \tag{14}$$

In order for a reinforcement learning algorithm to find the optimal action complete exploration of the state-space is necessary. This is however infeasible in many situations, due to practical reasons. When learning and control are both at stake, the agent must try to find a good balance between both. Exploration of alternatives to a given policy might lead to an improvement in the policy but a risk of performance degradation and exploitation of the current policy will never lead to improvement of current conditions.[13] Reinforcement learning methods usually include a stochastic action selector that will allow for exploration of the state space. A common method is to use a Gibbs or Boltzmann distribution, where the probability of exploring slowly decays and in the end the best action is always chosen.[9] A common policy π used in Q-learning is the so called ϵ-greedy policy. This policy chooses a random action with a probability of ϵ, otherwise taking the action that yields the maximum known reward in that state.

In order for the quadrotor to first train and focus on exploration of the state-space without inflicting any damage, the process of learning the Q-matrix has been split up into 2 parts. An off-line part which focuses on efficient exploration of the state-space to learn a base-line behaviour matrix and an on-line part where this base-line matrix has been implemented on the quadrotor and where the exploration-exploitation balance has shifted more towards exploitation since the control of the quadrotor is at stake.[14] This will ensure that the best known controller is exploited to achieve high performance, while the exploration will continue at a much lower rate to still be able to cope with unforeseen changes in the system or its environment. These changes include degradation of thrust efficiency, quadrotor mass differences or centre of gravity shifts. The balance between exploitation and exploration in the on-line phase needs to be carefully chosen since high performance is desired, however if the exploration is too slow the performance will decrease to unacceptable levels with changes in the system or the environment and it will take the quadrotor too long to learn the new behaviour that optimizes behaviour in the new situation.

In this paper reinforcement learning is used to learn which control strategy to use in different parts of the flight envelope. For any reinforcement learning problem it is important to define the reward function in a correct way, because the algorithm will learn by trial and error and uses the rewards it gets to determine which action is the best in each visited state. If the reward function is poorly constructed this will resonate through to the behaviour of the reinforcement learning algorithm and performance will not be optimal. The next section will elaborate on the intelligent controller selection algorithm and present the different reward functions that have been investigated.

IV. Intelligent Controller Selection Algorithm

Now that the underlying theory has been presented in the previous sections, the contribution of this paper is presented in this section. An intelligent controller selection algorithm that learns which control strategy to use at what time in order to achieve high flight performance even in the most extreme areas of the flight envelope. In order to safely explore the state-space matrix and find a base-line behaviour matrix the implementation of the intelligent control algorithm is divided into 2 parts:

- **Off-line training**: In a simulated environment. The quadrotor is initialized, stabilised to a pre-determined altitude and given a set of random manoeuvres to track, not knowing which control strategy is achieving the highest performance for that given set of manoeuvres. After the Q-matrix has converged and the behaviour matrix does not change significantly between epochs it can be implemented on-line.
- **On-line implementation and adaptation**: The baseline behaviour matrix that has been found to achieve high performance is used on-line. The policy is mostly exploited to achieve high performance,

however some exploration is still done to correct simulation model errors and adapt to new unforeseen changes in the system dynamics.

Subsection A will further elaborate on the off-line training phase and present the intelligent controller selection algorithm and the different reward functions that have been applied. Once the Q-matrix has converged the on-line implementation and adaptation phase begins, which is discussed in subsection B.

A. Off-line training phase

The off-line training phase starts with a quadrotor that has no prior knowledge of actions that yield high performance. The behaviour Q-matrix is initialized as an empty matrix and by visiting each state-action pair repeatedly the best action to take in that state is found. Since the action is selecting which control strategy to use, the resulting effect of choosing a particular controller can only be found after some time has passed and the control actions have taken effect. A visualisation of the intelligent controller selection algorithm can be found in Figure 3:

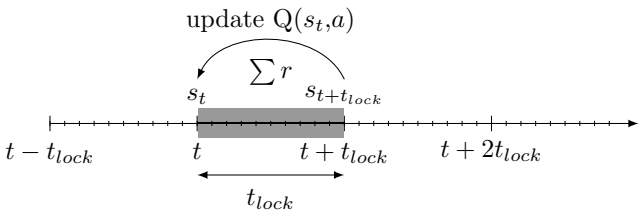


Figure 3: Intelligent Controller Selection Algorithm

The algorithm starts when the quadrotor is in state s_t at time t and one of the available control strategies is chosen. The controller choice is locked for a period of t_{lock} seconds, during which performance is measured according to a reward function. After the lock period has passed the accumulated reward r is awarded to state s_t and a new control strategy is chosen and the process repeats. The pseudo-code is as follows:

Algorithm 1 Intelligent Controller Selection Algorithm

1. Initialize Q-Matrix
2. Repeat (for each simulation run)
   - Initialize s
   - Start counter t
   - Repeat (for each timestep of simulation run)
     - **If** counter $t = t_{lock}$:
       - Collect reward r for timestep and save accumulated reward into Q-matrix for the state where the controller has been locked.
       - Save the current lock state s
       - Choose a random controller with probability ϵ, else observe s and choose best known controller.
       - Reset counter to $t = 0$.
     - **else:**
       - Collect reward r according to reward function chosen.
       - Increment counter t.
3. When the policy does not change significantly between epochs anymore, save Q-Matrix

Between simulation runs the learned Q-matrix is passed on and will slowly get filled throughout the entire state-space, since each simulation run consists of different manoeuvres with varying amplitude and frequency. The controller lock time t_{lock} has direct influence on how relevant the reward is for the state where the controller lock took place. A low value will make the controller performance measured relate to the instantaneous state of the quadrotor, while a larger value is relating to the overall performance in the current area of the flight envelope.

The value for t_{lock} needs to be chosen carefully. To ensure sufficient relevance to the locked state t_{lock} is kept as small as possible, while still allowing for enough time for control actions to take effect. The smallest t_{lock} that gives a clear distinction between controller selection is found by gradually increasing this parameter step by step until a distinct switching point is found.

Reward functions

The reward functions that have been used to learn intelligent controller selection look at maintaining altitude, as can be seen in Equations 15 to 17. For each timestep t the reward r is collected and summed until $t = t_{lock}$ is reached. During aggressive manoeuvring the quadrotor is commanded large pitch and roll angles and actuator saturation can occur due to the altitude control structure. The low angle controller uses average thrust to control altitude, which saturates available thrust at high tilt angles. This is not desirable and therefore altitude is chosen as the most critical parameter:

$$r = \sum_{t=0}^{t_{lock}} - |h_{ref}(t) - h(t)| \tag{15}$$

$$r = \sum_{t=0}^{t_{lock}} - (h_{ref}(t) - h(t))^2 \tag{16}$$

$$r = \sum_{t=0}^{t_{lock}} - |h_{ref}(t) - h(t)| + K_w \cdot sgn(\dot{h}(t)) \cdot (h(t) - h_{ref}(t)) \tag{17}$$

Equation 15 is an integration of the altitude tracking error during the controller lock period. A slight alteration is Equation 16, which uses the squared tracking error such that a bigger deviation from the commanded height is punished more severely. Although these reward functions should be sufficient for learning when to use a particular controller there are some downsides which Equation 17 tries to overcome.

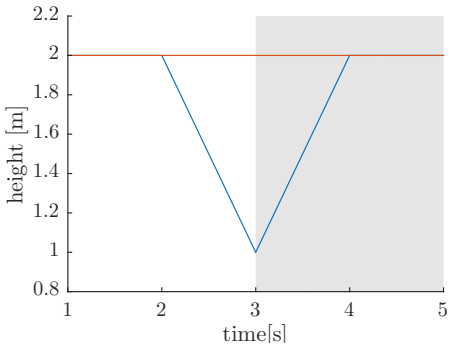


Figure 4: Possible Shortcoming Reward - red: commanded height, blue: height tracking, white background: low angle controller, gray background: high angle controller

If the altitude loss due to the low angle controller in more extreme angles is of the same speed as the recovery by the high angle controller, then using Equations 15 and 16 both controllers will receive a similar reward. This idea is visualised in Figure 4, where the white background indicates the low angle controller performing poorly and the gray background indicates the high angle controller recovering the quadrotor height. The reward function proposed in Equations 17 also takes into account whether the quadrotor is moving towards the reference signal and thus will improve tracking or moving away from it and give reward accordingly. The constant K_w is used to scale the importance of movement towards the reference to absolute distance from the reference.

B. On-line Implementation and Adaptation

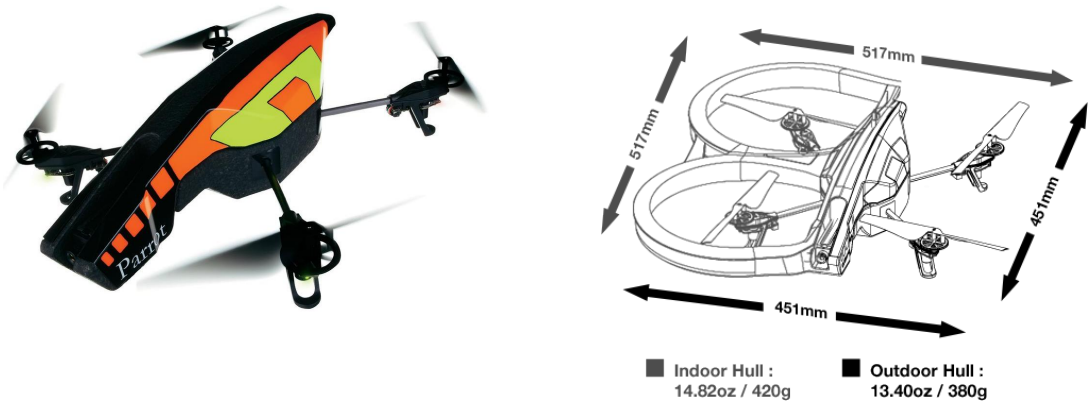
After having visited each state in the state-space enough times the quadrotor will have learned which is the best overall controller for that particular state, regardless of the reference that needs to be tracked. This

policy matrix can now be implemented on the physical quadrotor and provides a base-line behaviour to follow. During every initialisation of the actual quadrotor the Q-matrix will already have been filled with off-line training results. Through this experience replay the problem of early exploration leading to unstable situation is counteracted.

There often is a slight difference between reality and simulation environments, so the quadrotor is still allowed to continue learning and modify the behaviour matrix in order to maximize performance. It is noted that the learning rate during the on-line phase is much lower, since this phase mainly exploits the knowledge obtained in simulation. When actual quadrotor characteristics differ from the simulation model, such as quadrotor weight or motor thrust, the behaviour matrix will slowly shift towards a new optimum to increase performance.

V. Simulation Setup

The new proposed controller selection algorithm is tested on a simulation model of the AR.Drone 2 quadrotor with the outdoor hull, which can be seen in Figure 5. The AR.Drone 2 is equipped with a set of basic sensors that are found in most quadrotor platforms, these sensors are described in Table 1:



(a) Outdoor hull (b) Schematic view

Figure 5: The AR.Drone 2 by Parrot (http://ardrone2.parrot.com/)

Table 1: Onboard sensor specifications AR.Drone 2

Sensor	**Measured variable**	**Precision**
Accelerometer	Specific force in x, y, z directions	± 50 mg
Gyroscope	Angular velocity around x, y, z axes	2000 deg/sec
Magnetometer	Heading w.r.t. magnetic north	6 deg
Barometer	Air pressure	± 10 N/m^2

The model parameter estimation for the AR.Drone 2 has been done in previous work[15] and this data is used for the model in the off-line training phase to learn a baseline policy matrix. According to this work, white noise is added to the onboard sensors and a sensor delay of 0.015 seconds is used in the simulation to mimic realistic conditions found on the actual AR.Drone 2. The simulation runs at 200 timesteps per second, which is estimated to be the update time of the on-board computer.[15]

Each simulation run the reference signal is a randomly generated set of five manoeuvres. These manoeuvres consist of pitch, roll and yaw manoeuvres or combinations of them. Combined pitch and roll manoeuvres are more common and have a higher weight to occur more often, to ensure the whole state-space is effectively explored. The amplitudes of the manoeuvres are chosen between 0 to 80 degrees tilt angle either as step

input or through a sinusoid reference signal. Since aggressive horizontal manoeuvres are considered to prove intelligent controller selection algorithm the generated manoeuvres will all be executed on a fixed height. Finally it is important to note that every simulation run begins with a start-up manoeuvre to ensure the quadrotor is stable and loitering at 10 meters altitude.

VI. Results

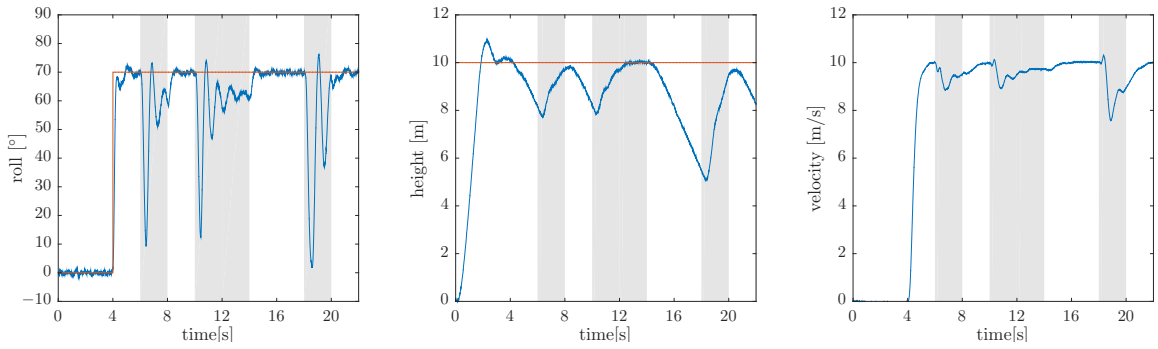
This section presents the results of the intelligent controller selection algorithm for the three proposed reward functions found in Equations 15 to 17. Subsection A will discuss the results obtained in the off-line training phase using different t_{lock} controller lock times, learning rates α, discount factors γ and ϵ-greedy settings from Equation 14. Selecting different learning parameters can influence the learning process and an effort has been made to optimize these learning parameters. However there is no guarantee that there are no better choices for learning parameters. This reveals one of the main disadvantages of implementing reinforcement learning methods: finding the optimum learning parameters is done by trial and error. Subsection B shows the effect of implementing the base-line policy matrix when model parameters differ from the simulation model. An investigation is done to see how the policy adapts to be suitable for the new model and if this adaptation can be realized at sufficient speed to prevent quadrotor crashes and maintain performance.

A. Off-line training

This section will first demonstrate the quadrotor response for the unlearned situation and compare it with the learned behaviour after the off-line training phase. After the best suitable lock time t_{lock} has been determined, the overall policy obtained with the reward functions proposed in Equations 15 to 17 are discussed for varying settings.

During the beginning of the training phase the quadrotor starts with no prior knowledge of which controller is preferable for the current state it is in, since the Q-matrix is initialized as a zero matrix. A Q-value of zero is only obtained with the theoretical highest possible reward for perfect reference tracking, these are optimistic initial conditions. This helps to encourage exploration in the early stages, since collected rewards will be lower than zero, which in turn increases the choice probability of the other controller.

In order to learn the performance of each controller, the quadrotor chooses which one to use according to the ϵ-greedy settings. The initial setting is chosen to be $\epsilon = 0.4$, which explores 40% and exploits the best known option 60% of the state visitations. In an early simulation run, the Q-matrix is hardly filled and the controller selection is mainly exploratory, this can be seen in Figure 6:



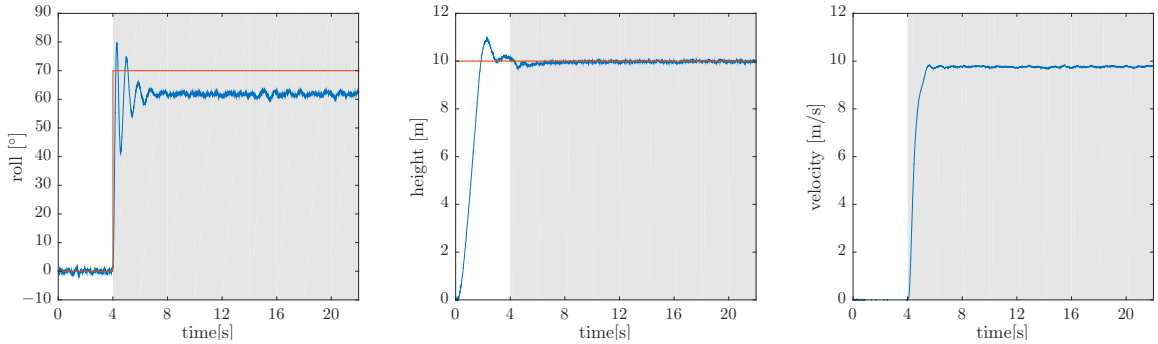
(a) Roll ϕ (b) Height (c) Velocity in y-direction

Figure 6: Roll step input response without prior knowledge (white background = low angle controller and gray background = high angle controller)

The white background in Figure 6 indicates the low angle controller is used, which basically tracks any given reference even if this means losing altitude. The gray background indicates the high angle controller is used instead, this controller tracks the reference altitude while keeping as close as possible to the reference angles. While the maximum horizontal velocity is higher for the low angle controller, the quadrotor loses

height quickly when an aggressive rolling manoeuvre is requested. This loss of altitude degrades performance and can not be sustained for more than a few seconds before it will result in a crash. When the high angle controller is selected, the altitude is recovered while the maximum roll angle is decreased.

After sufficient training has been done through the randomly generated manoeuvres, as have been described in Section V, the quadrotor has learned which is the overall best controller to select in every situation it has encountered. The ϵ parameter is decreased to zero and the known policy is exploited. A roll step input response after the training phase has passed can be seen in Figure 7:

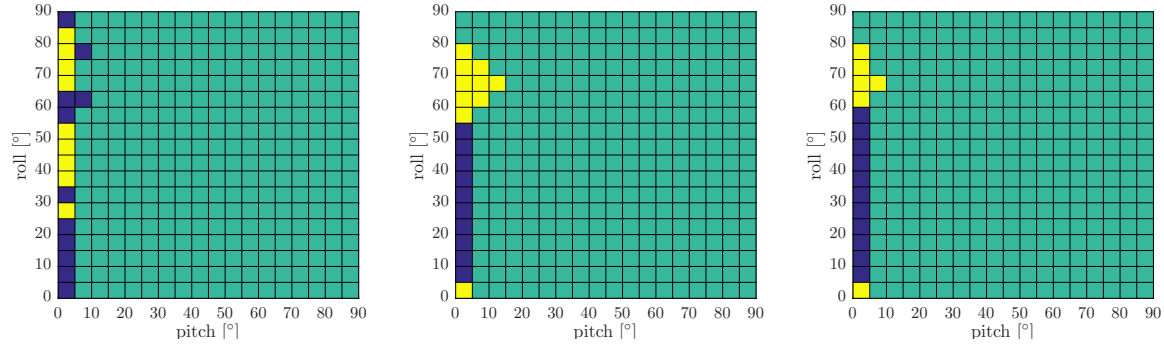


(a) Roll ϕ (b) Height (c) Velocity in y-direction

Figure 7: Roll step input response after training (white background = low angle controller and gray background = high angle controller)

The quadrotor has learned that tracking high pitch and roll angles is best done using the high angle controller. Even though the reference angle can not be tracked, the highest possible roll angle without altitude loss is maintained. This allows for aggressive manoeuvring at all altitudes while keeping the quadrotor airborne and makes these manoeuvres sustainable.

To measure the performance of a controller the intelligent controller selection algorithm is dependent on the lock time t_{lock} that is being used. The quadrotor needs time to react to control inputs and generate a reward according to the followed trajectory. In order to find the appropriate lock time t_{lock} first only the roll axis is considered and the reward function found in Equations 15 is used. The smallest lock time that generates a clear policy of when to switch between low angle and high angle controller can be found in Figure 8:

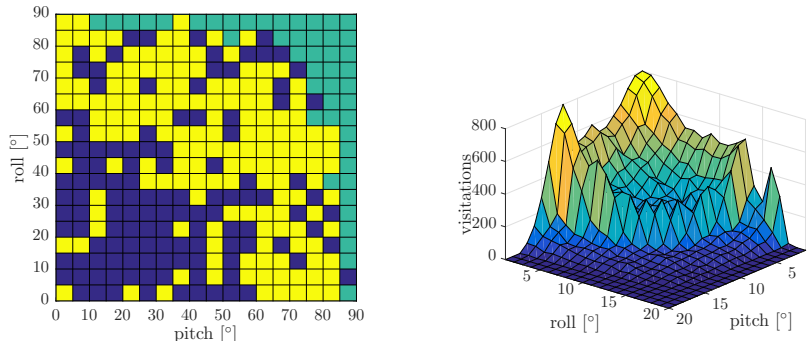


(a) t_{lock} = 300 timesteps (b) t_{lock} = 400 timesteps (c) t_{lock} = 500 timesteps

Figure 8: Roll only for t_{lock} calibration - 500 Epochs (blue = low angle controller, yellow = high angle controller, green = unvisited state)

As can be seen after 500 epochs, using a lock time of t_{lock} = 400 timesteps (2 seconds) and more, the switching point is clearly visible and is around 55-60 degrees roll. One epoch is one simulation run consisting of a set of 5 random manoeuvres over 50 seconds. Using a lower t_{lock} gives insufficient time to evaluate the

performance of the chosen controller and the algorithm will not learn the optimal behaviour. By increasing t_{lock} further the reward received becomes less and less relevant to the state in which the controller was first activated. It is therefore found that a lock time of $t_{lock} = 400$ timesteps (2 seconds) is a good balance between relevance for the state that is rewarded for the controller performance and evaluation time of this performance. This lock time is then used in the case where both the roll and pitch axes are excited. The resulting policy matrix after 5000 epochs along with the visitations per state can be found in Figure 9.

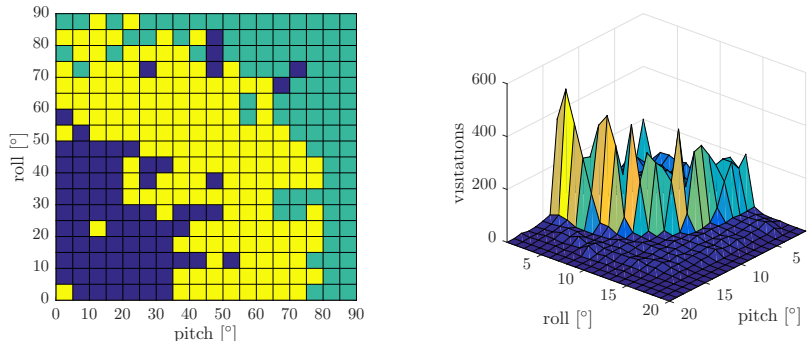


(a) Policy (b) Visitations per state

Figure 9: Absolute Height Tracking Error Reward - $t_{lock} = 400$ timesteps (2 sec) - 5000 epochs - All manoeuvres (blue = low, yellow = high, green = unvisited)

After 5000 epochs it can already be seen that for angles higher than 50 degrees the high angle controller is performing better overall and at angles from 0 to 50 the low angle controller is preferred. However there is still no clear separation between the two controllers.

When considering only step inputs, which is the case in aggressive horizontal manoeuvres, learning is considerably faster then was found in the initial training phase which used all manoeuvres. In Figure 10 it can be seen that there is clear distinction found, the low angle controller performs better for low pitch and roll angles and the high angle controller is best for angles over 50 degrees. By only using step inputs the visitations in the state-space are grouped more around the switching point, which is the area of most interest.



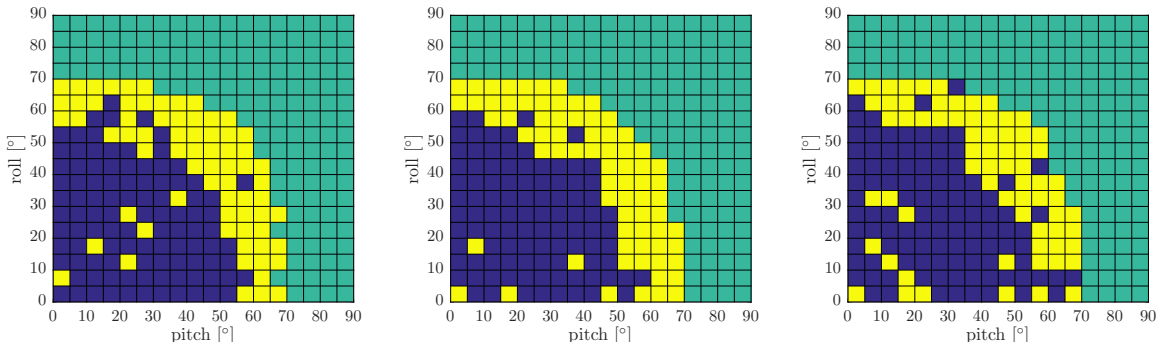
(a) Policy (b) Visitations per state

Figure 10: Absolute Height Tracking Error Reward - $t_{lock} = 400$ timesteps (2 sec) - 5000 Epochs - STEP input manoeuvres (blue = low, yellow = high, green = unvisited)

The change in generated manoeuvres to learn from has increased the convergence speed of the Q-matrix. To further speed up the off-line training phase the search space is reduced by limiting the maximum step input to 70 degrees, since the switching point will be below this angle. This reduces the search space from $18 \cdot 18 = 324$ states to $14 \cdot 14 = 196$ states, a decrease of 40%.

The newly found measures to speed up the training phase have been applied and the resulting policy after

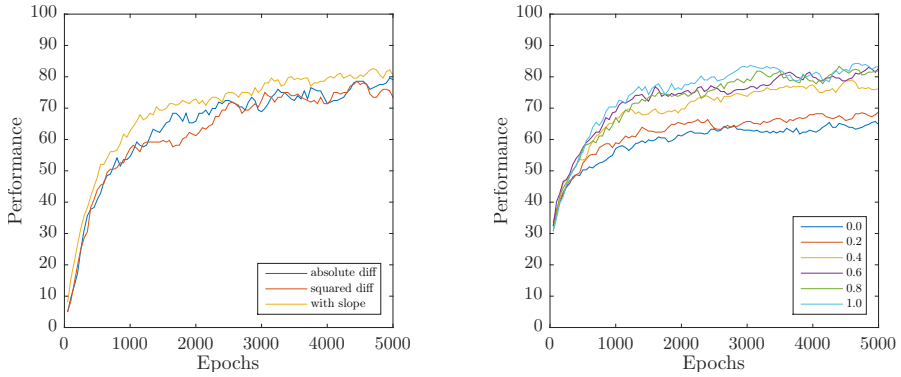
5000 epochs for the three proposed reward functions found in Equations 15 to 17 can be seen in Figure 11. As can be seen the 3 reward functions generate the same overall policy, where the switching point from low angle controller to high angle controller is located around 55 degrees pitch or roll. This switch angle is 10 degrees under the maximum sustainable pitch or roll angle of about 65 degrees during which the vertical component of thrust is exactly equal to the weight of the quadrotor. When getting close to this maximum tilt angle actuator saturation might occur, therefore switching to the high angle controller is preferable. The high angle controller controls height by thrust vectoring through pitch and roll angle changes instead of an average thrust increase as is the case for the low angle controller.



(a) Absolute Height Reward Function (b) Squared Height Reward Function (c) Height Slope Reward Function

Figure 11: Obtained Policy for $t_{lock} = 400$ timesteps (2 sec) after 5000 Epochs - STEP input manoeuvres (blue = low, yellow = high, green = unvisited)

The performance of the different reward functions is measured against the policy that has been obtained after 100000 epochs, this policy is assumed to have converged and be near optimal. The obtained policy is compared to the converged policy and performance is measured as a percentage of similarity. It is however preferable to keep the off-line learning phase as short as possible, while still obtaining a policy that meets performance standards. Figure 12 shows the performance of the proposed reward functions and influence of the ϵ parameter used to define the balance between exploration and exploitation.



(a) Reward functions (b) ϵ-greedy settings

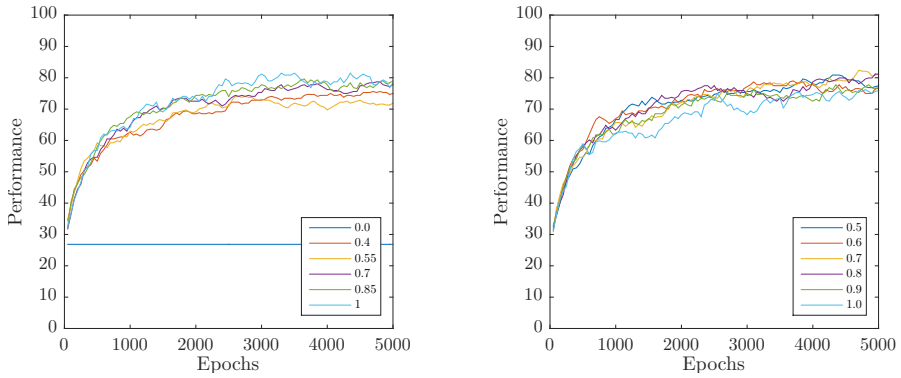
Figure 12: Performance of the different reward function and ϵ-greedy settings - $t_{lock} = 400$ timesteps (2 sec)

Due to the random sequence of manoeuvres with random amplitudes that is being trained on, off-line training phases with the same settings will not be identical. The average over several training phases is taken and the three proposed reward functions are compared to each other. As can be seen the performance of the reward functions is very similar, with an initial high performance gradient until 1000 epochs. This is due to exploring previously unexplored parts of the state-space. Even though the reward function that also incorporates the height derivative appears to perform slightly better, the small performance difference

is however not significant due to the random nature of training manoeuvres. After 5000 epochs, only 5% of training time compared to the performance metric, all reward functions have reached at least 75% performance and only continue to improve marginally between epochs.

Next the influence of the ϵ parameter is investigated and as found in Figure 12b a higher ϵ results in faster learning. This is as to be expected, since high values for ϵ imply random actions are chosen more than the best known action. Choosing the best known action will not improve the current policy, since no new knowledge is obtained. Still with $\epsilon = 0$ the performance increases, this is because of an optimistic Q-matrix initialization. The Q-matrix has been initialized as a zero matrix and obtained rewards will always be lower than zero, which results in selecting every action atleast once when only exploiting Q-matrix values.

In order to better understand the dependence on the parameters of the Q-learning algorithm found in Equation 14 the training phase for the absolute height tracking reward function is done using different values for the learning rate α and discount factor γ. The performance metric used is the same as mentioned earlier, the policy that has been obtained after 100000 epochs. The performance for different learning rates and discount factors can be found in Figure 13:



(a) Learning rate α (b) Discount factor γ

Figure 13: Performance of varying parameters for the Q-learning algorithm - $t_{lock} = 400$ timesteps (2 sec)

As seen above the different settings for learning rates α and discount factors γ do not seem to have much influence on the convergence speed of the policy and final performance. The learning rate determines to what extent the new reward overrides the previous known reward. In a deterministic world a learning rate $\alpha = 1$ is optimal since the reward for a particular action in a state is always the same. Since the simulation model does not contain an atmospheric model and is only subject to sensor delay and noise, the reward will be roughly the same each time an action is taken in the same state. A learning rate of zero causes no state-action values to be updated and the performance does not increase from the initial similarity to the metric. It is important to note that in the on-line implementation phase the learning rate should be lower than $\alpha = 1$ since the real world is stochastic due to atmospheric conditions.

The performance for different discount factors γ can be seen in Figure 13b. The discount factor determines the importance of future rewards, however the quadrotor does not have a terminal state making the future rewards infinite. Since the performance seems to be independent of the chosen discount factor, it is recommended to make the quadrotor short sighted and only consider current rewards. The discount factor $\gamma = 0$ will be used in the on-line implementation and adaptation phase.

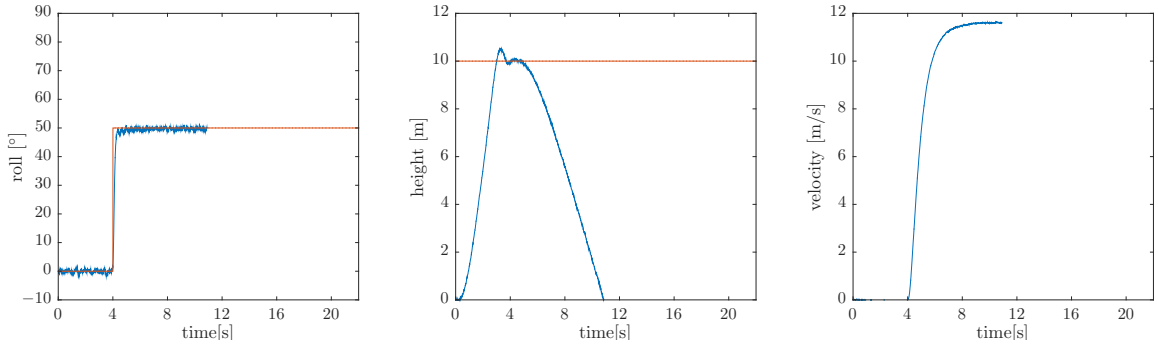
Now that an investigation has been done on the off-line training phase, a base-line policy has been found for controller selection during aggressive horizontal manoeuvres. The next section will investigate if this policy can be implemented on-line and provide sufficient performance even if the simulation model and actual model parameters slightly differ.

B. On-line implementation and adaptation

This section will discuss the implementation of the off-line model in an on-line scenario. The physical parameters of the quadrotor model are slightly altered and an investigation is done in how the policy adapts to this new situation and if this adaptation can be realised fast enough to prevent crashes. The learning rate

α and ϵ-greedy parameters have been significantly decreased to 0.10 and 0.15, respectively. This is because the on-line phase should mainly exploit the learned policy and not take unnecessary risks to explore new options.

The weight of the quadrotor is increased and performance is compared to the policy found using the simulation model to investigate if this baseline policy matrix can be used in the on-line scenario as prior knowledge that helps to skip early exploration and prevent crashes. One of the responses obtained is found in Figure 14:



(a) Roll ϕ (b) Height (c) Velocity in y-direction

Figure 14: On-line roll step input response for $t_{lock} = 400$ timesteps (2 sec) - (white background = low angle controller and gray background = high angle controller)

As can be seen the obtained policy in the off-line training phase is insufficient to prevent a crash. Since the weight of the quadrotor increased, the maximum tilt angle has become lower. The learned policy still selects the low angle controller for an angle of 50 degrees, resulting in a crash.

Since it is found that the learning rate in the on-line phase is considered to be too slow to be of practical use, it is recommended to exploit the policy found during training. The controller lock time t_{lock} can be lowered since performance does not need to be measured anymore. The controller lock time is decreased to $t_{lock} = 100$ timesteps (0.5 seconds) and provides sufficient performance if the simulation model resembles reality.

If the simulation model parameters differ too much from reality it is recommended to use on-line model identification to estimate the actual model parameters[16].[17] These model parameters should then be used in off-line training to form a usable policy.

VII. Conclusion

The intelligent controller selection algorithm that is presented in this paper uses Q-learning to learn a quadrotor controller selection throughout the flight envelope that keeps performance at acceptable levels. It has been tested on a simulation model of the AR.Drone 2 commanded to perform randomly generated sets of manoeuvres. The concept is split in 2 phases, an off-line training phase and an on-line implementation and adaptation phase.

During the off-line training phase it is found that the three proposed reward functions all share comparable performance. It is therefore recommended to keep the reward function simple. The lock time t_{lock} has to be chosen according to the system dynamics, such that there is sufficient time to evaluate the performance of a controller choice. Since the training phase is done in a simulated environment the quadrotor is allowed to crash by trying to improve its current policy. The ϵ-greedy settings can therefore be exploratory only, as this results in faster convergence to the optimal policy. Regardless of the reference signal that needs to be tracked, the best overall controller for each particular state is found by trial and error. After an initial off-line training phase it has been demonstrated that a clear policy is found for controller selection throughout the flight envelope.

Learning is too slow to account for any model differences in the on-line implementation. Instead it is recommended to exploit the policy found in the off-line phase by taking the controller lock time t_{lock} down

and decreasing the ϵ parameter. If the simulation model differs too much from the actual model on-line model identification methods should be used to redetermine the simulation model parameters and redo off-line training to form a suitable policy.

Altogether reinforcement learning proves to be a very powerful tool to help tune complex dynamic systems. The complex system dynamics do not need to be known exactly, only the important performance parameters need to be correctly formulated in a reward function. Using these rewards the Q-Learning algorithm learns the optimal behaviour and tunes quadrotor controller selection entirely by itself.

VIII. Recommendations

First of all it is recommended to add a turbulence model to the simulation environment to closer resemble realistic conditions involving gusts and find a value for the learning rate α that can also be used in the on-line implementation.

The aerodynamic model of the quadrotor models drag opposite to the velocity vector, this is an over simplification of the real case. When a quadrotor performs an aggressive horizontal manoeuvre the high tilt angle will also induce a drag force in the downwards direction due to the frame geometry. This limits the maximum tilt angle that can sustain constant altitude, since this downwards force needs to be compensated by increasing the vertical thrust component.

To give a more definitive answer to the slight changes that occur due to varying the reward function and learning parameters learning rate α and discount factor γ more simulations will have to be performed. This is due to the random nature at which the commanded manoeuvres are generated, which results in a different order of states that is being visited. Taking the average performance over more simulations evens out possible lucky runs, which visit the whole state-space faster thus finding the optimal policy earlier and bad runs, which might visit one region of the state-space above average slowing total exploration and convergence.

Finally this paper has been shown that reinforcement learning can be a powerful tool that is able to tune controller selection for quadrotors such that performance is maximised throughout the flight envelope.

References

[1]D. Pines and F. Bohorquez, "Challenges facing future micro air vehicle development" AIAA J. Aircraft, vol. 43, no. 2, 2006, pp. 290-305

[2]Bouabdallah, Samir, Pierpaolo Murrieri, and Roland Siegwart. "Design and control of an indoor micro quadrotor." Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on. Vol. 5. IEEE, 2004.

[3]Hoffmann, G. M., Waslander, S. L., & Tomlin, C. J. (2006b). "Distributed cooperative search using information-theoretic costs for particle filters with quadrotor applications". In Proceedings of the AIAA guidance, navigation, and control conference, Keystone, CO.

[4]Marzat, J., "Model-based fault diagnosis for aerospace systems: a survey". Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 2012, pp. 1329-1360.

[5]Diebel, J., "Representing attitude: Euler angles, unit quaternions, and rotation vectors". Matrix, 58, 2006, pp. 15-16.

[6]Stengel, R., "Flight Dynamics", Princeton University Press, 2004.

[7]Hoffmann, G. M., Huang, H., Waslander, S. L., & Tomlin, C. J., "Precision flight control for a multi-vehicle quadrotor helicopter testbed". Control Engineering Practice, 19 (9), 2011, pp. 1023-1036

[8]Bangura, M. and Mahony, R., "Nonlinear Dynamic Modeling for High Performance Control of a Quadrotor", Proceedings of Australasian Conference on Robotics and Automation, 2012.

[9]Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction", MIT Press, Cambridge, MA, 1998

[10]Watkins, Christopher John Cornish Hellaby. "Learning from delayed rewards". PhD Thesis, University of Cambridge, 1989

[11]Silvia Ferrari and Robert F. Stengel, "On-line Adaptive Critic Flight Control", Journal of Guidance, Control, and Dynamics 2004 27:5, pp. 777-786

[12]Watkins, C.J. and Dayan, P., "Q-learning". Machine learning, 8(3-4), 1992, pp. 279-292.

[13]Bertsekas, D. P., "Dynamic programming and optimal control" (Vol. 1) (No. 2). Athena Scientific, Belmont MA, 1995

[14]Erik-Jan van Kampen, Q.P. Chu, and J.A. Mulder, "Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics", AIAA Guidance, Navigation, and Control Conference and Exhibit. Keystone, Colorado. 2006

[15]Poppe, C., van Kampen, E., de Wagter, C., de Visser, C.C., Chu, Q.P., "Aggressive Quadrotor Manoeuvring by Using Non-linear Dynamic Inversion with Pseudo-Control Hedging and Flight Mode Transition", Delft University of Technology, 2014

[16]Laban, M., "On-line aircraft aerodynamic model identification". PhD Thesis, Delft University of Technology, 1994

[17]Lombaerts, Thomas, et al. "Online aerodynamic model structure selection and parameter estimation for fault tolerant control." Journal of guidance, control, and dynamics 33.3, 2010, pp. 707-723.

Part II

Additional Results and Simulation

Chapter 2

Additional Results

This chapter will present additional results that were not included in the paper. The evolution of the learned policy over time and some extra performance plots are given. Finally it is noted that one epoch consists of one simulation run that executes one set of manoeuvres.

2-1 Policy evolution

As can be seen in Figure 12 of the paper, the first 1000 epochs of learning show rapid performance increases for each of the reward functions. This is caused by filling previously empty locations in the search space. From 2000 epochs onwards the performance increases at a lower rate, due to learning the best performing controller for each state. This section presents the policy evolution for the three proposed reward functions.

The first reward function is the absolute tracking error and can be found in Equation 2-1:

$$r = \sum_{t=0}^{t_{lock}} - |h_{ref}(t) - h(t)| \tag{2-1}$$

The evolution of the policy per 100 epochs can be found in Figures 2-1 and 2-2. The state-space is filled quickly and the high angle controller states are found early on. The less extreme angles need a bit more visitations to find out that the low angle controller performs better.

Figures 2-3 and 2-4 show how the policy is formed using the reward function found in Equation 2-2

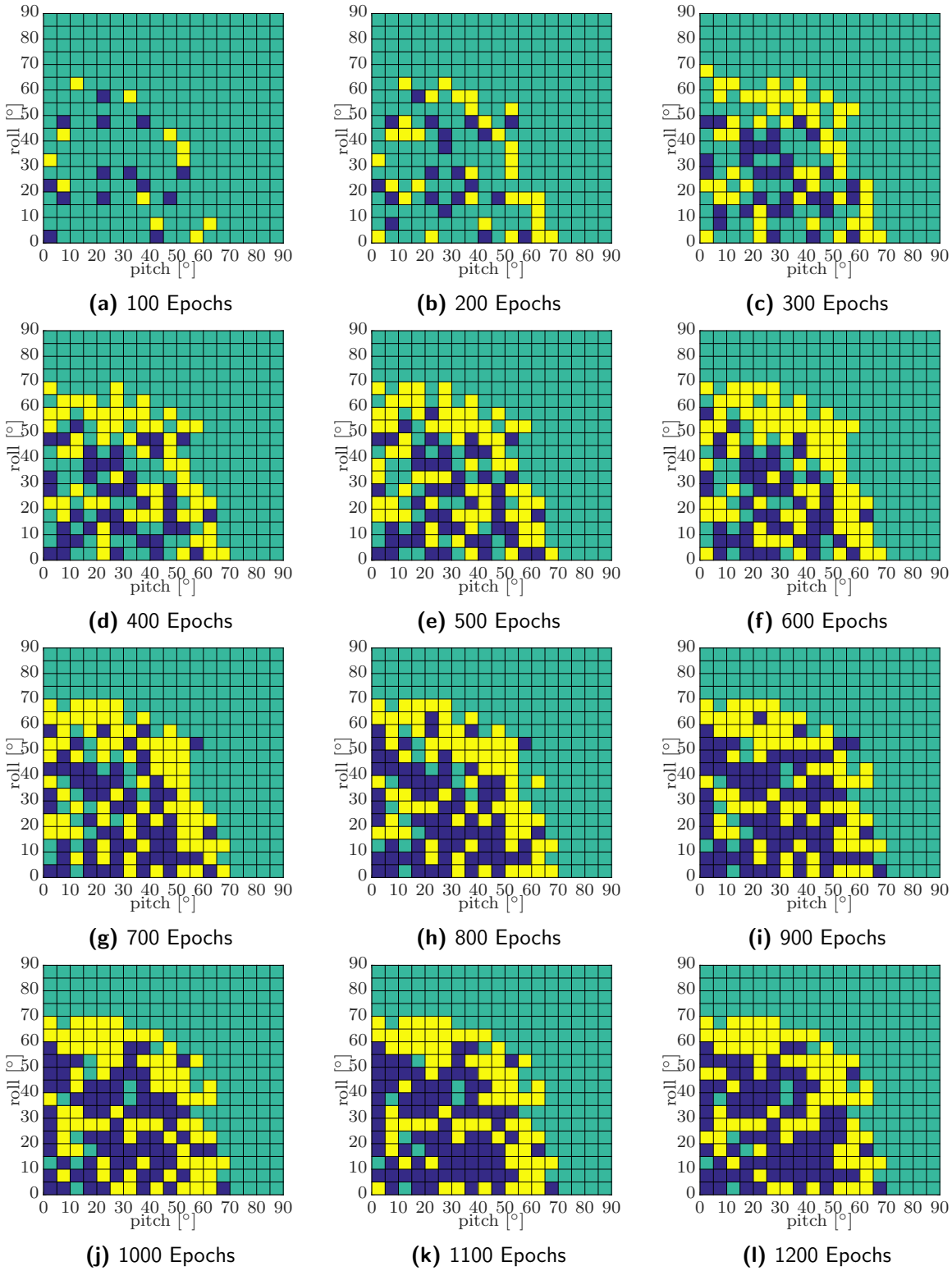
$$r = \sum_{t=0}^{t_{lock}} - (h_{ref}(t) - h(t))^2 \tag{2-2}$$

This reward function gives a higher penalty by using the squared tracking error. As can be seen the evolution is very similar to the previous reward function in Equation 2-1.

The reward function that also takes into account motion towards for away from the reference is found in Equation 2-3

$$r = \sum_{t=0}^{t_{lock}} - |h_{ref}(t) - h(t)| + K_w \cdot sgn(\dot{h}(t)) \cdot (h(t) - h_{ref}(t)) \tag{2-3}$$

Figures 2-5 and 2-6 show this learning progress in intervals of 100 Epochs. When comparing to the two previous reward functions in Equations 2-1 and 2-2 it is noticed that the policy is more clearly earlier on. Especially the outer band is of high angle controller selection is formed faster.



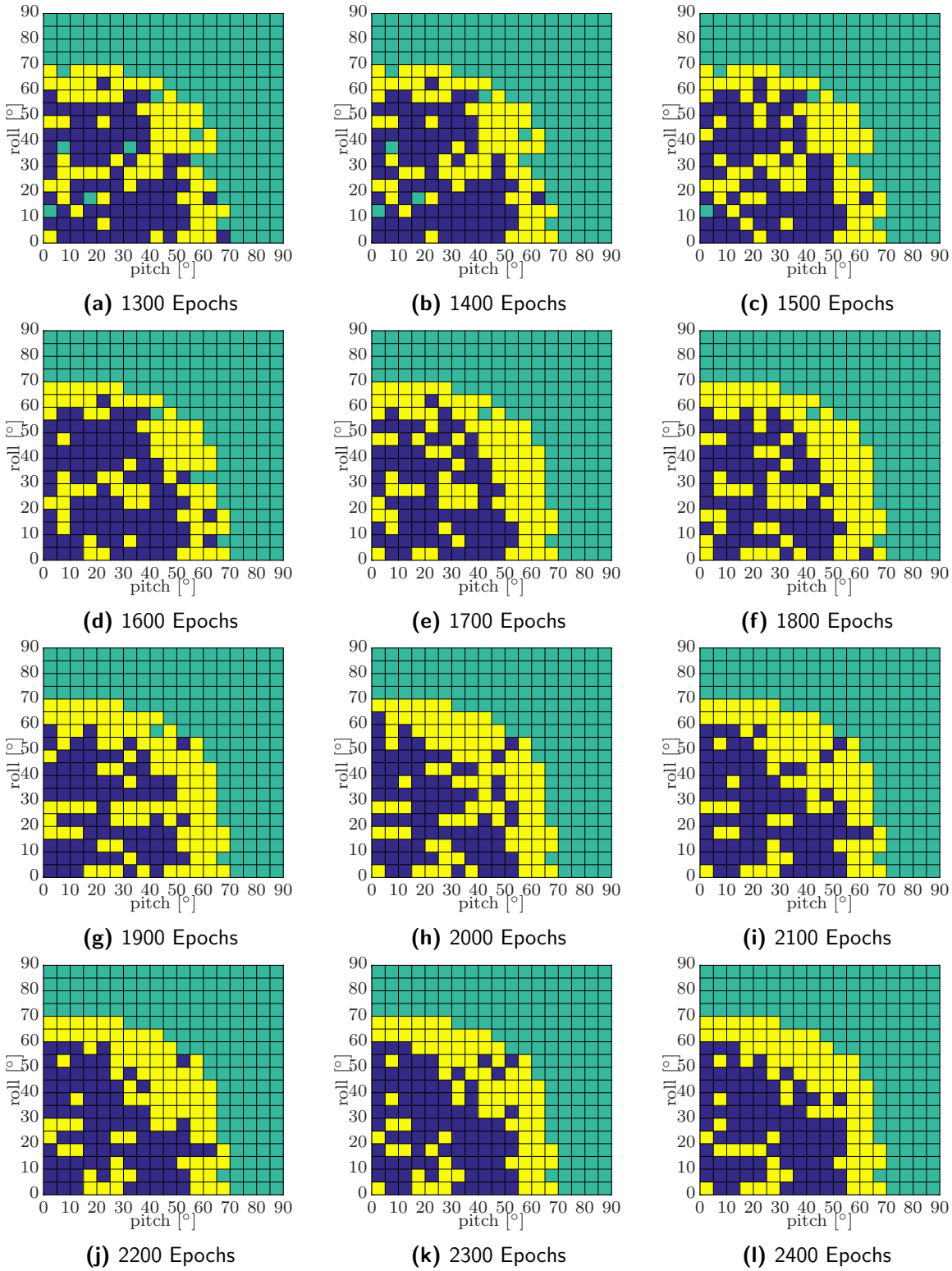
(a) 100 Epochs **(b)** 200 Epochs **(c)** 300 Epochs

(d) 400 Epochs **(e)** 500 Epochs **(f)** 600 Epochs

(g) 700 Epochs **(h)** 800 Epochs **(i)** 900 Epochs

(j) 1000 Epochs **(k)** 1100 Epochs **(l)** 1200 Epochs

Figure 2-1: Policy Matrix Progress Absolute Tracking Error for Epochs 100 to 1200



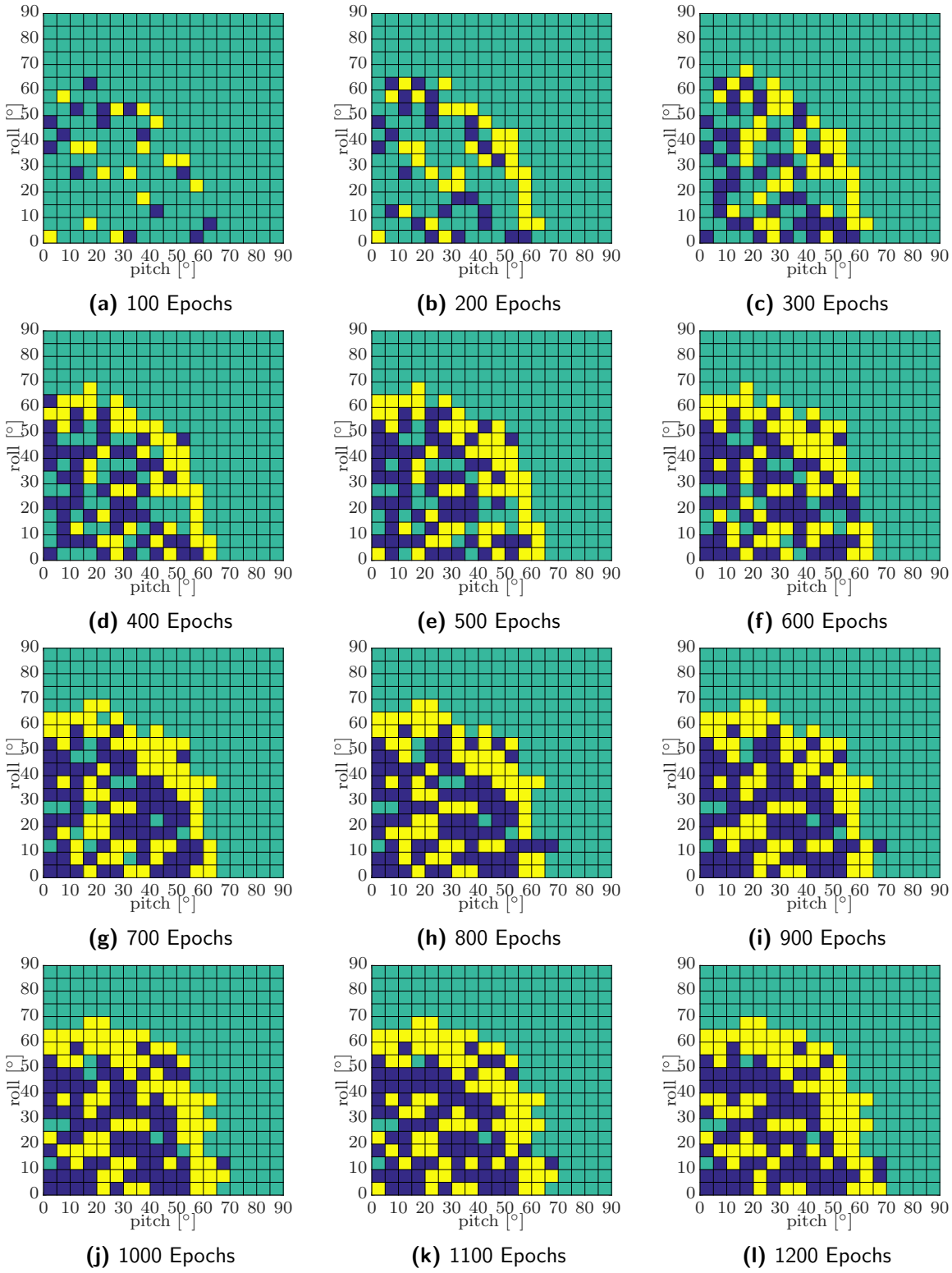
(a) 1300 Epochs **(b)** 1400 Epochs **(c)** 1500 Epochs

(d) 1600 Epochs **(e)** 1700 Epochs **(f)** 1800 Epochs

(g) 1900 Epochs **(h)** 2000 Epochs **(i)** 2100 Epochs

(j) 2200 Epochs **(k)** 2300 Epochs **(l)** 2400 Epochs

Figure 2-2: Policy Matrix Progress Absolute Tracking Error for Epochs 1300 to 2400



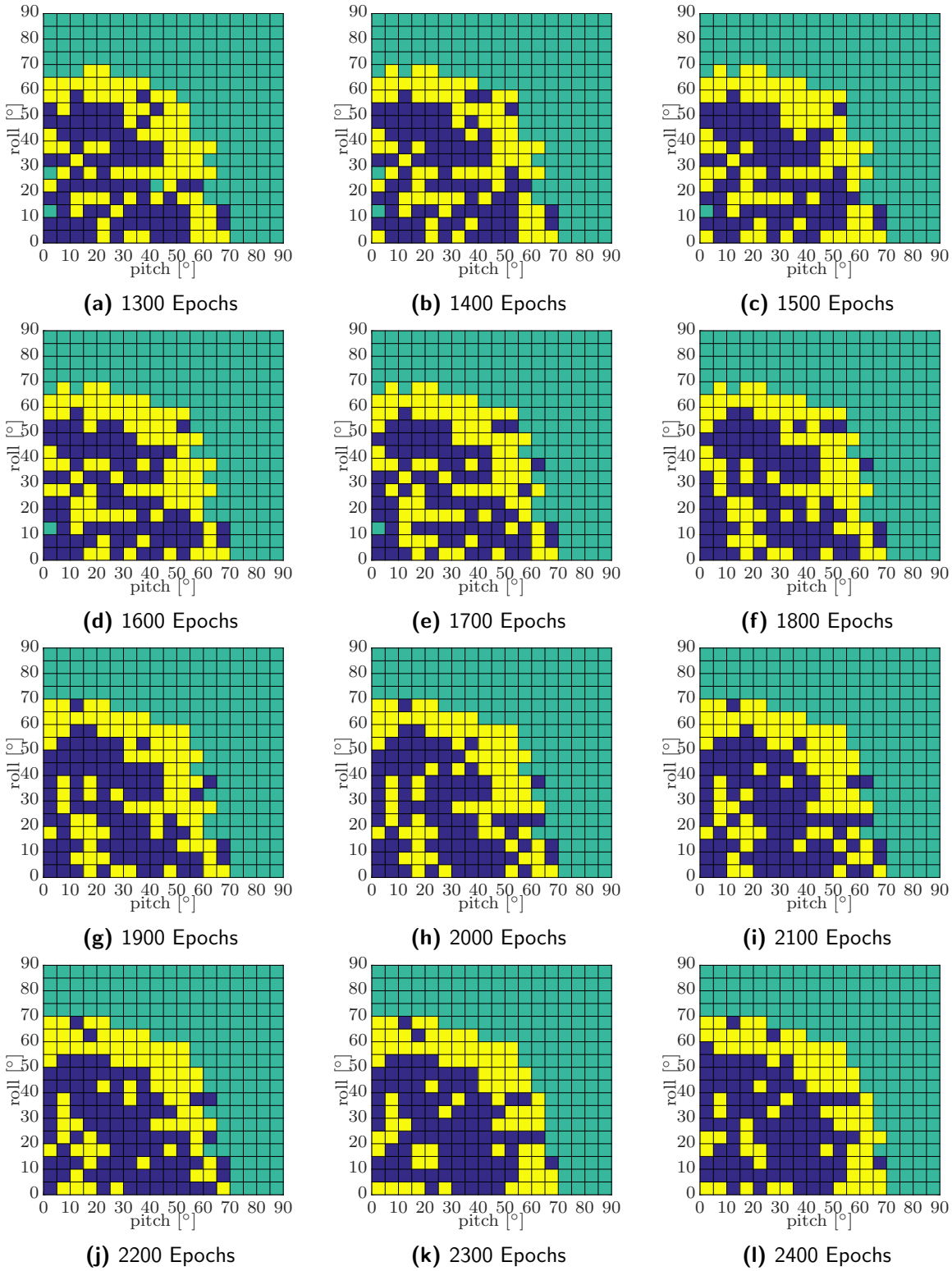
(a) 100 Epochs **(b)** 200 Epochs **(c)** 300 Epochs

(d) 400 Epochs **(e)** 500 Epochs **(f)** 600 Epochs

(g) 700 Epochs **(h)** 800 Epochs **(i)** 900 Epochs

(j) 1000 Epochs **(k)** 1100 Epochs **(l)** 1200 Epochs

Figure 2-3: Policy Matrix Progress Squared Tracking Error for Epochs 100 to 1200



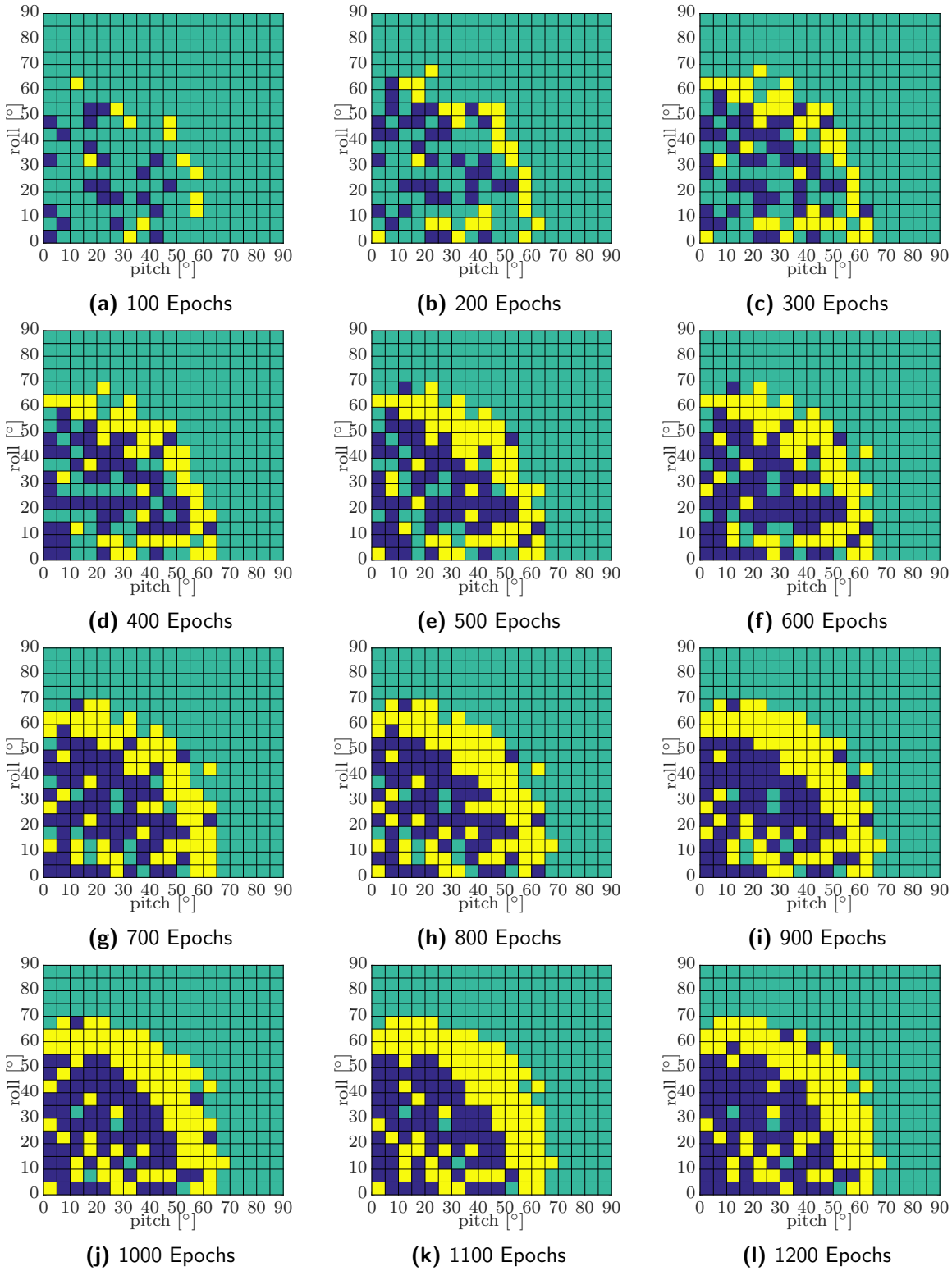
(a) 1300 Epochs **(b)** 1400 Epochs **(c)** 1500 Epochs

(d) 1600 Epochs **(e)** 1700 Epochs **(f)** 1800 Epochs

(g) 1900 Epochs **(h)** 2000 Epochs **(i)** 2100 Epochs

(j) 2200 Epochs **(k)** 2300 Epochs **(l)** 2400 Epochs

Figure 2-4: Policy Matrix Progress Squared Tracking Error for Epochs 1300 to 2400



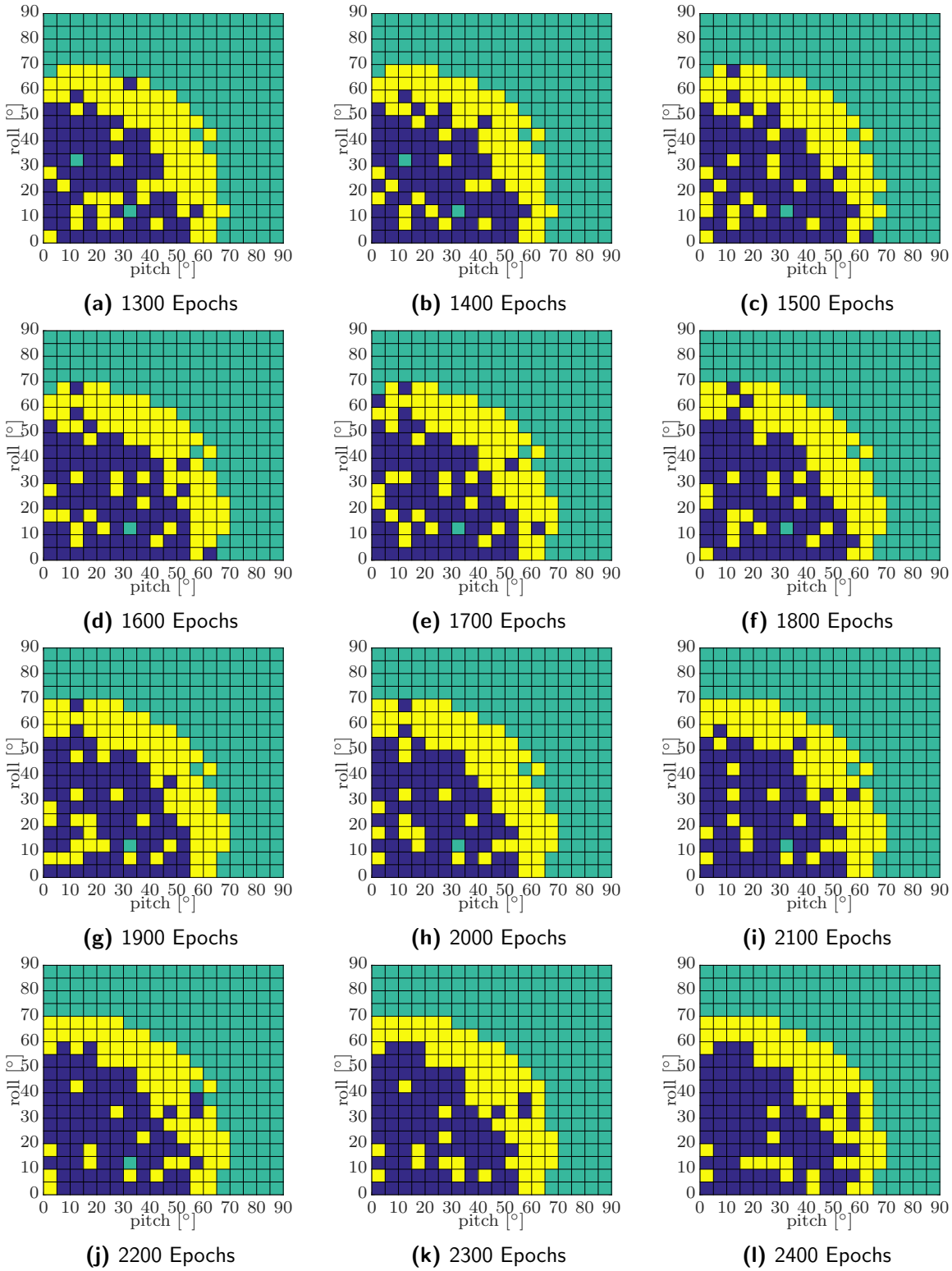
(a) 100 Epochs **(b)** 200 Epochs **(c)** 300 Epochs

(d) 400 Epochs **(e)** 500 Epochs **(f)** 600 Epochs

(g) 700 Epochs **(h)** 800 Epochs **(i)** 900 Epochs

(j) 1000 Epochs **(k)** 1100 Epochs **(l)** 1200 Epochs

Figure 2-5: Policy Matrix Progress Absolute Tracking Error with Slope for Epochs 100 to 1200



(a) 1300 Epochs **(b)** 1400 Epochs **(c)** 1500 Epochs

(d) 1600 Epochs **(e)** 1700 Epochs **(f)** 1800 Epochs

(g) 1900 Epochs **(h)** 2000 Epochs **(i)** 2100 Epochs

(j) 2200 Epochs **(k)** 2300 Epochs **(l)** 2400 Epochs

Figure 2-6: Policy Matrix Progress Absolute Tracking Error with Slope for Epochs 1300 to 2400

2-2 Performance Results

This section shows additional performance results obtained with extra simulations runs. The performance metric that has been used is the policy that was found after 100000 epochs and can be found in Figure 2-7:

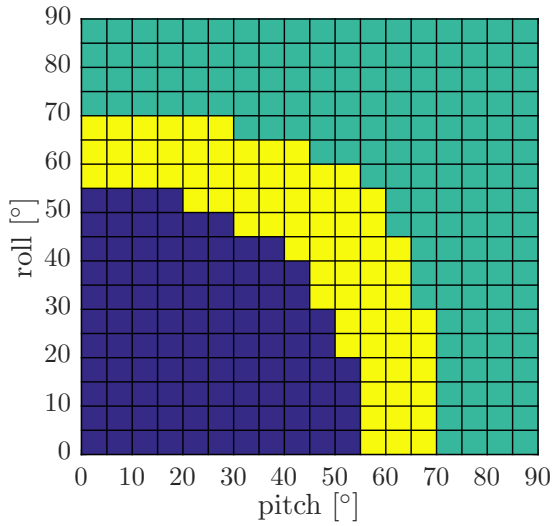


Figure 2-7: Performance metric after 100000 epochs

It has to be noted that for the simulation runs with 70 degrees as maximum tilt angle, this performance metric was also cropped to angles from 0 to 70 degrees to measure performance.

In the paper the Q-matrix was initialised as a zero matrix, which is a very optimistic setting since a value of zero corresponds with perfect tracking. This was done to stimulate early exploration. Figure 2-8 shows the performance for the three proposed reward functions using a randomly initialised Q-matrix:

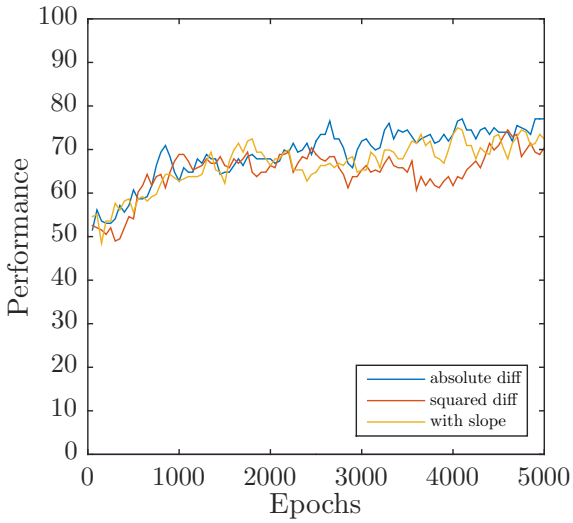


Figure 2-8: Performance for reward functions using randomly initialised Q-matrix

As can be seen the initial performance is higher, due to the fact that one controller is initialised with a higher Q-value than the other controller option. By comparing the found policy with the performance metric there are already some similarities, which is not the case for a zero initialisation where both controller perform equal. The final performance is slightly lower compared to zero initialisation of the Q-matrix, however this might be due to the randomness of exploring the state-space.

Chapter 3

Simulation Model

This chapter aims at explaining the integration of the reinforcement learning algorithm in the quadrotor simulation. The simulation model structure is presented and the individual blocks are discussed in detail. Finally the specifications of the AR.Drone 2.0 quadrotor that is used in the simulation are given and are elaborated upon.

3-1 Model Overview

The top level system overview of the simulation is found in Figure 3-1:

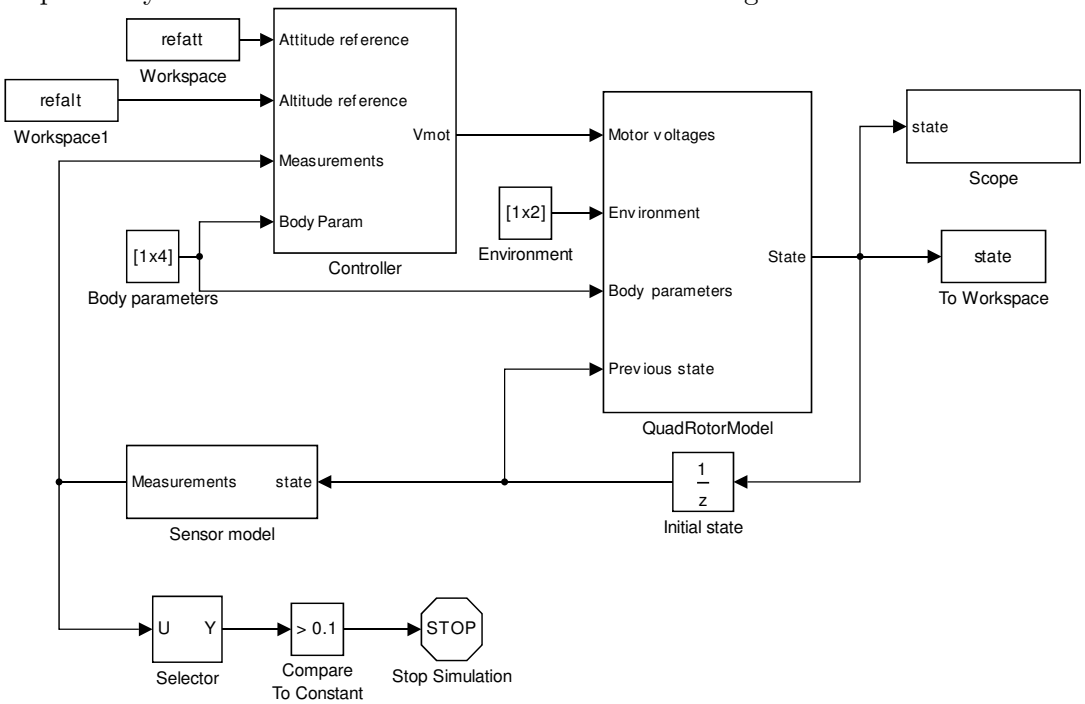


Figure 3-1: Top level system overview

The main blocks that form the simulation model are: the controller block, the quadrotor model block and the sensor model block. Input blocks are the reference attitude, the reference altitude and some constants. These constants are the body parameters, which describe the quadrotor geometry and weight, and the environmental parameters, which are the constants used for gravity $g = 9.81$ m/s^2 and air density $\rho = 1.225$ kg/m^3.

The following sections will discuss the flow through these three main blocks and their underlying processes.

3-1-1 Controller block

The controller block ultimately determines what motor voltages should be commanded in order to track a given reference attitude and altitude and feeds these to the quadrotor model block. The controller block takes as inputs the reference signals, measurements of the current state and the body parameters of the quadrotor. These inputs are fed to two controllers, the small and the large angle controller. This can be seen in Figure 3-2:

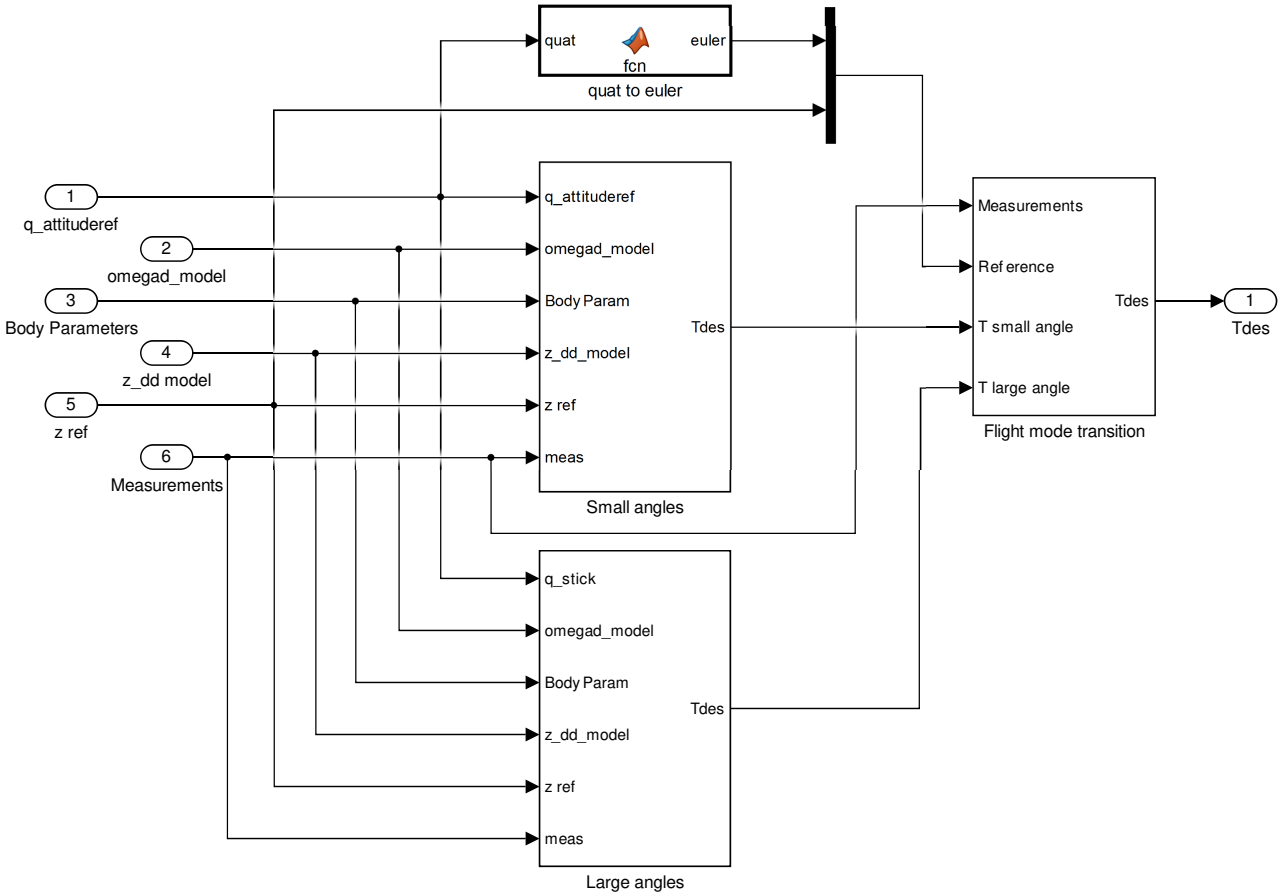


Figure 3-2: Dual controller structure block

The small angle controller block is designed to be used from hover to medium tilt angles and the large angle controller is used at large tilt angles close the maximum tilt angle. The

difference between the controllers is the manner in which they control altitude. The small angle controller follows the commanded tilt angle and controls altitude by varying average thrust, while the large angle controller holds altitude by varying the tilt angle and keeps an average thrust for the commanded tilt angle. This large angle approach helps precise tracking at more extreme angles, while avoiding actuator saturation problems that occur around the maximum tilt angle.

The desired thrust from both controllers is fed into the flight mode transition block, which determines when to use which controller output. The flight mode transition block contains the intelligent controller selection algorithm that learns controller performance for varying conditions using reinforcement learning. The flight mode transition block can be found in Figure 3-3:

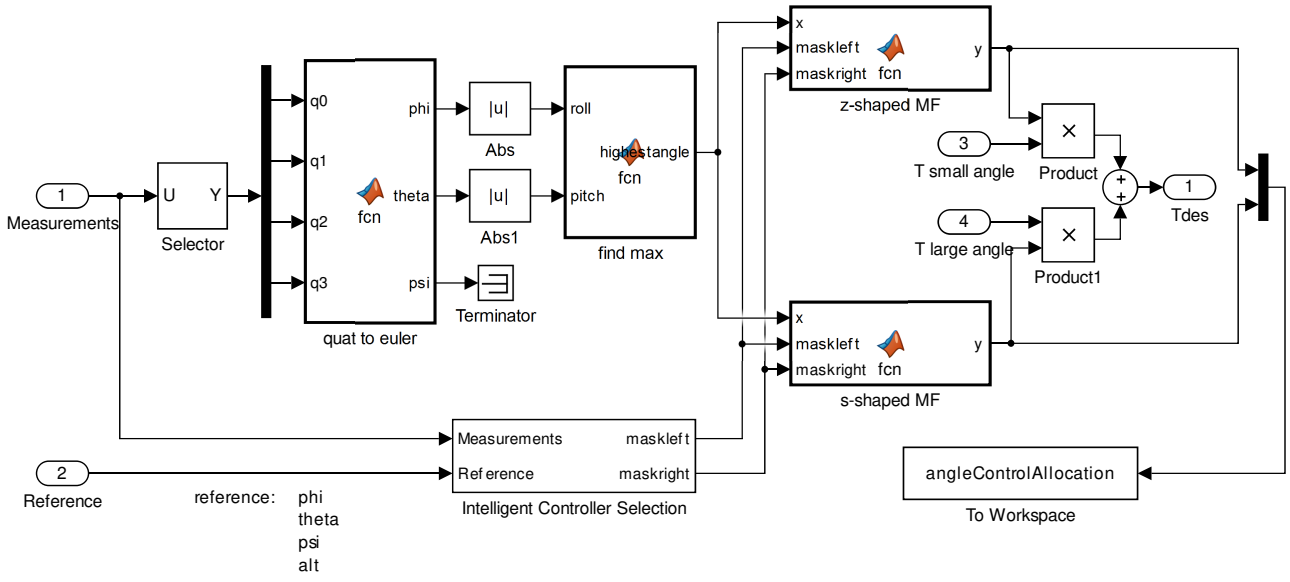


Figure 3-3: Flight mode transition block

The flight mode transition block receives state measurements, reference signals and the 2 controller inputs. Depending on the state measurements a controller output is chosen, using S- and Z- masking functions that are configured by the intelligent controller selection block. These masking functions determine to what extent the controller output is chosen and will contribute to the desired thrust output. In order to clearly investigate the controller performance either the small angle controller commands or the large angle controller commands are passed on, no mixing occurs. The desired thrust output is passed through a Thrust-to-Voltage block to ultimately command the quadrotor motors the desired voltage.

The intelligent controller selection block contains the reinforcement learning algorithm that

learns when each controller performs best, the block can found in Figure 3-4:

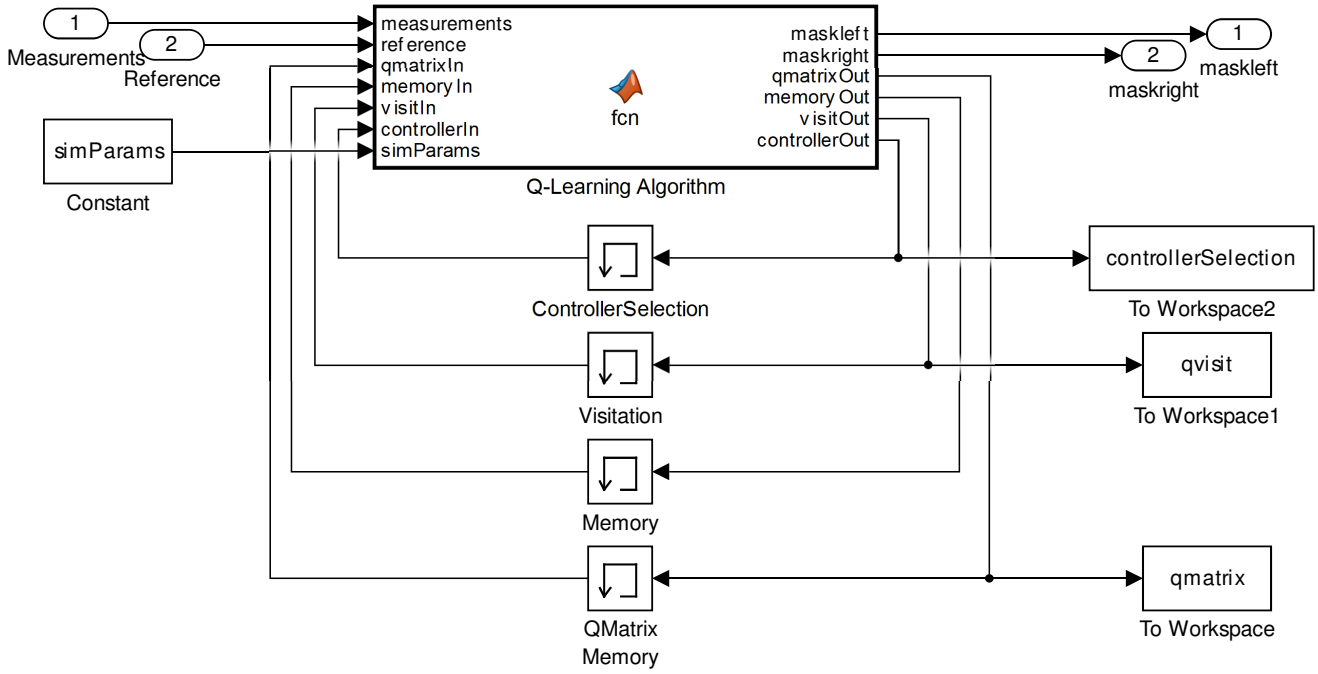


Figure 3-4: Q-Learning algorithm block

The Q-Learning Algorithm block is the learning part of the adaptive controller. It takes as inputs the current state of the quadrotor, the reference signal and the Q-learning simulation parameters learning rate α, discount factor γ, controller lock time t_{lock} and ϵ-greedy settings. Several memory blocks are used to pass variables such as the current learned Q-matrix, state-space visitations and controller selection histories to the next timestep. In the last timestep of the simulation these three matrices are saved. The Q-learning algorithm prescribes the mask settings to be used in the flight mode transition block.

At the start of the simulation the quadrotor is given an empty Q-matrix and the Q-matrix is passed on every timestep and will slowly get filled throughout the simulation. During a controller lock time the algorithm is simply collecting reward and only after the controller lock time has passed it will update the corresponding Q-matrix entry. The Q-Learning Algorithm uses the following logic:

Repeat (for each simulation run)
- Initialize s
- Start counter t
- Repeat (for each timestep of simulation run)
  - **If** counter $t = t_{lock}$:
    - Collect reward r for timestep and save accumulated reward into Q-matrix for the state where the controller has been locked.
    - Save the current lock state s
    - Choose a random controller with probability ϵ, else observe s and choose best known controller.
    - Reset counter to $t = 0$.
  - **else:**
    - Collect reward r according to reward function chosen.
    - Increment counter t.

Figure 3-5: Intelligent Controller Selection Algorithm

For the moment the mask settings are set using extreme values, such that in one particular state only one controller is chosen and no mixing occurs. This is done because the focus of this thesis project is autonomously determining which controller should be used throughout the flight envelope and where the switching point between controllers occurs. The influence of mixing both controllers should be separately investigated.

Finally the chosen controller passes on a desired thrust, which is translated into a desired motor voltage. This motor voltage is passed on to the quadrotor model block that will be discussed in the next section.

3-1-2 Quadrotor Model block

The quadrotor model block takes as inputs the motor voltages from the controller block, body and environmental parameters and the previous state. The body and environmental parameters are the same as previously discussed. Inside the quadrotor model block there are three blocks, the engine block, the forces and moments block and finally the dynamic model block, this can be seen in Figure 3-6:

The following sections will discuss each block briefly.

Engine block

The engine block takes as input the desired motor voltages and the previous state. Using a model of the motor the engine rotational velocity ω and the provided power P_m are passed into the forces and moments and dynamic model blocks, respectively.

The engine rotational velocity is modelled as a first order system:

$$\dot{\omega} I_e = K_V V_e - K_\omega \omega \tag{3-1}$$

with ω the rotational velocity, I_e the engine moment of inertia, V_e the applied voltage and K_V and K_ω gains to tune the model.

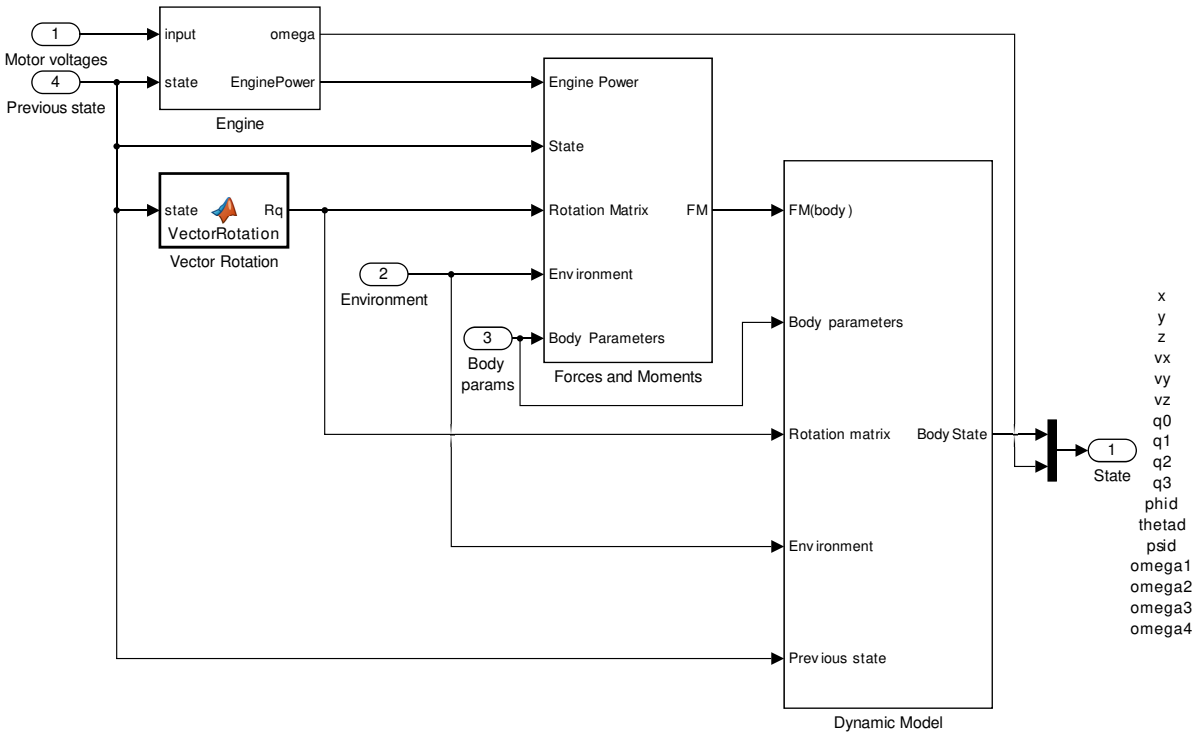


Figure 3-6: Quadrotor model block

Finally the provided power P_m is found through a cubic relation with ω using:

$$P_m = K_{P_m}\omega^3 \tag{3-2}$$

The tuning is done to make the model behave as the engine of the AR.Drone 2.0. The engine power generated is passed into the forces and moments block discussed in the next section.

Forces and Moments

Now that engine power that is generated have been determined, the forces and moments acting on the quadrotor need to be quantified. Equation 3-3 shows the forces acting on the quadrotor:

$$\mathbf{F}_B = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \mathbf{F}_{fus} + \mathbf{F}_{rot} \tag{3-3}$$

The first contribution is due to gravity acting along the z-axis with m the total mass and g the gravitational acceleration. The main part of the fuselage force $\mathbf{F}_{fus}$ is the drag force D in opposite direction to the airspeed, the fuselage also produces a lift force L perpendicular to the airspeed depending on the frame geometry (Hoffmann, Huang, Waslander, & Tomlin, 2011). Finally the rotor force $\mathbf{F}_{rot}$, which is the sum of thrust T_i generated by each rotor.

The aerodynamic model is reduced to a static relation between the thrust T, torque τ and rotor rotational velocity ω as suggested in previous work (Bangura & Mahony, 2012) based on the momentum theory:

$$\mathbf{F}_{rot} = \sum_{i=1}^{4} T_i = \sum_{i=1}^{4} C_T \omega_i^2 \tag{3-4}$$

with C_T a positive constant. The moment vector $\mathbf{M}_B$ consists of three contributions and is found in Equation 3-5:

$$\mathbf{M}_B = \mathbf{M}_{fus} + \mathbf{M}_{rot} + \mathbf{M}_G \tag{3-5}$$

The fuselage moment $\mathbf{M}_{fus}$ caused by the frame geometry is only a minor contribution. The moment vector due to rotor forces $\mathbf{M}_{rot}$ is caused by the difference in engine thrust T_i, difference in engine torque τ_i and blade flapping effects. The difference in engine thrust causes moments about the x- and y-axis, while difference in engine torque causes a moment about the z-axis as can be seen in Equation 3-6:

$$\mathbf{M}_{rot} = \begin{bmatrix} (T_3 + T_4)l - (T_1 + T_2)l \\ (T_1 + T_4)l - (T_2 + T_3)l \\ \tau_1 - \tau_2 + \tau_3 - \tau_4 \end{bmatrix} \tag{3-6}$$

$$\tau_i = C_\tau T_i \tag{3-7}$$

with T_i the thrust generated by motor i = 1,2,3,4, l the motor distance to the symmetry-axis, τ_i the generated torque and C_τ a positive constant. The gyroscopic moment $\mathbf{M}_G$ is the sum of the gyroscopic moments occurring in each rotor given by (Bangura & Mahony, 2012):

$$\mathbf{M}_G = -\sum_{i=1}^{4} (-1)^{i+1} \omega_i I_r \omega_i \tag{3-8}$$

with ω_i the rotor rotational speed and I_r the rotor moment of inertia. Finally the lateral effects that usually occur in helicopters in horizontal flight are cancelled due to the symmetry of the four rotors (Hoffmann et al., 2011).

The forces and moments found using these calculations are passed into the dynamic model block and will determine new quadrotor state.

Dynamic Model Block

This last section will discuss the equations of motion of the quadrotor. Figure 3-7 shows a schematic view of a quadrotor in 'X' configuration. The basic working principle is that all control is done by changing the thrust of two sets of counter-rotating propellers (Hoffmann et al., 2011).

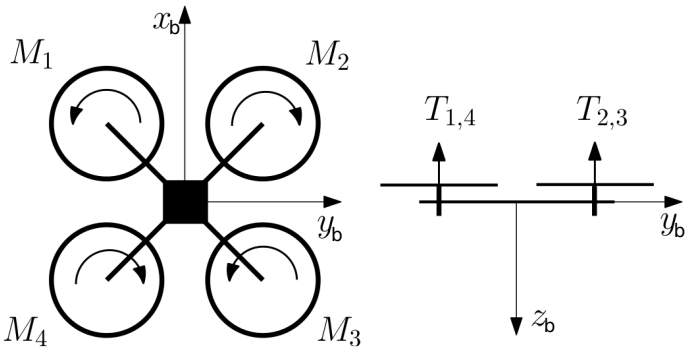


Figure 3-7: A schematic view of a quadrotor system

A body reference B is defined with its origin in the Center of Gravity (CG) of the quadrotor, which is the intersection of the beams coming from the engines. The x_B-axis points forward, in between motor M_1 and M_2, the z_B-axis points downward and the y_B-axis completes the right hand coordinate system.

Another reference frame E is defined as North-East-Down Earth-Fixed with the x_E and y_E-axis pointing North and East and the z_E-axis pointing down. The rotation of the body frame in the Earth-Fixed frame is defined using the unit quaternion $\mathbf{q} = [q_0 \;\; q_1 \;\; q_2 \;\; q_3]^T$. Because of the limitations of using Euler angles, quaternions are used to overcome this problem (Diebel, 2006). The kinematic relations are found in Equation 3-9 and 3-10:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}_E = R(\mathbf{q})^T \mathbf{V}_B \tag{3-9}$$

$$\dot{\mathbf{q}}(\mathbf{q}, \boldsymbol{\omega}_B) = \frac{1}{2} W'(\mathbf{q})^T \boldsymbol{\omega}_B \tag{3-10}$$

with $\mathbf{V}_B$ the velocity in the body frame and angular rates in the body frame $\boldsymbol{\omega}_B = [p \;\; q \;\; r]^T$. The rotation matrix $R(\mathbf{q})$ and quaternion rate matrix $W(\mathbf{q})$ are defined as (Diebel, 2006):

$$R(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3-11}$$

$$W'(\mathbf{q}) = \begin{bmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \tag{3-12}$$

The equations of motion are derived from Newton's Second Law (Mulder, van Staveren, van der Vaart, & de Weerdt, 2007):

$$\mathbf{F} = m\ddot{\mathbf{x}} \tag{3-13}$$

$$\mathbf{M} = \mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} \tag{3-14}$$

with $\mathbf{M} = [L \;\; M \;\; N]^T$ the moments acting around the body x, y and z-axis. $\boldsymbol{\omega} = [p \;\; q \;\; r]^T$, F the force vector and I the inertia matrix give by:

$$\mathbf{I} = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{pmatrix} \tag{3-15}$$

Using these relations the next state of the quadrotor is found and each timestep is saved. Following the simulation diagram this new state is passed into the sensor model.

3-1-3 Sensor Model block

The sensor model block is the last main block in the simulation and mimics the sensors found on the AR.Drone 2.0. White noise of intensity $1 \cdot 10^{-5}$ is added to the quadrotors new state and a sensor delay block of 3 timesteps is also applied. This noisy delayed data is finally passed through a low pass filter and fed back through the controller block and is the quadrotors state as perceived by the controller.

Finally it has to be noted that the simulation runs using a fixed discrete step size of 0.005 seconds, which is estimated to be the processing speed of the AR.Drone 2.0

3-2 Hardware: AR.Drone 2.0

The AR Drone 2.0 quadrotor by Parrot is used to base the simulation model on. The quadrotor is designed by the French company Parrot and is originally meant to be controlled by mobile or tablet operation systems. This section will quickly discuss its features.

Airframe

The airframe of the AR.Drone 2.0 is constructed of nylon and carbon fiber parts and comes in two types, an indoor hull and an outdoor hull for external flight, as can be seen in Figure 3-8. The AR.Drone 2.0 measures 51 cm by 51 cm when using the indoor hull and becomes slightly smaller and lighter when this extra protection is removed for outdoor flight (Parrot, 2015).

Power plant

The AR.Drone 2.0 has four actuators, which are the four motors oriented in an X configuration. They are driven through a reduction gear and rotate either clockwise or counter-clockwise depending on the position within the frame. It uses a 3 element 1000 mAh LiPo battery to power the motors that provides around 12 minutes of autonomy on a single charge (Parrot, 2015).

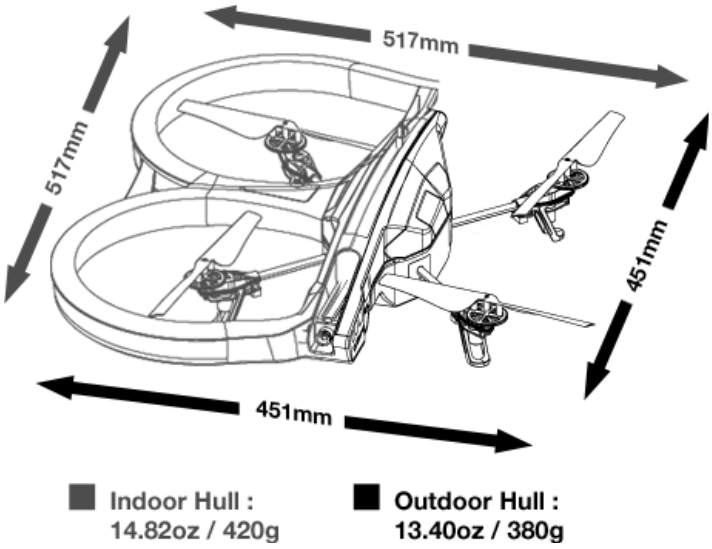


Figure 3-8: The AR Drone 2.0 (Parrot, 2015)

Motor and Electronic Speed Controller (ESC) specifications:

- 4 brushless inrunner motors. 14.5W 28,500 RPM
- Low noise Nylatron gears for 1/8.75 propeller reductor
- 8 MIPS AVR CPU per motor controller
- 3 elements 1000 mAH LiPo rechargeable battery (Autonomy: 12 minutes)

Onboard electronics

At the heart of the quadrotor are the electronics, this is where the magic happens. The AR.Drone 2.0 features a fast processor which integrates data received from various sensors, such as the accelerometer and gyroscope. In order to autonomously navigate the quadrotor is equipped with a GPS module to find its absolute position. (Parrot, 2015).

Sensor and processing specifications:

- 1GHz 32 bit ARM Cortex A8 processor with 800MHz video DSP TMS320DMC64x
- Linux 2.6.32
- 1GB DDR2 RAM at 200MHz
- USB 2.0 high speed for extensions
- Wi-Fi b/g/n
- 3 axis gyroscope 2000°/second precision
- 3 axis accelerometer +/- 50mg precision
- 3 axis magnetometer 6° precision

- Pressure sensor +/- 10 Pa precision
- GPS-module
- Ultrasound sensors for ground altitude measurement
- 30 FPS front 720p camera with 93° lens
- 60 FPS vertical QVGA camera with 64° lens for ground speed measurement

Part III

Literature Study

Chapter 4

Reconfigurable Control Techniques

In order for a UAV to react to failures, external disturbances and other unexpected situations a reconfigurable control architecture is necessary. This chapter will investigate controller structures that are able to reconfigure and adapt to varying situations. These controllers change their parameters to maintain performance and keep safety at a high level.

This chapter first looks into passive fault tolerant control before moving to more sophisticated active methods. The active fault tolerant control methods can be further divided into on-line automatic redesign, such as model predictive control or precomputed control laws, like gain scheduling. The design approaches that will be discussed can be found in Figure 4-1.

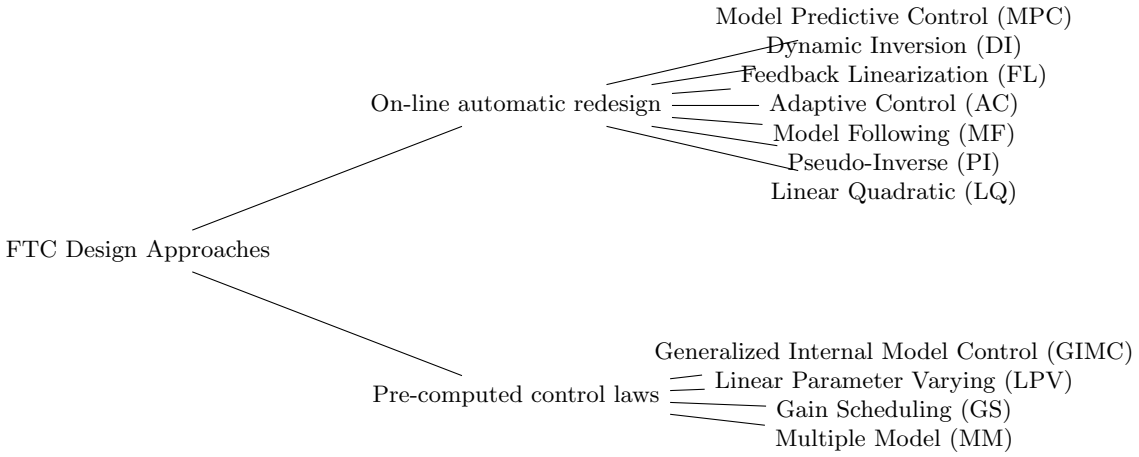


Figure 4-1: Common FTC Methods

4-1 Passive Fault Tolerant Control

Passive Fault Tolerant Control methods aim at achieving insensitivity to certain faults by creating a robust system. This fault-tolerance is often achieved by means of representing

certain faults as uncertainty in the system so that a robust controller can be designed.

4-1-1 Robust Control

Robust control designs a controller that not only meets the design specifications under normal operating conditions, but also maintains its performance in case of some faults. These approaches are usually based on quantitative feed back theory (Keating, Pachter, & Houpis, 1997) or robust H_∞ controller design (Zhou & Ren, 2001).

4-1-2 Reliable Control

A passive controller that aims at making the closed-loop system reliable so that it keeps its stability/performance in case some anticipated fault occurs. Usually a controller optimizes the worst-fault performance for all possible anticipated faults. For reliable control, complete failures may only occur in a predefined set of sensors and/or actuators (Veillette, Medanic, & Perkins, 1992).

4-2 On-line Automatic Redesign

This is the first of two sections dealing with active fault tolerant control methods. The active methods differentiate themselves from passive approaches because they take fault information explicitly into account and do not work with a static nominal model.

4-2-1 Model Predictive Control (MPC)

With its self-reconfiguration capability, Model Predictive Control (MPC) is a good candidate for achieving fault-tolerance, MPC is based on numerical optimization and is categorised as an optimal control strategy. Future control inputs are optimized and system responses are predicted at an interval using a system model. Important characteristics such as stability, optimality and robustness are well understood for MPC. A basic MPC method contains: prediction, optimization and receding horizon implementation. (Cannon, 2015).

Prediction

Making use of a dynamic model the future response of the system can be predicted. Consider the following discrete-time state-space system

$$x(k+1) = Ax(k) + Bu(k) \tag{4-1}$$

where $x(k)$ and $u(k)$ are the model state and input vector at the kth sample. Corresponding state predictions are generated by simulating the model given predicted input sequence. These

prediction are done over the prediction horizon, which is a certain time N intervals, into the future. This can be seen in the following equation:

$$u(k) = \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix}, \quad x(k) = \begin{bmatrix} x(k+1|k) \\ x(k+2|k) \\ \vdots \\ x(k+N|k) \end{bmatrix} \tag{4-2}$$

Here $u(k+1|k)$ and $x(k+1|k)$ are the input and state vectors at time $k+1$ that are predicted at time k. $x(k+i|k)$ evolves according to:

$$x(k+i+1|k) = Ax(k+i|k) + Bu(k+i|k), \quad i = 0, 1, \ldots \tag{4-3}$$

with initial condition defined

$$x(k|k) = x(k) \tag{4-4}$$

Optimization

The feedback law used is found by minimizing a cost function. This performance cost can be defined in terms of future values for x and u.

$$J(k) = \sum_{i=0}^{N} [x^T(k+i|k)Qx(k+i|k) + u^T(k+i|k)Ru(k+i|k)] \tag{4-5}$$

with Q and R positive definite matrices.

$$u^*(k) = \arg\min_u J(k) \tag{4-6}$$

$u^*(k)$ is the optimal input sequence for minimizing $J(k)$ and any input or state constraints should also be included as equivalent constraints on $u(k)$.

Receding horizon implementation

It is important to note that although a sequence of optimal inputs is computed, only the first element of $u^*(k)$ is input to the system.

$$u(k) = u^*(k|k) \tag{4-7}$$

This process of constantly generating an input sequence $u^*(k)$ by minimizing the cost function and only implementing the first element is repeated at each timestep. The optimization of $u^*(k)$ is an on-line optimization and the length of the prediction horizon does not change, this approach is known as the receding horizon strategy. This mechanism introduces feedback into the MPC law, Since the state prediction and optimal input sequence depend on the current state $x(k)$. The result is a degree of robustness against uncertainty and modelling errors (Chmielewski & Manousiouthakis, 1996).

4-2-2 Dynamic Inversion

Dynamic Inversion, also known as feedback linearisation, is a structured way to cancel the dynamics and then control the system as a linear system. It involves formulating a transformation of the non-linear system into an equivalent linear system through a change of variables and a suitable control input (Isidori, 1995). Given the following system:

$$\dot{x} = F(x, u) \tag{4-8}$$
$$y = H(x) \tag{4-9}$$

with x the state vector, u the control vector and y the output vector. For conventional uses, the function F can be considered linear in u, resulting in:

$$\dot{x} = f(x) + g(x)u \tag{4-10}$$

with f a non-linear state dynamic function and g a non-linear control distribution function. This can be written in companion form:

$$\begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} x_2 \\ \vdots \\ x_n \\ b(x) \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a(x) \end{bmatrix} \tag{4-11}$$

In other words, all non-linear terms only affect x_n. Also, the input only affect x_n. The virtual control input v is then defined as:

$$v = b(x) + a(x)u \Leftrightarrow u = a^{-1}(x)[v - b(x)] \tag{4-12}$$

This virtual control input v can be used to control the entire non-linear system in a simple linear way. Although the basics of DI are simple, it needs to be noted that it is assumed that $g(x)$ is invertible for all values of x, which is not always true. Also even if $g(x)$ is invertible, the control inputs u can become large and this growth is dangerous for actuator saturation. Dynamic Inversion requires exact knowledge of the model dynamics to achieve good performance, therefore it is normally used as an inner-loop controller in combination with an outer-loop controller that has been designed using other techniques to have robustness.

4-2-3 Model Reference Adaptive Control (MRAC)

The general idea of MRAC is to create a closed loop controller with parameters that can be updated to change the response of the system. The output of the system is compared to a desired response from a reference model. The control parameters are updated based on the error perceived. The goal is for the parameters to converge to ideal values that cause the plant response to match the response of the reference model. MRAC is often used as a final stage

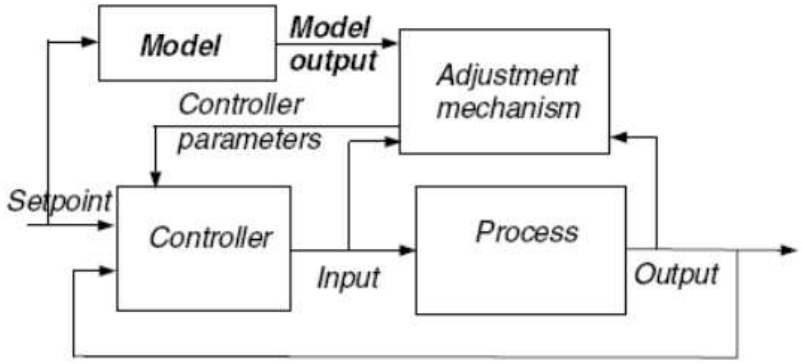


Figure 4-2: Block diagram for the MRAC Approach (Butler, 1991)

in other algorithms and is proven to be effective for many types of structural failures (Butler, Honderd, & Van Amerongen, 1991),(Sadeghzadeh, Mehta, Zhang, & Rabbath, 2011).

Consider a linear plant of the form found in Equation 4-13:

$$\begin{aligned} \dot{x} &= Ax + Bu + d \\ y &= Cx \end{aligned} \tag{4-13}$$

$$\dot{y}_d = A_d y_d + B_d r \tag{4-14}$$

$$u = C_0 r + G_0 x + v \tag{4-15}$$

The closed loop dynamics are found to be:

$$\dot{y} = (CA + CBG_0)x + CBC_0 r + CBv + Cd \tag{4-16}$$

The ultimate objective is for the closed loop dynamics to match the desired dynamics. Had the model be known exactly, the controller parameters C_0 G_0 and v could just simply be computed. However because a failure has occurred the model is not known exactly and the controller parameters need to be adapted (Kreisselmeier & Anderson, 1986).

Indirect Adaptation

$$\begin{aligned} C_0 &= (C\hat{B})^{-1} B_d \\ G_0 &= (C\hat{B})^{-1}(A_d C - C\hat{A}) \\ v &= (C\hat{B})^{-1}(Cd) \end{aligned} \tag{4-17}$$

The idea of identifying the model on-line and then computing a control law assuming the estimated model is perfect is common practise in the reconfigurable control literature, as is with EA and MMAE.

Direct Adaptation

The direct adaptive control tries to directly estimate the controller parameters C_0, G_0 and v, instead of first computing model parameters. These estimated values are defined as the values which will force the plant to track the reference model.

Adaptive control requires the system parameters to change slowly enough for the estimation algorithm to track them, however faults can be abrupt. There is no guarantee the system will be stable during this transient, which limits the use of adaptive control. Adaptive control used on its own is not yet enough to handle general faults, but it can be used as part of a bigger reconfigurable control system.

4-2-4 Model Following (MF)

The Model Following (MF) method is another form of active Fault-Tolerant Control (FTC). It is a state-space design methodology by which a control system is designed to make the system follow the output of a model with the desired behaviour. In this approach the design objectives are incorporated into the reference model, and by using a reference model difficulty in control system design is avoided (Zhang & Jiang, 2002). Basically the method considers a system and a reference model of the same form. Let the reference model be

$$\begin{aligned} x_{k+1}^M &= A_M x_k^M + B_M r_k^M \\ y_k^M &= C_M x_k^M \end{aligned} \tag{4-18}$$

and the system be represented by

$$\begin{aligned} x_{k+1} &= A x_k + B u_k \\ y_k &= C x_k \end{aligned} \tag{4-19}$$

where x_k^M, $x_k \in \mathbb{R}^n$, u_k^M, $u_k \in \mathbb{R}^m$, A_M, $A \in \mathbb{R}^{n \times n}$, B_M, $B \in \mathbb{R}^{n \times m}$, C_M, $C \in \mathbb{R}^{p \times n}$.. The corresponding transfer function matrices of the reference model and system are:

$$T_M(s) = C_M(sI - A_M)^{-1} B_M \tag{4-20}$$

$$P(s) = C(sI - A)^{-1} B \tag{4-21}$$

Let e_k represent the difference in state variables

$$e_k = x_k^M - x_k \tag{4-22}$$

In order to achieve perfect model following, it must hold that for any u_k^M and $e_0 = 0$ the error $e_k = 0$ for $k > 0$. However perfect model following is not always possible, in this case the objective is to keep the error e as small as possible.

The model following method has the advantage that it usually does not require a Fault Detection & Isolation (FDI) scheme. However a strong disadvantage is that this method is not applicable to sensor faults and also does not take model uncertainties into account (Gao & Antsaklis, 1992).

4-2-5 Pseudo-Inverse (PI)

The Pseudo Inverse (PI) method is one of the popular active methods to FTC due to its computational simplicity and its ability to handle a large spectrum of system faults (Gao & Antsaklis, 1990). The main objective is to maintain as much similarity as possible to the original design and provide graceful degradation in performance.

First the pseudo-inverse of matrix is discussed. Let the singular value decomposition of $A \in \mathbb{R}^{n\times m}$ be the following

$$A = U \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix} V^H \tag{4-23}$$

where V and U are orthogonal matrices. For any unitary matrix W, we have $W^H W = I$ and $||W||_2 = 1$. W^H is the complex conjugate of W. The pseudo-inverse of A is now defined as

$$A^\dagger = V \begin{pmatrix} \Sigma^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^H \tag{4-24}$$

The basic PI considers the following linear system

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{4-25}$$

for $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{p\times n}$. The nominal closed-loop system is designed with linear state-feedback control law $u_k = Kx_k$, $K \in \mathbb{R}^{m\times n}$, under the assumption that the state vector is available for measurement.

$$\begin{aligned} x_{k+1} &= (A + BK)x_k \\ y_k &= Cx_k \end{aligned} \tag{4-26}$$

where K is the state feedback gain. This method allows for a very general post-fault representation

$$\begin{aligned} x^f_{k+1} &= A_f x^f_k + B_f u^R_k \\ y^f_k &= C_f x^f_k \end{aligned} \tag{4-27}$$

where the new, reconfigured control law is taken with the same structure, $u^R_k = K_R x^f_k$. The new closed-loop system is

$$\begin{aligned} x^f_{k+1} &= (A_f + B_f K_R)x^f_k \\ y^f_k &= C_f x^f_k \end{aligned} \tag{4-28}$$

with K_R the new feedback gain to be determined. The goal is to find the new state-feedback gain matrix K_R in such a way that the distance between the A-matrices of the nominal and the post-fault closed-loop systems is minimized

$$\begin{aligned} K_R &= \arg\min_{K_R} \left\| (A + BF) - (A_f + B_f K_R) \right\|_F \\ &= B_f^\dagger (A + BF - A_f) \end{aligned} \tag{4-29}$$

where $B_f^{\dagger}$ is the pseudo-inverse of the matrix B_f. The advantages of this approach is that it is very suitable for on-line implementation due to its simplicity. It also allows for changes in all state-space matrices of the system and thus can be used for on-line accommodation for unanticipated failures. The main drawback of the PI is that the stability of the reconfigured system is not guaranteed. Therefore a new method with guaranteed stability is proposed, it is based on the PI and is called the modified pseudo-inverse method (Gao & Antsaklis, 1991).

4-2-6 Linear Quadratic

Linear Quadratic Regulator (LQR) design tries to use the least amount of control energy to make the output of the system to equilibrium condition/set-point - so called regulation problem. Under the Linear Quadratic (LQ) design goal, the controller is designed minimizing the following criterion:

$$J = \int_0^{\infty} ||x(t)|| + \rho||u(t)||dt \tag{4-30}$$

where ρ is a positive constant. The norm $||x(t)||$ corresponds to the energy of the system and $||u(t)||$ corresponds to the energy of the control signal. The role of ρ is to balance the trade-off between system and control input (Yu, Zhang, Yan, Qu, & Liu, 2013).

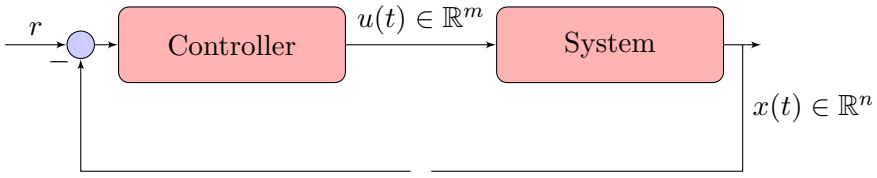


Figure 4-3: Linear Quadratic Control with state feedback

Consider the following system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{4-31}$$

and accompanying it the following objective function:

$$J_{LQR} = \frac{1}{2}\int_0^{\infty} x(t)'Qx(t) + u(t)'Ru(t)dt \tag{4-32}$$

where Q and R are positive semi-definite matrix and positive definite matrix with corresponding dimensions (Yang, Wang, & Soh, 1999). A first choice for the matrices Q and R can be found by using Bryson's rule. The LQR controller design can be now formulated in terms of a Hamiltonian:

$$H = x^T Qx + u^T Ru + \lambda^T(Ax + Bu) \tag{4-33}$$

where λ is the vector of Lagrangian multipliers. The solution is obtained from the following Partial Differential Equation (PDE)

$$\dot{\lambda} = -\frac{\delta H}{\delta x} = -A^T\lambda - Qx = 0,\ \lambda(\infty) = 0 \tag{4-34}$$

$$\dot{X} = \frac{\delta H}{\delta \lambda} = Ax + Bu = 0,\ \lambda(0) = 0 \tag{4-35}$$

$$\dot{U} = \frac{\delta H}{\delta u} = Ru + B^T\lambda = 0 \tag{4-36}$$

$$\frac{\delta H}{\delta x^2} = Q \geq 0 \tag{4-37}$$

From the above equations it can obtain the optimal control solution. To find the state feedback control law, the state vector x is assumed to be measurable. The control law is $u = Kx$, where K is to be found from the following Ricatti equation (Looze, Weiss, Eterno, & Barrett, 1985):

$$A^TP + PA - PBR^{-1}BT + Q = 0 \tag{4-38}$$

where $K = R^{-1}B^TP$, $Q \geq 0$, $Q = V^TV$ and $R > 0$, $R = R^T$. If $\{A, B\}$ is controllable and $\{C, A\}$ is observable, then the solution is definite; and with $k \geq 0.5$ (Hespanha, 2009):

$$\mathfrak{Re}\{\lambda(A - kBR^{-1}B^TP)\} < 0 \tag{4-39}$$

The previous was LQR under fault-free conditions. If the following cost function is considered for a system under fault conditions

$$J = \int_0^\infty e^{2at}(x^TQx + u^TRu)dt \tag{4-40}$$

$$(A + \alpha I_n)^TP + P(A + \alpha I_n) - PBR^{-1}BT + Q = 0 \tag{4-41}$$

where $\alpha > 0$ in case of a fault. All other conditions are the same as in Equation 4-38. Consequently the following holds:

$$\mathfrak{Re}\{\lambda(A - kBR^{-1}B^TP)\} < -\alpha \tag{4-42}$$

Now recall that $K = R^{-1}B^TP$, which means the value of K depends on R. R is a design parameter and all fault scenarios need to be considered when designed controller K. When considering actuator faults, Equation 4-31 can be rewritten as follows:

$$\begin{aligned} \dot{x} &= Ax + BL_iu \\ y &= Cx \end{aligned} \tag{4-43}$$

Now using LQR technique, the following can be obtained:

$$\mathfrak{Re}\{\lambda(A - BL_iK)\} < -\alpha \tag{4-44}$$

The parameter α can now be used to simulate loss of effectiveness in different fault scenarios.

4-3 Pre-computed Control Laws

This section will deal with pre-computed control laws used as a reconfigurable control technique. Different fault scenarios are considered off-line and a control law to stabilize each fault is designed beforehand. Several techniques, such as Gain Scheduling (Section 4-3-2) and Multiple Model (Section 4-3-3), are discussed below.

4-3-1 Generalized Internal Model Control

The Generalized Internal Model Control (GIMC) is a feedback controller which allows to separately design for robustness and performance. This is advantageous because in traditional feedback control there is usually a conflict between these two (Zhou & Ren, 2001). To start, the general derivation of GIMC is quickly shown. Consider the standard feedback configuration as show in Figure 4-4. Here P is a Linear Time Invariant (LTI) plant and K is a LTI controller.

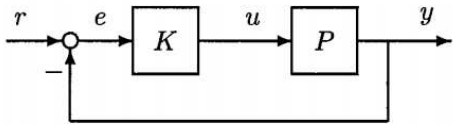


Figure 4-4: Standard Feedback Configuration (Zhou, 2001)

The feedback controller K can be also represented in the way shown by Figure 4-5. Note that $K = (\tilde{V} - Q\tilde{N})^{-1}(\tilde{U} + Q\tilde{M})$ and representation as a total transfer function by one block in Figure 4-4 or by 5 blocks in Figure 4-5 which for a fixed Q is essentially the same. In standard Youla controller parametrization, $\tilde{U}$, $\tilde{V}$, U and V are chosen so that $\tilde{U}N + \tilde{V}M = I$ and $\tilde{N}U + \tilde{M}V = I$ (Kučera, 2011).

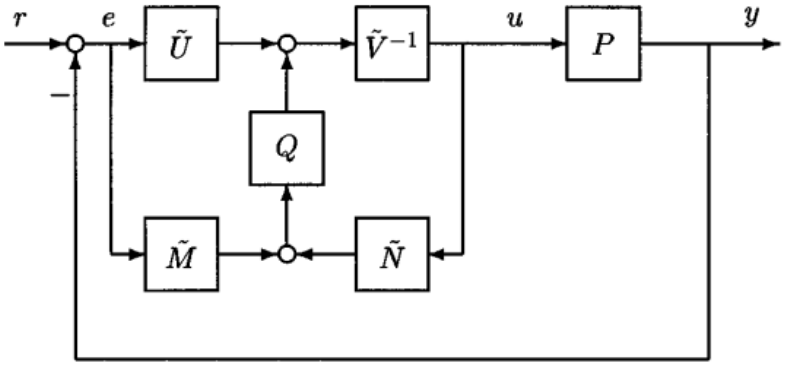


Figure 4-5: Youla Controller Parametrization (Zhou, 2001)

The model P is usually not perfectly known and as such a nominal model P_0 is known. Now assume K_0 is a stabilizing controller for P_0 and they have the following coprime factorizations:

$$K_0 = UV^{-1} = \tilde{V}^{-1}\tilde{U} \qquad P_0 = NM^{-1} = \tilde{M}^{-1}\tilde{N} \tag{4-45}$$

Then every stabilizing controller for P_0 can be written as previously stated:

$$K = (\tilde{V} - Q\tilde{N})^{-1}(\tilde{U} + Q\tilde{M}) \tag{4-46}$$

for some $Q \in H_\infty$ such that $det(V(\infty) - N(\infty)Q(\infty)) \neq 0$ (Cui, Zhang, & Yang, 2014). Another way to implement the controller $K = (\tilde{V} - Q\tilde{N})^{-1}(\tilde{U} + Q\tilde{M})$ can be seen in Figure 4-6, where the position of $\tilde{M}$ has changed. Nevertheless the internal stability of the system did not change because the transfer function from y to u still remains unchanged. This controller implementation also stabilizes internally the feedback system with plant P_0.

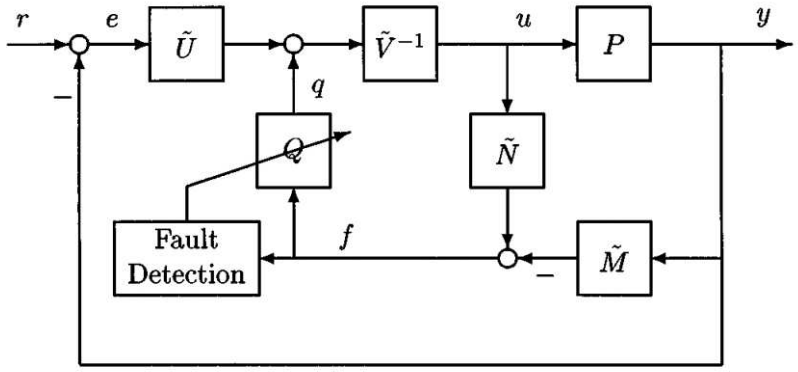


Figure 4-6: Fault-Tolerant GIMC (Zhou, 2001)

This controller framework is called GIMC and the distinguished feature of this controller implementation is that the inner loop feedback signal f as seen in Figure 4-6 is in fact the residual signal used in fault diagnosis. This signal is always zero if the plant model is perfect. The inner loop is only active when there is a model uncertainty, disturbances, sensor noise or a fault present in the system. So Q can be designed to robustify the feedback system.

A high performance robust system can be designed using GIMC in two steps:

1. Design $K_0 = \tilde{V}^{-1}\tilde{U}$ to satisfy system performance specifications by assuming no faults.
2. Design Q to satisfy system robustness requirements. This controller will not affect nominal system performance.

This method works as long as Q is allowed to be any non-linear time varying stable system. It is also possible to design a Q for different fault scenarios and switch when a fault is detected through residual f.

4-3-2 Gain Scheduling

Gain scheduling is one of the most popular and well known approaches to non-linear control that has been widely applied successfully in various fields. It is characterised by a divide and conquer strategy in that it decomposes the non-linear control design task into a number of linear subtasks. This strategy is the main reason of its popularity since it enables the well established linear methods to be applied to non-linear methods. The analysis of non-linear system remains relatively difficult, whilst the techniques for analysing linear time-invariant system have been better developed. One or more observable variables, the scheduling variables, are used to determine what operating region the system currently is in and the linear controller that belongs to it. For example, for a UAV the attitude and airspeed might be the scheduling variables, with different linear controllers for the hover and extreme angle regions.

4-3-3 Multiple Model

Multiple Model (MM) is another active approach to FTC that is not an on-line redesign method but rather a pre-computed based method. The MM method used a set of linear

models M_i, $i = 1, 2, ..., N$, where each model hypothesises a different fault scenario that describes the system in different operating conditions. For all of these models M_i a controller C_i has to be designed off-line.

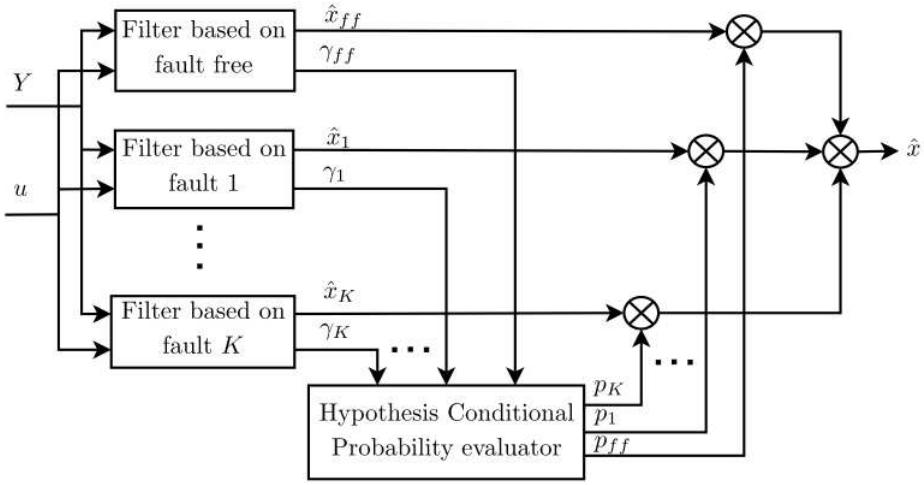


Figure 4-7: Block diagram for the MMAE Approach (Lu, 2014)

When using a MM approach the important design factor is to create an on-line decision maker which constructs the overall control action through a weighted combination of different control actions that need to be taken according to each model. In most literature the control action weighting is found using a bank of Kalman Filters (Ducard & Geering, 2008), each Kalman Filter is designed for one of the local models M_i. Using the residuals of the Kalman filter, the probability μ_i of each model to be in effect, is computed. The control action is then found as the weighted combination as can be seen in Equation 4-47. The basic idea is that the Kalman Filter which produces the most well-behaved innovation contains the model which matches the true faulty model best.

$$u(k) = \sum_{i=1}^{N} \mu_i(k) u_i(k), \qquad \sum_{i=1}^{N} \mu_i = 1 \tag{4-47}$$

where $u_i(k)$ is the proposed control which the i-th local model controller suggests.

The MM approach is an attractive way to model the control of nonlinear systems. Normally there are only a finite number of models, and thus a finite number of faults can be accounted for. This way if one of these faults is encountered, its corresponding model will be in effect with weight μ_i close to unity and all other weights close to zero. In this case only one controller is 'active'. One disadvantage is that model uncertainties cannot be considered and might lead to a wrong model in effect. Another downside of only having one model in effect is that if the current model is not included in the set of predesigned fault models a combination of existing models will be used to control the system, which is not optimal anymore and can lead to instability. To avoid dealing with unmodelled, or unanticipated, faults a bank of predictive controllers often put into place. These controllers form the global control action in such a manner that the optimal control action for the current model is used (Kanev, 2004).

There are 2 types of MM methods: Multiple Model Switching and Tuning MMST and Interacting Multiple Model IMM, both of which will be shortly covered below. In both methods all possible occuring faults are predetermined through an extensive analysis of the system and fault models are constructed to provide stability for each fault. When a fault happens, the MMST method switches to another model with a pre-computed control law that matches to the current fault. Instead of switching to a single model that matches the current fault the best, IMM constructs a new fault model by using a linear combination of the complete set of fault models.

Multiple Model Switching and Tuning (MMST)

When using Multiple Model Switching and Tuning each fault that can occur is formulated its corresponding model. All models are placed in parallel, they all have their controller that stabilises for that particular fault. The problem now arises in choosing which model to use at what time. A performance index is used to evaluate which model most closely matches the current system. This can be a cost function mapping of the deviation between actual and estimated output. A commonly used performance index (Narendra & Balakrishnan, 1997):

$$J_i(t) = \alpha e_i^2(t) + \beta \int_0^t e^{-\lambda(t-\tau)} e_i^2(\tau) d\tau \tag{4-48}$$

where α and β are parameters that determine the ratio between of instantaneous and long-term accuracy measures. The model with the lowest performance index is that model to be used for a certain time period. This time period is put into place to make sure the controller does not switch between models very fast. Multiple model switching has a proven stability if the amount of models provided is sufficiently detailed (Narendra & Balakrishnan, 1997).

When systems only have a few failure modes, MMST can be a very good candidate due to its speed and stability. The problem arises when there is a fault scenario that has not been accounted for or when a big systems need to be controlled. For bigger system the models used for MMST increase rapidly. (Edwards, Lombaerts, & Smaili, 2010).

Interacting Multiple Model (IMM)

The main idea of IMM solve the limitations encountered to using the switching strategy, which implies that every fault needs to be modelled. IMM allows for mixing between the models that were formerly only switched between:

$$M_f = \sum_{i=1}^{N} \mu_i M_i = \mu^T \begin{bmatrix} M_1 \\ \vdots \\ M_N \end{bmatrix}, \quad M_i \in \mathcal{M} \tag{4-49}$$

Detection of faults and choosing how to mix the established models is done by finding μ_i. This can be done in two ways. One way is by using a temporal approach, looking at different instants and at the estimate at this instant, a minimal value for μ is found which represents the model mixing. Another method is by using a Kalman Filter (KF) for each model, and

evaluating the probability that each of the models represent the real system. By combining the probability that each model represents the actual system, which allows for calculating a value for μ a model mixing can be obtained.

IMM is a clever method that can handle different fault scenarios by combining failure models. However since after fault detection the μ coefficients need to be found, this makes it lose some of the speed the MMST has.

4-4 Conclusion

In this chapter many different options for reconfigurable controllers have been discussed. Depending on the implementation some control strategies are more suitable than others. During the literature study it became apparent that it was more interesting to investigate a smart controller selection for aggressive horizontal manoeuvring of a quadrotor. A conventional quadrotor controller is not able to cope with aggressive manoeuvres and will lose height when attempting to track these extreme angles. A new controller has been designed that keeps tracking altitude by changing the angle of attack, this controller is more accurate in vertical tracking while degrading horizontal and yaw tracking. This controller should only be activated when it will outperform the conventional controller, e.g. only during the aggressive manoeuvres. The conventional controller is superior in the hover regime, so adaptive control is needed depending on the state of the quadrotor. The next chapter will deal with reinforcement learning, which will be the method of adaptive control and learning the controller selection intelligently.

Chapter 5

Reinforcement Learning

A reinforcement learning controller is an adaptive controller that can constantly learn and alter its behaviour to keep performance and safety at an acceptable level. Since it became apparent after the research into reconfigurable controllers that an adaptive controller using reinforcement learning is the most interesting approach this chapter will deal with Reinforcement learning (RL), an area of machine learning. Reinforcement learning is characterised by trial-and-error search and cumulative reward. First a general introduction of reinforcement learning is given. Secondly the Markov Decision Process (MDP) and some of the popular methods used are explained. Finally a motivation of how this could prove to be useful for high performance control is presented.

5-1 Agent-Environment Interface

The learning environment considered in a RL system consists of two main components, the learning agent and the environment. During each time step the agent observes the environment and takes action which alters the state, the agent receives a reward to indicate how well it is performing the required task. The agent's goal is to minimize a cost function or equivalently maximize a reward function. In other words the agent must learn the conditions that will lead to a reward or punishment and map states to actions. This is fundamentally different to supervised learning (Haykin, 1998), where the reward is given beforehand during a training procedure. In RL experience is the only teacher, it is based on a feedback mechanism which makes the agents autonomous.

The above can be visualized as can be seen in Figure 5-1. The basic RL model consists of the following (Sutton & Barto, 1998):

1. A set of environment states S
2. A set of actions A
3. Rules of transitioning between states which is usually stochastic

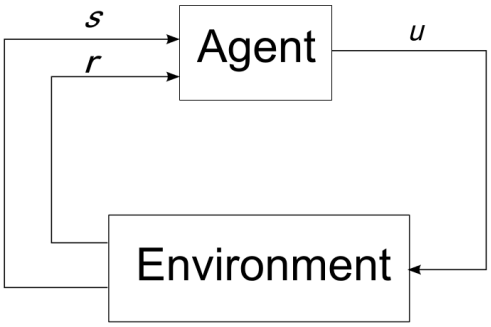


Figure 5-1: Interaction between agent and the environment in reinforcement learning

4. Rules that determine the immediate reward R of a transition
5. Rules that describe what the agent observes

At each time step, the agent implements a state action mapping, which is called the agent's policy π. RL methods can alter the agents policy according to its experience to achieve maximum total reward over a period of time.

5-2 Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical framework which greatly helps to model decision making and this framework is useful for studying optimization problems solved using RL.

The Markovian condition states that any observation $\mathbf{x}$ made by the agent must be a function only of its last observation, its last action and some disturbance. That is, given the present, the future does not depend on the past:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, u_t, w_t) \tag{5-1}$$

with $\mathbf{x}_t$ the observation, u_t the action taken and w_t the disturbance. Another way of illustrating the MDP dynamics can be seen in Figure 5-2:

$$\mathbf{x}_0 \xrightarrow{u_0} \mathbf{x}_1 \xrightarrow{u_1} \mathbf{x}_2 \xrightarrow{u_2} \mathbf{x}_3 \xrightarrow{u_3} \dots$$

Figure 5-2: MDP dynamics

The other case, when observations made by the agent are insufficient to fully determine all information about the state, a non-Markovion condition takes place:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \dots, u_t, u_{t-1}, u_{t-2}, \dots, w_t) \tag{5-2}$$

It is possible to transform this non-Markovian case into a Markovian one by using information vectors that summarize the available information (Striebel, 1965), however the computational power requirements increase rapidly.

5-2-1 Continuous state MDP

Since many real-world problems are not discrete, but rather continuous, an infinite number of states will exist for the MDP. For example a quadrotor, the state can be represented as $(x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$. So $S = \mathbb{R}^{12}$, which is an infinite set of states, because there is an infinite number of possible positions and orientations for the quadrotor. The simplest way to solve the problem of infinite states is to discretize the state space, however there are also some downsides. The first downside is that it assumes the value function is constant over the discritization interval, which for most smooth function is not a good representation. The second downside of discritizing is known as the curse of dimensionality. Suppose $S = \mathbb{R}^n$ and the n dimensions are discritized into k values, the total number of discrete states is k^n. This number grows exponentially in the state space, and does not work well for big problems.

5-3 Policy, Reward and Value Function

After each action is taken the agent receives a reward R which is a measure of performance. It is important to note that the goal of the agent is to try and maximize the total cumulative reward, which can be defined as:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \tag{5-3}$$

where the discount factor $0 \leq \gamma \leq 1$ forces rewards nearby to be more important than rewards in the far future. Since there can be many possibilities starting from an initial state, actions are chosen over time to maximize the expected reward:

$$\mathrm{E}[\, R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \,] \tag{5-4}$$

As mentioned earlier the policy π is a function that maps states s to actions a. So for an agent with policy π the action in state s will be $a = \pi(s)$ (Russell & Norvig, 1995). There are a number of policies such as:

- Random: Randomly select an action
- Greedy: Select the action with maximum immediate reward
- ϵ-Greedy: Random action ϵ times, otherwise action with maximum immediate reward.

The state-value function $V^\pi(s)$ for a given policy π is the expected reward starting in state s and following policy π:

$$V^\pi(s) = \mathrm{E}\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid s_t = s \Big] \tag{5-5}$$

By following the optimal policy the agent is able to maximize this state-value function. Another way of expressing 5-5 is by taking Bellman's Optimality Equation, which gives a recursive definition of the state-value function $V(s)$:

$$V^\pi(s) = \underbrace{R(s)}_{\text{immediate reward}} + \underbrace{\gamma V^\pi(s+1)}_{\text{future discounted rewards}} \tag{5-6}$$

For any RL problem it is very important to define the reward function in a correct way, because the RL algorithm will learn using the rewards it gets. If the reward function is poorly constructed this will resonate through to the behaviour of the RL algorithm and performance will not be optimal.

An extension of representing a value function is by using an action-value function $Q^{\pi}(s,a)$, which is defined as taking action a in state s and following optimal policy π^* after this:

$$Q^{\pi}(s,a) = \mathrm{E}\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid s_t = s, a_t = a \Big] \tag{5-7}$$

Finally some relations that are useful for arriving at the optimal policy are given. The optimal state-value function is reached by acting through the optimal policy:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{5-8}$$

The same holds for the action-value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{5-9}$$

The interrelation between the state-value and action-value function can clearly be seen in the following Equation:

$$V^*(s) = \max_{a} Q^{\pi^*}(s,a) \tag{5-10}$$

Finally arriving at the optimal policy π^* which are the actions that lead to maximum $V^*(s)$ as can be seen by the arg max operator.

$$\pi^*(s) = \arg\max_{a} Q^{\pi^*}(s,a) \tag{5-11}$$

5-4 Solving a MDP problem

Now that a way of expressing the value function has been established and the total reward obtained under a certain policy can be calculated, how can the optimal policy be found? This can be done using Dynamic Programming (DP), which requires complete knowledge about state transition probabilities and rewards. Two methods will be discussed in the sections below.

5-4-1 Value Iteration method

The idea behind value iteration is simple: if the true value of each state is known, the decision would simply be the action that gives the highest reward. However as a starting point the state's true value is not known, as only the immediate reward is known. This can make a difference for states that have a low reward on its own, but is on the path to a high reward state (Bellman, 1957).

Value iteration starts at the end and from here works backward:

1. Assign each state a random value
2. For each state, calculate its new $V(s)$ based on its neighbour's utilities.
3. Update each state's $V(s)$ based on: $V_{i+1}(s) = R(s) + \gamma \max_a V_i(s+1)$
4. If no value changes by more than ϵ, stop

Value iteration is guaranteed to converge to the optimal solutions, but can be slow. Convergence occurs out from goals, information about shortcuts propagates from where the reward is.

5-4-2 Policy Iteration method

Value iteration has a few downsides, it can take a long time to converge and it calculates the value of each state to extract the optimal policy. Policy iteration directly finds the optimal policy. It starts with an arbitrary policy and iteratively improves it:

$$\pi_1 \rightarrow V^{\pi_1} \rightarrow \pi_2 \rightarrow V^{\pi_2} \rightarrow \dots \pi^* \rightarrow V^* \rightarrow \pi^* \tag{5-12}$$

Given a starting policy π_1 with value function V^{π_1} the policy can be improved to yield a better policy π_2, now V^{π_2} is found and the policy can be improved again (Howard, 1960).

The policy iteration algorithm is as follows:

1. Initialize an arbitrary policy by selection random actions at each state
2. While not done:
   - Compute $V(s)$ for each state given the currently policy
   - Update state values
   - Given these new values, select the optimal action for each state.
3. If no action changes, stop

5-5 Reinforcement Learning Methods

Solving MDP problems using Dynamic Programming requires a model of the process, because the iteration methods use the state transition probabilities explicitly. To overcome this limitation Monte Carlo methods can be used, but these require complete runs before any updates can be performed. This section will discuss basic RL algorithms which allow model-free and online updating as the agent is learning the optimal policy (Sutton & Barto, 1998).

5-5-1 Temporal Difference Method

Temporal Difference (TD) learning is a combination of Monte Carlo ideas and dynamic programming ideas. These methods learn estimates based on other estimates, which is called bootstrapping. If value functions need to be calculated without estimation, the agent would need to wait until the final reward before all state-action pairs can be updated. Unlike Dynamic Programming methods TD methods can learn directly from raw experience without a model. Using TD methods an estimate of the final reward is calculated at each state and the state-action value is updated at every time step:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{estimate of return}} - V(s_t)\Big] \tag{5-13}$$

This method is known as TD(0) and can be fully implemented in an online fashion. Note that a state-value function is used as opposed to an action-value function which is used in the next 2 sections.

5-5-2 SARSA

State-Action-Reward-State-Action (SARSA) is an on-policy TD control algorithm. The main function to update that Q-value is dependent on, according to its name: the current state and action taken, the reward for choosing this action, the next state the agent is in and finally the next action the agent will choose in this new state:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\substack{\text{learning}\\ \text{rate}}}\Big[\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}}\underbrace{Q(s_{t+1}, a_{t+1})}_{\substack{\text{estimate of future}\\ \text{reward with policy } \pi}} - \underbrace{Q(s_t, a_t)}_{\text{old value}}\Big] \tag{5-14}$$

As can be seen from Equation 5-14 the learning rate α will determine to what extent the newly acquired information will override the current information.

5-5-3 Q-Learning

Q-learning is one of the most important breakthroughs in RL and is an off-policy TD control algorithm. Every state is assigned an estimated value, a Q-value. When a state gets visited and the agent receives a reward, the Q-value is updated according to the following:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\substack{\text{learning}\\ \text{rate}}}\Big[\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}}\underbrace{\max_a Q(s_{t+1}, a)}_{\substack{\text{estimate of}\\ \text{optimal future value}}} - \underbrace{Q(s_t, a_t)}_{\text{old value}}\Big] \tag{5-15}$$

In this case, the action-value function Q which is learned, directly approximates the optimal action-value function Q^*, independent of the policy that is being followed. The policy still determines which state-action pairs are visited and updated, however the only requirement for convergence is that all pairs continue to be updated.

5-5-4 Eligibility Traces

Eligibility traces are a basic mechanism used throughout reinforcement learning. In the TD(λ) algorithm the λ refers to the use of an eligibility trace. These traces can be seen as a temporary record of the occurrence of an event, such taking an action and visiting a state. When a TD error occurs the states and actions that have caused this error can easily be found through their eligibility. TD(λ) can be seen as the bridge between TD(0) which only uses a single state transition and Monte Carlo which uses a whole episode of transition data.

5-5-5 The Exploration/Exploitation Trade-off

In order for RL algorithm to find the optimal action a complete exploration is necessary. This is however infeasible in practical situations. When learning and control are both at stake, the agent must try to find a good balance between exploration of alternatives to a given policy which might lead to an improvement in policy and exploitation of the current policy that will never lead to improvement (Bertsekas, 1995). RL methods usually include a stochastic action selector that will allow for exploration of the state space. A common method is to use a Gibbs or Boltzmann distribution, where to probability of exploring slowly decays and in the end the best action is always chosen.

5-6 Example: Windy Grid World

In order to illustrate the general working principle of reinforcement learning, a simple problem has been formulated. The problem is known as the "Windy Grid World" problem and can be seen in Figure 5-3

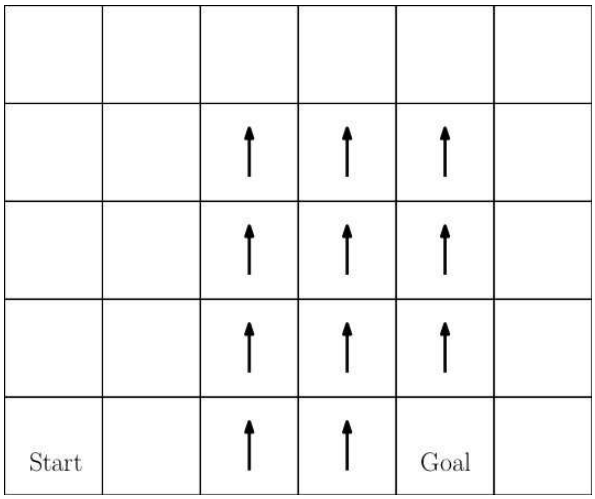


Figure 5-3: Windy Grid World

Each cell in Figure 5-3 is a state and has four possible actions: move up, down, left and right. This is a deterministic domain, each action moves the agent to one cell in the direction chosen. If the agent is on the boundary of the Windy Grid World an action that would move it outside of the world will keep the agent in the same cell from which it tried the action.

Note that in certain cells there are arrows, symbolizing a "windy" state. If the agent ends up in these states it experiences an extra push upward. For example, if the agent is in one of these states and wants to move left or right, it will also move one cell up.

This is an episodic task where the agent has no more than 30 time steps. At the beginning of an episode the agent is placed in the "Start" state. Reward in the grid world is zero for every state except the "Goal" state. The agent receives a reward of 10 when it executes any action from the goal state. The episode ends after 30 time steps or when the agent takes any action after having arrived in the goal state.

The goal of the agent is to find the shortest path from the "Start" state to the "Goal" state, in other words the actions that will result in the highest reward. The general strategy to find the optimal policy can be seen in Figure 5-4

- Initialize $Q(s, a)$ arbitrarily and $Q(\text{terminal-state}, -) = 0$
- Repeat (for each episode)
  - Initialize state s
  - Repeat (for each step of episode)
    - Choose action a from state s using policy derived from Q (for example ϵ-greedy)
    - Take action a, observe reward R, next state s'
    - update $Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max_a Q(s', a) - Q(s, a)]$
    - update $s \leftarrow s'$
  - until s is terminal

Figure 5-4: Q-Learning: Off-policy TD control algorithm (Sutton & Barto, 1998)

For the Windy Grid World an ϵ-greedy strategy is used with an $\epsilon = 0.1$ so that the agent will explore a random action 10% of the time, while otherwise taking the best known action. The learning rate α determines to what extent the newly acquired information will override the old information. Since the Grid World is a deterministic world a value of $\alpha = 1$ is used, the same action will always result in the same reward. However for stochastic problems, which most real-world problems are, a more suitable value would be $\alpha = 0.1$. Using the latter in this example still causes converge, it will only just take longer to learn the optimal path. The discount rate γ determines the importance of future rewards, since our agent should reach the goal state as soon as possible the discount factor should be less than 1. Table 5-1 summarizes the values used in the Windy Grid World problem:

α	1.0
γ	0.9
ϵ	0.1

Table 5-1: Q-Learning Parameters

When first initializing the agent in the environment and starting with an empty Q-matrix (all zeros), the agent does not know anything yet and will randomly walk around the grid until it reaches the Goal state for the first time. This will update the Q-matrix and from here on out

the Q-matrix will slowly start to get filled. At some point the Q-matrix converges, meaning that it does not change more than a certain threshold between updates. Once the Q-matrix has converged the most optimal path to the goal state has been found. Tracking this sequence of states now is as simple as taking the action with highest Q-value in each state.

The solution to the shortest path in the Windy Grid World can be seen in Figure 5-5, this is exactly what was to be expected.

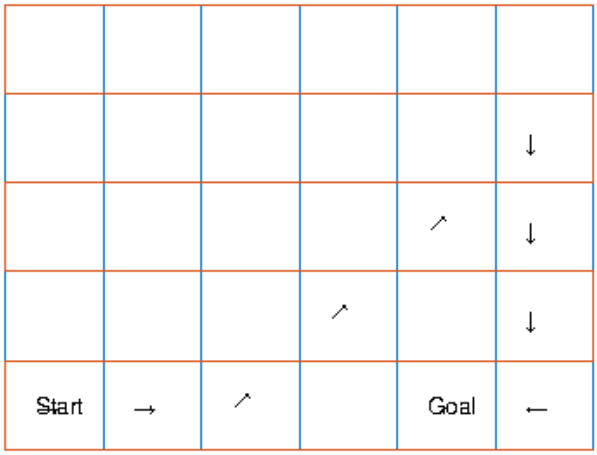


Figure 5-5: Windy Grid World Solution - after 315 epochs

As demonstrated in this section, Q-learning is a method to find the optimal policy through experience. The agent starts with no prior knowledge of which action to take in each state and learns through trial and error. After having sufficiently searched the state-space the Q-matrix will converge and by following the maximum Q-values in each state the agent has learned the best strategy for the problem it has to solve.

5-7 Conclusion

Reinforcement Learning can be used as a form of adaptive control that learns the best behaviour to optimise reward or performance. There are several strategies that can be used such as Temporal Difference, SARSA and Q-learning, however out of these strategies Q-learning seems the most promising. This is because Q-learning is an off-policy method. This means that regardless of the policy followed the optimal action-value function Q is approximated. Although real-world problems are continuous there are several techniques ways to handle this, however the problem can quickly end up with a very large state-space. This makes it difficult not only computationally wise, but also state visitation wise. The agent must visit a state and take an action before it can determine if the action it took was a better alternative than the best known action up to that point.

As seen in the Windy Grid World example reinforcement learning learns by trial and error and needs some time to reach sufficient performance. This needs to be taken into account when applying this concept to a UAV. The UAV needs to be sufficiently stable before learning

starts and the exploration of the agent should be done in a safe way in order for the UAV the remain airborne. One way to accomplish this is to first run the experiment in a simulation environment and gain enough experience off-line before implementing this onto the UAV and continue learning on-line. By continuing to learn on-line the UAV can continue to optimize its performance against certain unexpected failures and external conditions.

Bibliography

Bangura, M., & Mahony, R. (2012). Nonlinear dynamic modeling for high performance control of a quadrotor. *Proceedings of Australasian Conference on Robotics and Automation*.

Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics 6*.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1) (No. 2). Athena Scientific Belmont, MA.

Butler, H., Honderd, G., & Van Amerongen, J. (1991). Model reference adaptive control of a gantry crane scale model. *Control Systems, IEEE*, *11*(1), 57–62.

Cannon, M. (2015). *Oxford lecture slides on model predictive control.*

Chmielewski, D., & Manousiouthakis, V. (1996, November). On constrained infinite-time linear quadratic optimal control. *Syst. Control Lett.*, *29*(3), 121–129.

Cui, L., Zhang, Y., & Yang, Y. (2014). Youla parameterization based adaptive controller design for fault tolerant control. *Asian Journal of Control*, *16*(1), 292–295.

Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, *58*, 1–35.

Ducard, G., & Geering, H. P. (2008). Efficient nonlinear actuator fault detection and isolation system for unmanned aerial vehicles. *Journal of Guidance, Control, and Dynamics*, *31*(1), 225–237.

Edwards, C., Lombaerts, T., & Smaili, H. (2010). *Fault tolerant flight control.* Springer.

Gao, Z., & Antsaklis, P. (1991). On the stability of the pseudo-inverse method for reconfigurable control systems. *Proceedings of the IEEE National Aerospace and Electronics Conference*.

Gao, Z., & Antsaklis, P. J. (1990). Pseudo-inverse methods for reconfigurable control with guaranteed stability. In *the 11th ifac world congress.*

Gao, Z., & Antsaklis, P. J. (1992). *Reconfigurable control system design via perfect model following* (Vol. 56) (No. 4).

Haykin, S. (1998). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Hespanha, J. P. (2009). *Linear systems theory.* Princeton University Press.

Hoffmann, G. M., Huang, H., Waslander, S. L., & Tomlin, C. J. (2011). Precision flight

control for a multi-vehicle quadrotor helicopter testbed. *Control Engineering Practice*, *19*(9), 1023–1036.

Hoffmann, G. M., Waslander, S. L., & Tomlin, C. J. (2006). Distributed cooperative search using information-theoretic costs for particle filters with quadrotor applications. *In Proceedings of the AIAA guidance, navigation, and control conference*.

Howard, R. A. (1960). *Dynamic programming and markov processes*. Cambridge, MA, USA: Technology Press-Wiley.

Isidori, A. (1995). *Nonlinear control systems*. Springer Verlag, London.

Kanev, S. (2004). *Robust fault-tolerant control*. PhD thesis, University of Twente.

Keating, M., Pachter, M., & Houpis, C. (1997). Fault tolerant flight control system: QFT design. *International Journal of Robust and Non-Linear Control*, *7*, 551–559.

Kreisselmeier, G., & Anderson, B. D. (1986). Robust model reference adaptive control. *Automatic Control, IEEE Transactions on*, *31*(2), 127–133.

Kučera, V. (2011). A method to teach the parameterization of all stabilizing controllers. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, *18*, 6355–6360.

Looze, D., Weiss, J., Eterno, J., & Barrett, N. (1985). An automatic redesign approach for restructurable control systems. *IEEE Control Systems Magazine*, *5*.

Marzat, J. e. a. (2012). Model-based fault diagnosis for aerospace systems: a survey. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, *226*, 1329–1360.

Mulder, J. A., van Staveren, W. H. J. J., van der Vaart, J. C., & de Weerdt, E. (2007). *ae3-302 flight dynamics lecture notes*. Delft University of Technology, Faculty of Aerospace Engineering.

Narendra, K., & Balakrishnan, J. (1997). Adaptive control using multiple models. *IEEE Transactions on Automatic Control*, *42*(2), 171–187.

Parrot. (2015). *Ar.drone 2.0. parrot new wi-fi quadricopter*. Available from `http://ardrone2.parrot.com/`

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Sadeghzadeh, I., Mehta, A., Zhang, Y., & Rabbath, C.-A. (2011). Fault-tolerant trajectory tracking control of a quadrotor helicopter using gain-scheduled pid and model reference adaptive control. In *Annual conference of the prognostics and health management society* (Vol. 2).

Striebel, C. (1965). Sufficient statistics in the optimum control of stochastic systems. *Journal of Mathematical Analysis and Applications*, *12*(3), 576–592.

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning* (1st ed.). Cambridge, MA, USA: MIT Press.

Veillette, R., Medanic, J., & Perkins, W. (1992). Design of reliable control systems. *IEEE Transactions on Automatic Control*, *37*(3), 290–304.

Yang, G.-H. Y. G.-H., Wang, J. L. W. J. L., & Soh, Y. C. S. Y. C. (1999). Reliable LQG control with sensor failures. *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*, *4*.

Yu, B., Zhang, Y., Yan, J., Qu, Y., & Liu, Z. (2013). Fault tolerant control using linear quadratic technique against actuator faults in a UAV. *Control Conference (CCC), ...*, 6294–6299.

Zhang, Y., & Jiang, J. (2002). Active fault-tolerant control system against partial actuator failures. *IEE proceedings-Control Theory and applications*.

Zhou, K., & Ren, Z. (2001). A new controller architecture for high performance, robust, and fault-tolerant control. *IEEE Transactions on Automatic Control*, *46*(10), 1613–1618.