Design and Analysis of Gain Scheduled Robust Control Systems in Applied Robotics

Dissertation submitted to the

Department of Engineering and Mathematics

Sheffield Hallam University

in partial fulfilment of the award of the degree of
MSc Automation, Control and Robotics

by

Thomas Pile

June 2018 - August 2018

Supervisor: Dr Lyuba Alboul

Abstract

This dissertation investigates the topic of LPV robust control in the context of quadrotor control as an example of less widely applied control techniques. The aim is to compare LPV to more widely applied robust control techniques and establish whether the additional cost and complexity is justified. A mixed mode quadrotor is used as the testbed. The topic is introduced and the background theory of robust and LPV control is presented in the opening chapters. The model and a loopshaped H-infinity robust controller are developed and analysed against uncertainty and then Simulated. Later the Coprime Factorised model which is the basis of loopshaped robust control is found for an LPV model of the quadrotor and robust controllers with and without pole placement constraints are analysed. The dissertation closes with a discussion of implementation issues, conclusions and how LPV techniques can be better applied within robotics.

Contents

Nomenclature

ARE	Algebraic Riccati Equation
BIBO	Bounded Input Bounded Output
COTS	Commercial Off-The-Shelf
EKF	Extended Kalman Filter
FLAVIIR	Flapless Air Vehicle Integrated Industrial Research
GCARE	Generalised Control Algebraic Riccati Equation
GFARE	Generalised Filter Algebraic Riccati Equation
IMU	Inertial Measurement Unit
LMI	Linear Matrix Inequality
LPV	Linear Parameter Varying
LQG	Linear Quadratic Gaussian (Controller)
LQR	Linear Quadratic Regulator (Controller)
LSDP	Loop Shaping Design Procedure
LTR	Loop Transfer Recovery
MRAC	Model Reference Adaptive Control
NCF	Normalised Coprime Factorisation (McFarlane-Glover Method)
PDLF	Parameter Dependent Lyapunov Function
PID	Proportional Integral Derivative (Controller)
PPC	Pole Placement Constraints
SISO	Single Input Single Output
SQLF	Single Quadratic Lyapunov Function

List of Figures

Chapter 1

Introduction

This dissertation concerns the development of several robust control algorithms for a small UAV type aircraft. First a $\mathcal{H}_\infty$ (H-Infinity) controller is developed for a static coprime factorised model, and then families of controllers are developed for a coprime factorised model of an LPV plant using LMI methods. Secondary aims were to implement and test these algorithms in system, but getting these projects into the air is often quite difficult in constrained timescales.

Control for aircraft has been very well covered in the literature, but this is typically done with less advanced algorithms. The aim of this project is to apply progressively more specific robust control to the problem of aircraft control. The use of Linear Parameter Varying coprime factorised models with $\mathcal{H}_\infty$ robust control with represents a very specific application of this and has seen comparatively little application. The primary objective in such studies is to further applied control theory, rather than to improve control of aircraft.

UAV projects are often very difficult to make work, particularly at MSc and group MEng level, the main causes of this are technical problems and insufficient time. The technical problems can involve software problems, obtaining low noise estimates of UAV angles via EKFs, communication problems with the microcontroller's I^2C and SPI buses, and the generation of control signals for the motors.

Structure of the Dissertation

In chapter 2 the literature review introduces some of the work that has taken place in the field of UAV flight control with a view to avoiding the mistakes of others, adopting some of the good ideas, and as a guide as to where my research efforts will be most productive. Most of the review is dedicated to highlighting the state of the art of robust control in flight control applications.

Chapter 3 introduces the control theory used later in the dissertation. This starts with an introduction to robust control theory, robust stability conditions, the $\mathcal{H}_\infty$ norm,

uncertainty representations, and linear fractional transformations. Section 3.2 covers the coprime factorisation approach to $\mathcal{H}_\infty$ control which is relevant to chapter 5. Section 3.3 covers the coprime factorisation for linear parameter varying systems used in chapter 6. Finally section 3.4 covers additional control methods referred to in the literature review.

Chapter 4 describes the kinematics of the quadrotor system and the interpretation of the state space model linearised at an equilibrium point. The system identification to obtain coefficients for the state space model is also contained in this chapter.

Chapter 5 describes the $\mathcal{H}_\infty$ controller development using coprime factorised models, as well as an analysis of the resulting controllers. Linear and nonlinear simulations are performed to give further confidence in the validity of the controllers performance.

Chapter 6 covers the application of coprime factorisation $\mathcal{H}_\infty$ control to LPV systems. Chapter 5 focuses on controllers intended for practical use, chapter 6 is more concerned with working with leading edge control techniques and so there is less analysis of the controller in the context of a specific application, and more about the synthesis technique itself. Significant detail is given of the process of synthesising the controller, and the application of Linear Matrix Inequalities (LMI) to solve the convex optimisation of the $\mathcal{H}_2$ problem. An example controller is developed for the model from chapter 5.

1.1 Research Questions

1. How applicable are robust control techniques to robotic applications. What are future directions of research at the interface of control systems and robotics.

2. How well suited are Linear Parameter Varying robust control techniques for control of quadrotor aircraft. Are the benefits of parameter varying control justified by the increased complexity.

3. How do control synthesis techniques for LPV models compare? With particular reference to static non-varying controllers, SQLF based, and PDLF based controllers.

Chapter 2

Literature Review

Although there has been a dramatic increase in the number of small UAV type flying vehicles, most of these use quite simple control methods like PID control. This can be assumed to be due to a law of diminishing returns as ever more complex algorithms deliver marginal improvements over simpler methods, yet require dramatically more investment. Eventually a level is reached where control algorithms are so difficult to develop that they are only implemented in very high budget industrial programmes or in academic projects. The choice is not binary, there is a wide range of control methods which can be applied, these include: PID, pole placement, Linear Quadratic control [1] (LQR, LQG, LTR), fault tolerant control [20], feedback linearisation and iterative backstepping [2], $\mathcal{H}_\infty$ control [3] with loop shaping [4] or mixed synthesis, with static controllers and parameter varying controllers. Various vision based methods also exist including horizon sensing (infrared), visual horizon detection, and detection of markers in structured environments. Strictly speaking most of the visual methods are not control methods, but are means of sensing attitude which is then used for control instead of an IMU. An exception to this is optic flow control methods. Of the non-trivial control methods, choice of control algorithm is strongly in influenced by the aircraft type. For vertical take-off tri/quad/hex-rotors fully non-linear controllers are often used such as backstepping and dynamic inversion, and for fixed wing vehicles extensions to the state space approach are more often used such as LQR and $\mathcal{H}_\infty$. Pole placement and PID have a significant advantage in that their implementation in system is simpler than approaches such as $\mathcal{H}_\infty$.

Much of this dissertation refers to $\mathcal{H}_\infty$ loopshaping (LSDP), and Linear Parameter Varying Loopshaping (LPV-LSDP), and other forms of robust control, which are often applied to difficult control problems like nano-positioning [5], tokomaks [6], multi-arm inverted pendulums [7], artillery shells [8] and of course flight control. An early example of the application of $\mathcal{H}_\infty$ to flight control is the VAAC Harrier [9] project, a vertical take-off aircraft at QinetiQ/DERA (Defence Evaluation Research Agency), UK MoD.

Application of Robust Control to Aircraft

This section seeks to evaluate the application of $\mathcal{H}_\infty$ robust control to real world flight control problems. Of particular interest is to (1) determine the levels of complexity that can be involved, and (2) what advantages were found in each case, and (3) what trends can be identified. It is difficult to compare the performance of the controllers between projects in terms of responses as they involve different aircraft models. The projects have been selected as they apply the concepts of robust control to aircraft but in different ways. These include a basic robust controller for a desk based model [10], a quadrotor with LSDP and nonlinear control [21], an LPV $\mathcal{H}_\infty$ controller for a fixed wing aircraft [24], and an LMI LPV based $\mathcal{H}_\infty$ coprime factorisation controller for a fixed wing aircraft [15].

Chen, Z

Chen, Z [10] at the University of Sheffield carried out a similar project with the use of standard mixed synthesis and later loopshaping, but applied to a table-top helicopter type model that is used for the module ACS6110. The dissertation outlined the details of the mathematical model of the helicopter and some of the theoretical background for how controller synthesis works. The model was also improved by modelling a friction coefficient. During the project it was found that mixed synthesis does not produce good results without weights included, and that a rank error occurred when applying mixed synthesis to the plant. LSDP was then used as an alternative which was a good decision as it can handle integrators by factorising the plant as two stable factors M^{-1} and N.

From the project by Chen [10] an obvious conclusion is that the hinfmix() function is likely to be unsuitable for synthesising controllers for a quadrotor model. Analysing this further the problem seems to be that the stability of poles on the complex axis which is undefined. An example of the issue Chen [10] had is provided in Appendix A. It is also possible that Chen [10] could have confused the McFarlane-Glover "loopshaping" method with the Matlab $\mathcal{H}_\infty$ loop shaping method which shapes loop transfer functions such as S, T and KS.

An alternative is to use LQG which is suitable for such a model as was shown in the project Hurley [11], a student at the same University using the same aircraft model. Hurley obtained obtained good results with this approach, but also attempted unsuccessfully to implement mixed synthesis $\mathcal{H}_\infty$ control method. The main weakness of the LQG method is that is that it is not a robust control technique. In terms of complexity the $\mathcal{H}_\infty$ methods applied are approximately medium, but the LQG method is much simpler, assuming in both cases that Matlab commands are used to synthesise the controllers. The lower complexity of LQG is due to it being a pole placement method and hence easier to implement in a real system.

Chen [10] cited a paper on weight optimisation [12] which deals with quite a complex

weight optimisation algorithm for LSDP although it is not a weight selection tutorial, but the PhD thesis [13] on which the paper is based would have been more useful as a guide.

Kara Mohammed

The topic of robust Trirotor control has been examined in two related works at the University of Manchester, the first by Kara Mohammed [21, 22] and the second by Hu [23]. The objective of Kara's thesis [21] was to control a Trirotor aircraft with actuator rotated motor/propeller mounts, using $\mathcal{H}_\infty$ loopshaping and iterative backstepping for the non-linear dynamics of the thrust actuators. Iterative backstepping uses feedback to cancel the non-linear parts of the system by substituting "virtual controls". In Kara's case the state space system resulted in a double integrator system, again forcing the use of the loop shaping design procedure with coprime factorised models (LSDP). The control system is summarised in the block diagram in Fig: 2.1 [21].

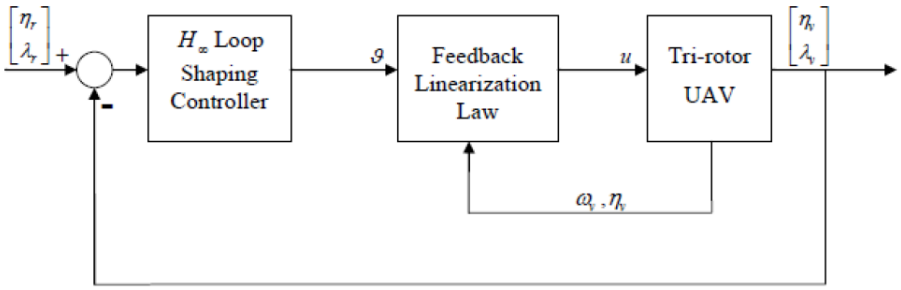


Figure 2.1: Kara Feedback Linearisation Diagram

The formulas for the iterative backstepping controller represent a valuable contribution in that they can be reused in the general case, and required significant work in their original derivation. Hardware problems prevented the system from being tested so in-system results were not given but simulated results were. The position measurement was somewhat unconventional with lasers projected onto the ground and a camera used to derive altitude, and landmarks from feature recognition used to determine position. The lasers used were obtained from COTS laser rangefinders, although the range measurements were not used, they were just used as lasers. The project by Kara was further developed in Hu [23] to be used in distributed formation control. For the control component the same control methods and formulas appear to have been used, and no further in-system tests were performed, but the system was run through simulation again.

As this is the first advanced control project to be considered the complexity is quite high. This is due to the nonlinear control techniques used more so than the robust loop-shaping techniques which are of medium complexity. The vision element is also of high complexity, and also quite an interesting development, but probably not as relevant to my project. An alternative perhaps would have been to use lasers with a cross shaped pattern

as these are widely used in construction. The system had to be used on flat ground, but again a cross shaped pattern would have less limitations in this regard as the shape of the light can be used to estimate the geometry of the ground in local regions and the altitude estimates corrected accordingly.

Another conclusion that can be drawn is the difficulty in getting UAV projects to complete testing due to technical issues with hardware which is trend with such projects. It is also shown that the feedback linearisation can be used to improve stability of the aircraft by including otherwise unmodelled actuator components. Restricting this to just the actuators instead of applying it to the full aircraft model also saves on computing requirements and makes the mathematics simpler. It is my view that while these are definite advantage, the lack of thrust vectoring on my proposed aircraft model means that I am not able to make use of the technique. It would also make the application of LPV techniques much more difficult.

The Trirotor used during the project is shown in Fig: 2.2 [21].

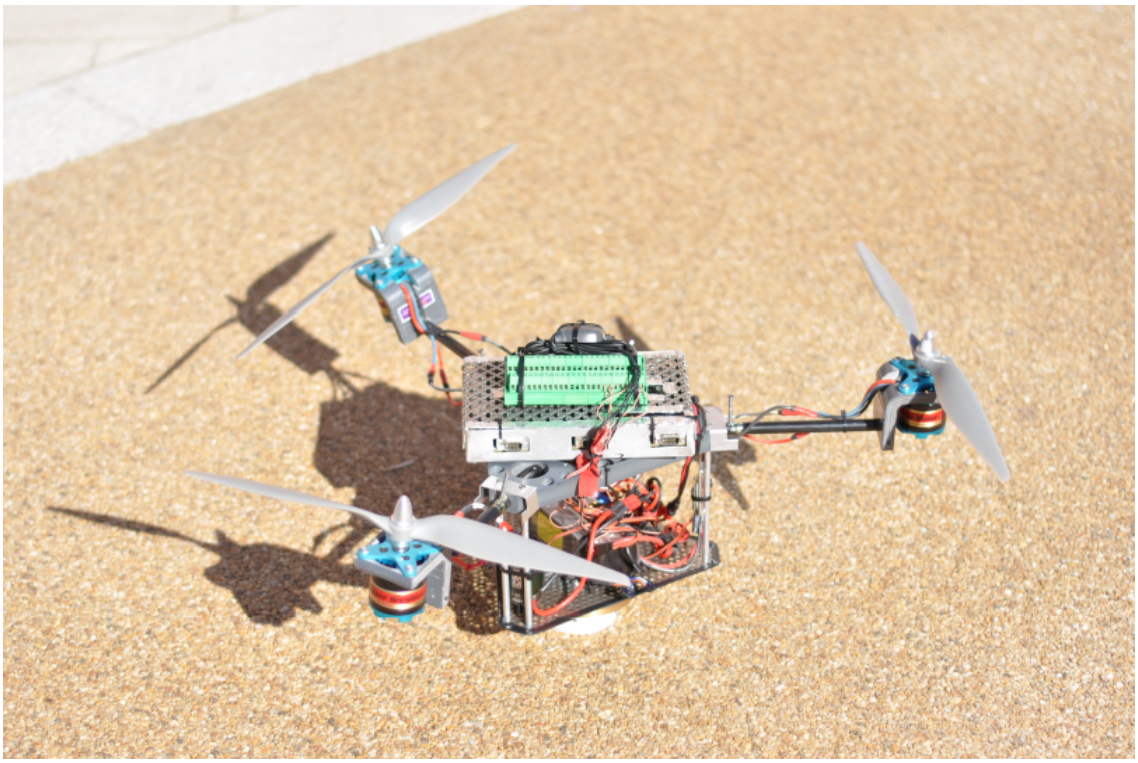


Figure 2.2: Trirotor UAV

Chumalee

Chumalee [24] applied LPV $\mathcal{H}_\infty$ control to the longitudinal and lateral directional models of a fixed wing target aircraft [for surface to air missiles]. The purpose of the project was for the analysis of the control algorithms, not for the production of an aircraft. In the

work extensive derivations were made of the longitudinal and lateral-directional models of the aircraft, and these were used to create LPV models using the Jacobian and Tensor-Product methods, and analysis of the different methods was performed. A $\mathcal{H}_\infty$ mixed synthesis controller was produced for the LPV model, with simulations and flight tests performed on a Jindivik test aircraft. Improved conditions for LPV systems were also found which enabled synthesis of the controller using LMIs with fewer decision variables and fewer matrix inequality constraints. The equations for these are very involved and so are not repeated here.

The complexity of this method of obtaining a controller is quite high if the models have to be derived, but Matlab now has functions such as hinfgs() to perform synthesis for similar classes of model which makes the complexity less. It is shown that SQLF are conservative compared to PDLFs [42] when applied to LPV models with slowly varying parameter, though the advantage of SQLFs being simpler is likely to outweigh the conservatism. It is proposed that the method is a good candidate for future UAV flight control, but lack of performance is not a major weakness of SQLF, complexity of implementation is, and this method is quite complex to implement. It is a good candidate for flight control, but only for aircraft with very strict performance requirements. It is also very useful as an example of how LPV models interact with fixed wing dynamics, but this part is not particularly relevant to my project.

FLAVIIR / Chen

The UK government along with BAE systems ran a joint academic and industrial programme to build a fixed wing UAV called FLAVIIR [14], a flap-less aircraft which uses ducted air bled from the engines, fed from the trailing edge of the wing to control the roll attitude of the aircraft using aerodynamic phenomena known as the Coanda effect. The control system for the FLAVIIR was developed at the University of Leicester by a team including Ian Postlethwaite, Da-wei Gu, Jianchi Chen and Kenan Natesan. The two theses that came from this are Chen [16, 15] and Natesan [17], each designing an LPV controller in a different manner. Natesan used a technique called mu-synthesis [19, 18], and Chen used both a standard signal based method and a coprime factorisation method.

In Chen linear parameter varying models of the plant were developed from a separate system identification of the aircraft. The LPV models were used with $\mathcal{H}_\infty$ methods for controller synthesis, in one case with a signal based synthesis model, and in another case with a coprime factorisation of the model for robust loopshaping. Parameter Dependent Lyapunov Functions (PDLF) were used for proof of stability in the development of the LPV controllers because they reduce the conservativeness of the Lyapunov functions by considering the rate at which the parameters change. The LPV model was scheduled on velocity. The robust loopshaping element is of particular interest here. The approach taken

in synthesis has two main steps. The first is to obtain the coprime factorisation by solving a generalised $\mathcal{H}_2$ problem using Linear Matrix Inequalities (LMI) based on the algorithm by Prempain [25]. The coprime factorisation was not of the complete LPV model, instead gridding was used to perform the coprime factorisation at discrete velocities. The second step is to synthesise the $\mathcal{H}_\infty$ controller with the PDLF, again using LMI methods.

This is a very difficult way to obtain a $\mathcal{H}_\infty$ controller for an aircraft due the complexity of LMI methods, and the lack of any built in functions in Matlab for the combination of LPV, coprime factorisation and $\mathcal{H}_\infty$ in control. The complexity also increases the possibility of software flaws compared to simpler controllers. The advantage is that, at least theoretically, the controller should be less conservative. Pole placement constraints were successfully used to eliminate the high frequency poles produced by $\mathcal{H}_\infty$ synthesis which makes the controller more accurate to the one synthesised when implemented. What is particularly impressive about this project is that it involved the application of bleeding edge control algorithms to an aircraft of substantial size and complexity. Successful flights tests were carried out on the aircraft.

The Eclipse UAV on which FLAVIIR is based is shown in Fig: 2.3 [15].

Figure 2.3: Eclipse UAV

Final conclusions and analysis

One trend in the more advanced applications such as [15] and [24] is towards the use of LMI methods, and hence a shift away from ARE based approaches. This seems to be due to faster computation speeds particularly with larger models, the potential to apply multiple constraints such as pole placement and $\mathcal{H}_2$ optimality conditions, and to work with LPV models. Another trend in the two projects is towards the use of PDLFs for stability analysis, although it is my opinion that the complexity of them somewhat limits them to larger aircraft, particularly as the parameter rates are slower and so benefits from decreased conservatism could be more significant. Small UAV aircraft such as the one proposed in my project are likely to have varying parameters that switch at higher rates

and so the conservatism advantages of PDLF over SQLF will be less, but at the same time the complexity of PDLF over SQLF will remain.

In Masters dissertations there is a trend towards using LQG control, or robust control methods with internal Matlab functions. An area for improvement in this case would be to actually test the controllers against some uncertain models, not to just rely on the γ optimality value. The advantage of this is that it would better illustrate the robustness properties by showing the effect on other metrics like frequency and time responses, which are not evident from γ values.

Two potential gaps in the literature have been identified from this review. The first is the potential to apply feedback linearisation to part of the model such as actuators or other nonlinearity, and to apply LPV control to the main part of the system. It is possible that the two approaches could complement each other, with LPV addressing slow variations, and feedback linearisation addressing faster and more significant nonlinearities. The effectiveness of this is unknown. A second potential gap in the literature is the development of a continuous, as opposed to gridded coprime factorisation of the LPV model. This is unlikely to be a practical problem to solve as the coprime factorised LPV model would also have to remain convex.

It is proposed to use the $\mathcal{H}_\infty$ loopshaping design procedure with a coprime factorised model (LSDP) as the basis for the dissertation as this has been shown to be successful and achievable in the dissertation of Chen [10], and is the most likely method to successfully control the aircraft. This can then be built upon if time permits with Linear Parameter Varying LSDP as in the case of Chen [15] and Chumalee [24]. Another key conclusion that can be drawn is the consistent underestimation of the difficulty of implementing the final control in hardware, either due to unexpected technical problems of under-estimation of the required time. Such problems are expected.

Chapter 3

Theory

This chapter begins by defining what robust control using loopshaping (LSDP) is, and how it is applied. This includes the use of the normalised coprime factorised form of uncertainty representation and how this is used with $\mathcal{H}_\infty$ methods to obtain a robust controller. It then moves to the topic of Linear Parameter Varying systems in the context of Normalised Coprime Factor Synthesis (LPV-NCFS).

3.1 Robust Control

This subsection gives an introduction to some key parts of robust control, but is not intended to be a complete overview. Robust control is the design of control systems which maintain their stability properties when the plant frequency response or parameters are varying. This can also be extended to provide robust performance by representing the performance as an uncertainty. Stability can be considered as either open loop (uncontrolled) or closed loop (with a controller), but in the context of robust control stability is taken to refer to the closed loop. Nominal stability is the most basic level of stability where a nominal model in closed loop is stable. Robust stability is where it is nominally stable and also stable against the modelled uncertainties. A controller that is nominally unstable, but robustified would just lead to a robustly unstable controller. Robust controllers are also required to meet some performance requirements, this is nominal performance. If the performance is to maintained in the face of uncertainties then this is robust performance.

In robust control a nominal model is developed, and then the uncertainty is included as an additional description. The two main methods for this are as component uncertainty such as additive or multiplicative, or as a generalised uncertainty as used in coprime factorisation. In all cases robustness is established as a measurable quality in terms of an infinitive norm which is to be minimised.

Robust control uses weighting matrices to include design specifications, this can be

done either with signal based methods [3, Chapter 16] [30, p.362] or with loopshaping. In the loopshaping case the weights are connected in series with the plant. In the signal based case disturbance input (w) and disturbance output (z) signals are series connected with frequency domain weights. Depending on the locations in the loop at which the signals are placed the S, T, and KS functions can be weighted. The signal based (mixed synthesis) is the most flexible and powerful method, but also depends on a lot on knowing the gains and cutoff frequencies for the weights. A more complete overview of robust control is given in Skogestad [30], and more advanced theoretical treatment can be found in Zhou, Glover and Doyle [3].

3.1.1 Uncertainty

There are two methods of representing uncertainty, one is to use perturbations [7, P13] at the input or output, the other is to use coprime factorisation. The perturbation approach can be used with either real parametric uncertainty or lumped frequency bounded uncertainty. The principles are similar for both multivariable and scalar systems, but in the multivariable case structured singular values have to be used to account for directionality.

Considering first frequency bounded uncertainty, this is uncertainty in the gain of a transfer function of the system at different frequencies, and so can be represented as a ball of uncertainty along the Nyquist plot in the SISO case. This is modelled as a perturbation which is additive or multiplicative on some input or output signal, and so can also be represented in terms of a perturbation on a transfer function.

Additive

Additive noise can be represented as in Eq. 3.1.

$$P = P_0 + W_1 \cdot \triangle \cdot W_2 : \quad \|\triangle\|_\infty < 1 \tag{3.1}$$

This can be read as the perturbed plant, the left hand side, is equal to the nominal plant P_0 plus the multiplication of a perturbation $\triangle$, or δ in the scalar case, by two weights W_1 and W_2. The condition $\|\triangle\|_\infty < 1$ requires the uncertainty to be norm bounded to 1. This corresponds to minimising the weighted KS transfer function, with KS being the transfer function from the reference, disturbance and noise inputs to the control output.

$$\|W_2 K S W_1\|_\infty < \gamma \tag{3.2}$$

S and KS are defined as in Eq. 3.3 and Eq. 3.4.

$$S = (I + GK)^{-1} \tag{3.3}$$

$$KS = K(I + GK)^{-1} \tag{3.4}$$

Multiplicative

Multiplicative uncertainty represents a multiplicative perturbation on a signal and hence is bounded. This is typically used to represent unmodelled high frequency dynamics. Output multiplicative uncertainty is defined as in Eq. 3.5.

$$P = (W_1 \cdot \triangle \cdot W_2)P_0 \quad \|\triangle\|_\infty < 1 \tag{3.5}$$

Multiplicative uncertainty can also be represented at the input, and can be moved to the output by a process called "reflection". This corresponds to minimising the weighted complementary sensitivity function T as per Eq. 3.6.

$$\|W_2 T W_1\|_\infty < \gamma \tag{3.6}$$

T is the transfer function from the noise and reference inputs to the system output as defined in Eq. 3.7.

$$T = GK(I + GK)^{-1} \tag{3.7}$$

In both cases the perturbation $\triangle$ is bounded to 1, which means that during simulation and analysis 1 corresponds to 100% of system uncertainty, 0.2 to 20% and so on. The multivariable case differs from the scalar case if the perturbation is not diagonal as this corresponds to coupling between the channels.

Real / Parametric

Parametric uncertainty is a variance in a component of the model, such as a spring or damper value. This variance obviously causes an uncertainty in the frequency response of the system as well, but the representations are not equivalent as the uncertainty is not necessarily the full ball. Often in analysis such a parametric uncertainty is approximated by a more conservative frequency bounded ball for simplicity.

3.1.2 Stability Conditions and Norms

There are several different definitions of system stability with varying strengths of conditions. BIBO stability, Bounded Input Bounded Output, is the most basic form of stability where for a bounded input the transfer function will have a bounded output, but it has some limitations. This type of uncertainty, without perturbations, is known as nominal stability. Asymptotic stability assumes a zero input and non-zero initial conditions for the state (equilibrium), the system being stable if the system goes to zero from that state.

Exponential stability is a stronger form of asymptotic stability, imposing a minimum decay rate on the system. Internal stability is verified by checking all four transfer functions from disturbance and control inputs to measurement and control outputs for stability. If there are no pole zero cancellations between the controller and the system then stability of one transfer function is enough to verify internal stability. The stability of the transfer function can be established using the small gain theorem (generalised Nyquist criterion [30, p.152]). The four transfer functions are defined in Eq. 3.8.

$$\begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} (I+KG)^{-1} & K(I+GK)^{-1} \\ G(I+GK)^{-1} & (I+GK)^{-1} \end{bmatrix} \begin{bmatrix} du \\ dy \end{bmatrix} \tag{3.8}$$

$$\begin{bmatrix} S & KS \\ T & S \end{bmatrix} \tag{3.9}$$

S, T and KS are again defined as Eq. 3.10 to Eq. 3.12.

$$S = I + D^T D \tag{3.10}$$

$$T = GK(I+GK)^{-1} \tag{3.11}$$

$$KS = K(I+GK)^{-1} \tag{3.12}$$

For robust control a key stability condition is that of the small gain theorem. The small gain theorem states that a closed loop feedback system M is quadratically stable if the plant is open loop stable and the loop transfer function has a gain magnitude of less than 1 over all frequency as per Eq. 3.13. This is equivalent to the condition that the induced $\mathcal{L}_2$ norm from $w \to z$ being bounded by γ. This is a generalisation of the Nyquist stability criterion which states that the Nyquist curve must be within the unit circle at all frequencies. The stability margin is defined as the distance to instability.

$$\|M\|_\infty < 1 \; \forall\omega \tag{3.13}$$

The magnitude of signals in control is measured through the use of norms, the most important of these in the context of robust control is the $\mathcal{H}_\infty$ norm, which gives the peak gain over frequency. A common application is in defining robustness requirements as it represents the worst case perturbation about an equilibrium, but it can also be applied other problems like minimising peak actuator usage to reflect actuator limits. The $\mathcal{H}_\infty$ norm of a system is the maximum singular value of the system in the standard case, which comes from the principle that the peak Eigenvalue is the maximum gain. The $\mathcal{H}_\infty$ norm

is defined as Eq. 3.14.

$$\|G\|_\infty = \sup_{\omega \in \mathbb{R}} \|G\|_\infty \tag{3.14}$$

In the time domain the $\mathcal{H}_\infty$ norm is defined as the induced $\mathcal{L}_2$ norm. the $\mathcal{L}_2$ norm defined as [28, p295] in Eq. 3.15.

$$\mathcal{L}_2 = \left(\int_{-\infty}^{\infty} z(t)^T z(t) dt \right)^{0.5} \tag{3.15}$$

This is useful in that it illustrates how the $\mathcal{H}_\infty$ norm is a gain of the system, though in practice the $\mathcal{H}_\infty$ norm is computed in frequency domain. The definition states that if a system input and output are energy bounded ($w, z \in \mathcal{L}_2$) then the system is BIBO stable by the small gain theorem [26]. The $\mathcal{H}_\infty$ norm can then be defined in the time domain as an induced norm from the output to the input.

$$\|G\|_\infty = \sup_u \frac{\|Gu\|_{\mathcal{L}_2}}{\|u\|_{\mathcal{L}_2}} = \sup_{s \in Re^+} \overline{\sigma}(G(s)) \tag{3.16}$$

The following is a further summary of the four main stability conditions that are encountered [30]

Nominal Stability $\Rightarrow$N is internally stable

Nominal Performance $\Rightarrow \|N_{22}\|_\infty < 1$ and system nominally stable

Robust Stability $\Rightarrow F_u(N, \triangle)$ stable $\forall \triangle$ for $\|\triangle\|_\infty < 1$ and system nominally stable

Robust Performance $\Rightarrow \|F_u(N, \triangle)\|_\infty < 1,\ \forall \triangle,\ \|\triangle\|_\infty < 1$ and system nominally stable

with N the lower LFT of the model and controller.

3.1.3 Linear Fractional Transforms

Linear Fractional Transforms were originally developed by Redheffer [27] and provide a way to express a control problem involving many different inputs and outputs in the form of a two port system. Linear Fractional Transforms exist in three forms: upper, lower and star product. The purpose of the upper and lower LFT is to close the loop with an uncertainty or controller block respectively. The upper LFT is a function of the interconnection coefficient matrix and the norm bounded uncertainty block, and the lower LFT is a function of the interconnection matrix and the controller. M and K are assumed

to be fixed, whereas $\triangle$ is variable but norm bounded. The upper LFT is defined in Eq. 3.17.

$$F_u\left(M, \triangle\right) = M_{22} + M_{21}\triangle\left(I - M_{11}\triangle\right)^{-1} M_{12} \tag{3.17}$$

The lower LFT is defined in Eq. 3.18.

$$F_l\left(N, K\right) = F_u\left(F_l\left(M, K\right), \triangle\right) \tag{3.18}$$

Fig: 3.1 illustrates the role of the upper and lower LFT.

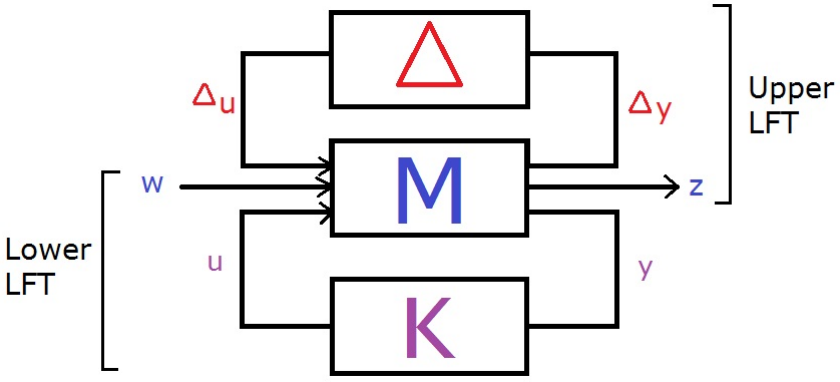


Figure 3.1: Linear Fractional Transforms

The performance output of the system is obtained by closing both loops to obtain the transfer function from $w \rightarrow z$. In almost all cases w and z are vector signals. Performance specifications are expressed as weights on these to reflect different gain and frequency aspects of the performance objectives.

$$z = F_u\left(F_l\left(M, K\right), \triangle\right) w \tag{3.19}$$

There are two important properties of LFTs. The first is that an LFT of an LFT is an LFT, meaning that LFTs can be combined from component LFTs. The second property concerns whether an LFT is well defined. A lower LFT is well defined if $I - M_{22}K$ is invertible [3] and an upper LFT is only well defined if $I - M_{11}\triangle$ is invertible.

3.2 Normalised Coprime Factorisation Synthesis

Robust loopshaping control is based on using a combination of $\mathcal{H}_\infty$ control, loopshaping design, and coprime factorisation. This is also known as the McFarlane-Glover [4] method.

The $\mathcal{H}_\infty$ control aspect seeks to find a controller which minimises the peak gain over frequency, the "H-Infinity" norm of the signal, as this guarantees stability by the small gain theorem. The objective is to find control solutions for plants with unstructured uncertainties where the only available information is the upper bound on magnitude over frequency such as that from a Bode plot with error bounds. This summary of coprime factorisation and robust loopshaping intends to be somewhere between the work of Chen [15], Glover/McFarlane [4], and Gu [7], but with the intent to lead into the parameter varying case addressed by Chen [15] and Prempain [25]. To this aim some equations are presented in both the Gu and Chen notation.

Also, for clarification normalised coprime factorisation is the method of representing the model, LSDP is the method for incorporating control specifications into the framework.

Coprime Factorisation

The nominal coprime factorisation provides a way to represent the uncertainty using two stable factors, rather than in the original form. The two factors are coprime because they themselves are not prime but their product is. This representation allows a plant to be expressed with no hidden modes and so it can represent a plant with poles on the complex axis, and in the perturbed case poles which are shifted by the uncertainty. The uncertainty considered is complex, unstructured stable additive perturbations on the coprime factors. The representation can be in the form of a left or right coprime factorisation, with left being the most common, but both are functionally the same. The nominal form of the factorisation is presented first, and then the perturbed version for uncertain systems is presented. The factorisation was developed by Vidyasagar [29], but the solution to the control problem was developed by Glover and McFarlane [4].

If M^{-1} and N are coprime factors, then the left coprime factorisation is defined in Eq. 3.20.

$$G = M^{-1}N \tag{3.20}$$

The coprime factorisation exists only if there exist $U \in \Re\mathcal{H}_\infty$ and $V \in \Re\mathcal{H}_\infty$, and two coprime factors N and M to satisfy the Bezout identity Eq. 3.21.

$$N(s)U(s) + M(s)V(s) = I \tag{3.21}$$

The coprime factorisation is only normalised if the following condition is met. This is quite an important condition for the solution of the Nehari extension problem [3, p.2, p.200].

$$N(s)N^\dagger(s) + M(s)M^\dagger(s) = I \tag{3.22}$$

In the original development the coprime factorisation was found from the solution to an Algebraic Riccati Equation (ARE) [3, Chapter 13]. The ARE for the left coprime factorisation, the generalised filter algebraic Riccati equation (GFARE) is defined as [7] in Eq. 3.23.

$$\left(A - BD^T R^{-1} C\right) Z + Z(A - BD^T R^{-1} C)^T - ZC^T R^{-1} CZ + B(I - D^T R^{-1} D)B^T = 0 \quad (3.23)$$

where$Z > 0$ is the solution, † denotes complex conjugate transpose and the remaining terms are defined in Eq. 3.24 to 3.26.

$$R = I + DD^T \quad (3.24)$$

$$H = -(ZC^T + BD^T)R^{-1} \quad (3.25)$$

$$\left[\begin{array}{cc} \widetilde{N} & \widetilde{M} \end{array}\right] = \left[\begin{array}{c|cc} A + HC & B + HD & H \\ \hline R^{-\frac{1}{2}} C & R^{-\frac{1}{2}} D & R^{-\frac{1}{2}} \end{array}\right] \quad (3.26)$$

Alternatively the form used in Chen [15] is equivalent and given as Eq. 3.27

$$\left[\begin{array}{cc} \widetilde{M} & \widetilde{N} \end{array}\right] = \left[\begin{array}{c|cc} A + LC & L & B + LD \\ \hline R^{-\frac{1}{2}} C & R^{-\frac{1}{2}} & R^{-\frac{1}{2}} D \end{array}\right] \quad (3.27)$$

The other terms are defined in Eq. 3.28 to Eq. 3.30.

$$L = -(BD^T + ZC^T)R^{-1} \quad (3.28)$$

$$R = I + DD^T \quad (3.29)$$

$$S = I + D^T D \quad (3.30)$$

The factorised model can be used to model uncertainties by considering perturbations on the factors. The model now becomes Eq. 3.31.

$$G = (M + \triangle_M)^{-1} (N + \triangle_N) \quad (3.31)$$

The uncertainties are assumed to be norm bounded by the stability margin $\epsilon = \gamma^{-1}$, which is found from the synthesis procedure. This is shown in Eq. 3.32.

$$\left\| \begin{array}{cc} \triangle_M & \triangle_N \end{array} \right\|_\infty < \epsilon = \gamma^{-1} \tag{3.32}$$

The normalised coprime factorised plant is shown in the diagram in Fig: 3.2.

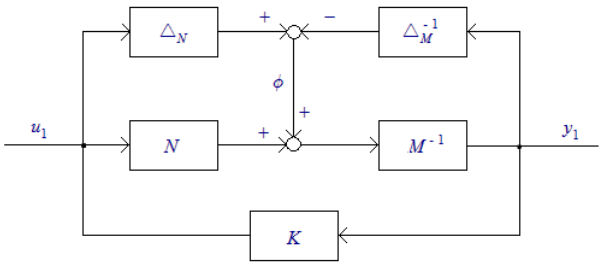


Figure 3.2: Structure of a Coprime Factorisation

The uncertainties are not described directly using $\triangle_M$ and $\triangle_N$, but instead the coprime factorisation is used as a generalised description of complex uncertainty. Complex uncertainty, is lumped uncertainty like a ball on the Nyquist plot, as opposed to parametric real uncertainty. As with other forms of complex uncertainty this can be a conservative representation, and so the resulting controlling may be robust against more uncertainty than may actually be present in the system.

Controller Synthesis

The controller synthesis procedure aims to find a stabilising controller K to maximise the robust stability of the closed loop system by minimising γ, with the minimum value denoted γ_0. The minimisation problem is shown in Eq. 3.33.

$$\left\| \begin{bmatrix} K \\ I \end{bmatrix} (I + PK)^{-1} \tilde{M}^{-1} \right\|_\infty = \gamma \tag{3.33}$$

This is solved, at least in the ARE case by reduction to a Nehari extension problem in Glover and McFarlane [4]. In the Nehari extension problem $\begin{bmatrix} U \\ V \end{bmatrix}$ is an optimal extension of the matrix function $\begin{bmatrix} -N^* \\ M^* \end{bmatrix}$ that satisfies the inequality in Eq. 3.34.

$$\left\| \begin{bmatrix} -N^* \\ M^* \end{bmatrix} + \begin{bmatrix} U \\ V \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} \tilde{N} & \tilde{M} \end{bmatrix} \right\|_H \tag{3.34}$$

U and V form a right coprime factorisation of the stabilising controller K. Therefore the solution of the controller problem using the right coprime factorisation of the controller is related to the left coprime factorisation of the plant. The right coprime factorisation of the plant is defined as Eq. 3.35.

$$K = UV^{-1} \tag{3.35}$$

The normalisation constraint in the coprime factorisation is relevant as it makes the controller unique, as otherwise a plant could have an infinite number of factorisations.

$$N(s)U(s) + M(s)V(s) = I \tag{3.36}$$

The Central Controller

The control problem can be solved using γ iteration as in the case of the $\mathcal{H}_\infty$ mixed synthesis problem. This an important point as it is used in obtaining the LPV controller in chapter 6. An alternative method for coprime factorised models is to use the central controller method. The central controller for a given γ is solved using the solution X to the Generalised Control Algebraic Riccati Equation (GCARE) Eq. 3.37 and the solution Z to the generalised Filter Algebraic Riccati Equation (GFARE) Eq. 3.38.

$$\left(A - BS^{-1}D^TC\right)^T X + X\left(A - BS^{-1}D^TC\right) - XBS^{-1}B^TX + C^T(I - DS^{-1}D^T)C = 0 \tag{3.37}$$

$$\left(A - BD^TR^{-1}C\right)Z + Z(A - BD^TR^{-1}C)^T - ZC^TR^{-1}CZ + B(I - D^TR^{-1}D)B^T = 0 \tag{3.38}$$

A third ARE exists to be solved but the use of a normalised factorisation allows the controller to be found using the solutions of just two AREs and spectral radius condition [30, p.358].

$$\gamma_0 = \frac{1}{\sqrt{1 - \lambda_{max}(YQ)}} \tag{3.39}$$

The state space formulation for the controller is given in Eq. 3.40.

$$K_0 = \left[\begin{array}{c|c} A + BF + \gamma^2(L^T)^{-1}ZC^T(C + DF) & \gamma^2(L^T)^{-1}ZC^T \\ \hline B^TX & -D^T \end{array}\right] \tag{3.40}$$

However as shown in Gu [7], L is singular if $\gamma = \gamma_0$ and so an alternative descriptor form is used Eq. 3.41, with a suboptimal controller i.e $\gamma > \gamma_0$

$$K = \left[\begin{array}{c|c} -L^Ts + L^T(A + BF) + \gamma^2ZC^T(C + DF) & \gamma^2ZC^T \\ \hline B^TX & -D^T \end{array}\right] \tag{3.41}$$

Where F and L are defined by Eq. 3.42 and Eq. 3.43 respectively.

$$F = -S^{-1}(D^T C + B^T X) \tag{3.42}$$

$$L = (1 - \gamma^2)I + XZ \tag{3.43}$$

ARE Solution Assumptions

Some standard assumptions [30, p.354] have to be met for a stabilising controller to exist.

1. (A, B_2) stabilisable and (A, C_2) detectable. B_2 is the control input and C_2 is the measurement output. This is required for the existence of a stabilising controller

2. D_12 and D_{21} have full rank. This is a sufficient condition for the controller to be proper.

3. $\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank $\forall\omega$. Assumptions 3 and 4 are constraints to prevent a controller being synthesised which cancels complex poles or zeros.

4. $\begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix}$ has full row rank $\forall\omega$

5. $D_{11} = 0$ and $D_{22} = 0$. This assumption requires that there be no direct feed-through from the controller input to controller output.

These are all quite standard assumptions in control synthesis.

The Loopshaping Design Procedure

The loopshaping design procedure [13, p.17] (LSDP) is the means by which coprime factorisation and $\mathcal{H}_\infty$ are used to create controllers with defined performance specifications other than just robust stability. The transfer function used in LSDP is equivalent to the S/KS control problem in mixed synthesis, which suggests that it gives robustness against additive uncertainty and either nominal reference tracking or disturbance rejection. In common with other $\mathcal{H}_\infty$ control methods the objective is to minimise the infinitive norm of the transfer function. The stacked S/KS configuration is shown in Eq. 3.44.

$$\left\| \begin{bmatrix} w_s S \\ w_{ks} KS \end{bmatrix} \right\|_\infty \tag{3.44}$$

The specifications are defined in terms of the closed loop singular values, but are implemented by augmenting the plant with pre-weights (W_1) and post-weights (W_2) on the open loop system. Combined these form the augmented plant, to which coprime factorisation is then applied before the controller is obtained. The augmented loop is defined in Eq. 3.45.

$$G_s = W_2 G W_1 \tag{3.45}$$

The weights must also be placed in series with the synthesised controller:

$$K_s = W_1 K W_2 \tag{3.46}$$

This creates a problem as the closed loop response is not easily defined in terms of the open loop response, although broad estimates can be made of the effect of an open loop weight on the closed loop response. Typically in loop shaping the minimum singular value of the open loop plant should be large at low frequency which provides good reference tracking response, and the maximum singular value should be small at high frequency for good robustness. In the transition between the two regions the roll-off rate should be less than -20dB per decade. Unlike in Bode type loopshaping, the phase is not shaped. Good results can be obtained with one frequency domain weight and a scalar. This is best illustrated with a diagram as shown in Fig: 3.3.

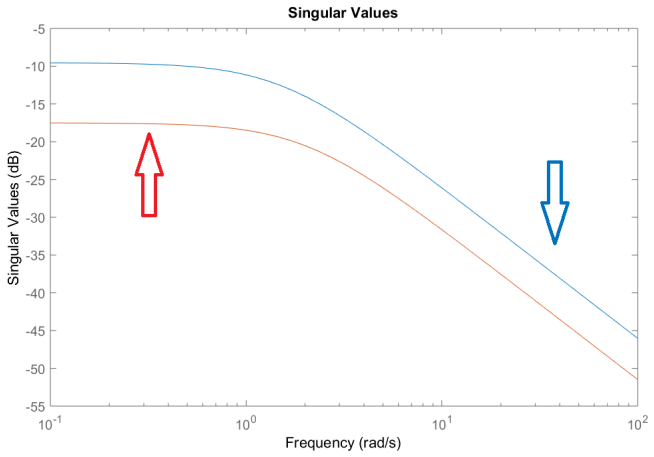


Figure 3.3: Loopshaping Procedure

The final step in the LSDP algorithm is to compute the solutions to the AREs which solve the controller and factorisation problems. An advantage of the LSDP method is that it avoids a requirement for γ-iteration as is required with mixed synthesis. This is because LSDP uses a spectral radius condition on the solution of the two AREs. The LSDP algorithm leads to a value of γ which is indicative of the robustness of the designed system. Ideally γ should be the following range [13, p.16, Theorem 2.5.1] in Eq. 3.47.

$$1 \leq \gamma < 3.3 \tag{3.47}$$

A γ value exceeding 3 indicates poor robustness, particularly values much larger such as 40 or 50. Values less than 1 would indicate a stability margin greater than 1, as they are reciprocal, and so robustness against more than 100% uncertainty.

The methods to solve the LSDP problem are through Algebraic Riccati Equations (ARE) or through Linear Matrix Inequalities (LMI). The ARE is the traditional approach and require the solution of two AREs and a spectral radius condition. This was the first method used to solve the LSDP problem. LMIs are a more complex method to solve the LSDP problem but have the advantage that pole placement constraints can be defined to eliminate high frequency dynamics introduced by the controller. The principles of the loopshaping procedure are also applicable to the LPV method in the next section.

Gain scheduling Control

Linear controllers are designed to operate at a given operating point, but performance can be enhanced by using multiple controllers, each at a different operating point. This is achieved through a process of gain scheduling, where a gain or controller is chosen based on the value of some independent parameter of the system state or output measurement, typically denoted ρ. In the LPV chapter more complex type of scheduling are addressed, but here a method based on interpolating discrete linear controllers is used. This is illustrated in Fig: 3.4.

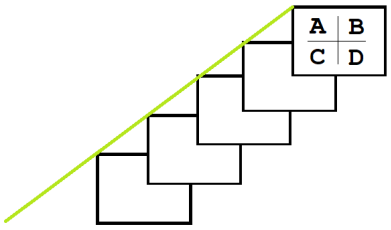


Figure 3.4: Standard Gain Scheduling

The main drawback is that the controller switching system is itself a dynamical system which can be stable or unstable, and can exhibit oscillations and limit cycles in much the same way as the physical system. This is why the more complex methods exist.

3.3 LPV-NCFS

Linear Parameter Varying control (LPV) can be applied to the McFarlane Glover method of synthesis. This is a less widely studied area of control but has been successfully applied. There are two main steps, the first is the creation of the coprime factorised model, and the second is the creation of the LPV controller, and in each case involve solving a system of Linear Matrix Inequalities (LMIs).

In the control synthesis step the stability of LPV based $\mathcal{H}_\infty$ controller is established through Lyapunov functions. Aleksandr Lyapunov studied the flow of energy in rotating

masses [34, 35], but his work was later found to be applicable to nonlinear systems such as generalised inverted pendulums (rockets). Lyapunov proposed two analysis methods, but it is the second that is relevant here. For reference the first method is the requirement that the real parts of the Eigenvalues be negative definite. The principle of Lyapunov stability using his second method is that a system is stable at some equilibrium point if there exists a positive definite energy function $V(x)$ whose derivative $\dot{V}(x)$ is negative semi-definite. The energy function could describe something like the kinetic energy in a spring, and it is known that when a spring is released from compression the energy function derivative will be negative semi-definite until it reaches equilibrium. This can be written as Eq. 3.48.

$$\dot{x} = Ax \tag{3.48}$$

$$V(x) > 0$$

$$\dot{V}(x) \leq 0$$

The Quadratic Lyapunov function is a generalisation of this principle. This is used as the basis of the stability proof for time varying systems such as LPV. A time varying system, and Lyapunov candidate function can be defined as in Eq. 3.50 [33]. Note that the system is time varying because $A(t)$ is a function of time.

$$E(t)\dot{x} = A(t)x(t) \tag{3.49}$$

$$V(x) = x^T P x \tag{3.50}$$

The solution to this, $Q = P^{-1}$, can be found as the solution to Eq. 3.51. This requires an infinite number of LMIs, but can be made finite by using Affine or Polytopic systems.

$$A(t)QE(t)^T + E(t)QA(t)^T < 0 \tag{3.51}$$

Two approaches are available in the application to LPV systems, Singular Quadratic Lyapunov Functions (SQLF) and Parameter Dependent Lyapunov Functions [42] (PDLF). The SQLF assumes that the parameter varies arbitrarily fast, and so is more conservative as any real world parameter is likely to have some bounds on its derivative. SQLF has the advantage of being conceptually simpler, and being an easier condition to test. The PDLF case is much the same as the SQLF, but with the variation rate being considered, and so trade an increase in LMI complexity for a less conservative estimate of stability.

3.3.1 LPV System Representation

LPV systems are obtained first by linearisation of the non-linear model, and then by representation in some kind of varying framework. The methods for linearisation can be Jacobian [24, p.21], State Transformation [24, p.24], or Function Substitution [24, p.25]. Jacobian is by far the most common method because it is the one most Engineers are familiar with, and the easiest to apply without error. The method for representing the LPV model is less straightforward as the several which have widespread usage including Grid LPV, Affine LPV [24, p.27] and Tensor-Polytopic (TP) [24, p.28] LPV models. In Affine systems the varying model is represented as a set of state space matrices which are affinely dependent on the independent parameter, $\theta(t) \in [\underline{\sigma}_i, \overline{\sigma}_i]$. In Matlab [33] each matrix is an ltisys object, and each S matrix represents the model contribution associated with parameter θ_i. The number and meaning of the θ values depends on the choice of basis function, so θ_1 could be velocity and θ_2 could be the square of velocity. The only exception to this is S_0 which is the baseline nominal model onto which the other components are added. With reference to the descriptions in [24, p27].

The system matrix combination is Eq. 3.52.

$$S(\theta) = S_0 + \sum_{i=1}^{n} \theta_i S_i = S_0 + \theta_1 S_1 + \theta_i S_i + \ldots + \theta_n S_n \tag{3.52}$$

Each S has the following form Eq. 3.53.

$$S_i = \begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} = \left[\begin{array}{c|cc} A & B_1(\theta) & B_2(\theta) \\ \hline C_1(\theta) & D_{11}(\theta) & D_{12} \\ C_2(\theta) & D_{21} & D_{22} \end{array}\right], \; i = 0 \ldots n \tag{3.53}$$

This can be written as a convex combination of the matrix vertices, with $r = 2^n$ being the total number of vertices. This has to be done although it has the effect of making the model far less intuitive. The convex representation can be obtained using the aff2pol function. This is the representation required for the method presented in Prempain [25], Eq. 3.54.

$$S(\rho) = \sum_{j=1}^{r} \alpha_j \hat{S}_j = \alpha_1 \hat{S}_1 + \alpha_2 \hat{S}_2 + \ldots + \alpha_r \hat{S}_r \tag{3.54}$$

Where S_j is defined in Eq. 3.55.

$$\hat{S}_j = \begin{bmatrix} \hat{A}_j & \hat{B}_j \\ \hat{C}_j & \hat{D}_j \end{bmatrix} \tag{3.55}$$

The controllers are obtained at an operating point by convex interpolation of the vertex controllers. The controller, or plant, can be accessed using psinfo. This request would have

to be made in the vertex form in most cases, and so the point can be described in affine space and converted using polydec(pv,p), where pv is the parameter description, and p is the point. The code ultimately would have the following form.

```
c = polydec(pv,p)
psinfo(pdK,'eval',c)
```

The parameter varying system is composed of a parameter varying A matrix, but with B, C and D matrices fixed. This constraint is required to ensure that the number of LMIs to be solved is finite. If B and C must vary then this can be addressed by pre-filtering them, although this is not required here. This is shown in Eq. 3.56.

$$G(\rho) = \left[\begin{array}{c|c} A(\rho) & B \\ \hline C & D \end{array}\right] \tag{3.56}$$

3.3.2 Coprime Factorisation

The coprime factorisation used in Chen [15] is that from Prempain [25]. It is much the same as that used in standard McFarlane Glover synthesis, but is parameter varying rather than fixed. This makes the problem significantly more difficult. The factorisation is unique and can be found by solving a generalised $\mathcal{H}_2$ optimisation problem involving a finite number of LMIs.

The LMI to be solved, for P and Z, to obtain the coprime factorisation from [25] is Eq. 3.57 to 3.59.

$$\min trace(Z) \tag{3.57}$$

$$\left[\begin{array}{cc} PA_i + A_iP - C^TC & PB - C^TD \\ B^TP & -R \end{array}\right] < 0, i = 1, \ldots, r \tag{3.58}$$

$$\left[\begin{array}{cc} Z & I \\ I & P \end{array}\right] > 0 \tag{3.59}$$

Where R is defined as Eq. 3.60.

$$R = I + D^TD \tag{3.60}$$

This leads to two values, Z and P, more often denoted Y and X respectively. P can be put into a formula to obtain the left coprime factorisation [25] in Eq. 3.61.

$$\left[\begin{array}{cc} \tilde{M}(\rho) & \tilde{N}(\rho) \end{array}\right] = \left[\begin{array}{c|cc} A(\rho)+LC & L & B+LD \\ \hline \tilde{R}^{-\frac{1}{2}}C & \tilde{R}^{-\frac{1}{2}} & \tilde{R}^{-\frac{1}{2}}D \end{array}\right] \tag{3.61}$$

Where L is defined as Eq. 3.62 and R is defined as Eq. 3.63.

$$L = -(BD^T + P^{-1}C^T)\tilde{R}^{-1} \tag{3.62}$$

$$\tilde{R} = I + DD^T \tag{3.63}$$

This coprime factorised model is constructed using the components of the LPV model at points i, and the solution of the ARE.

3.3.3 McFarlane Glover LPV Controller

The controller step is similar. A controller K is sought to satisfy the following inequality for all parameters in the polytope. This is an extension of the inequality in standard McFarlane Glover synthesis for the robustness γ of the system, to a parameter varying system $G(\theta)$. The optimisation problem is Eq. 3.64.

$$\left\| \left[\begin{array}{c} K(\rho) \\ I \end{array}\right] (I + G(\rho)K(\rho))^{-1}\tilde{M}(\theta)^{-1} \right\|_\infty < \gamma \tag{3.64}$$

The solution is to find matrices Q and S by solving the LMIs in Eq. 3.65 to Eq. 3.68. These can then be substituted into standard formulae to obtain a family of parameter varying McFarlane Glover controllers.

$$S(A_i + LC) + (A_i + LC)^T S - \gamma C^T \tilde{R}^{-1} C < 0 \tag{3.65}$$

$$\left[\begin{array}{ccc} A_iQ + QA_i^T - \gamma BB^T & QC^T - \gamma DB^T & -L\tilde{R}^{\frac{1}{2}} \\ CQ - \gamma DB^T & -\gamma\tilde{R} & \tilde{R}^{\frac{1}{2}} \\ -\tilde{R}^{\frac{1}{2}}L^T & \tilde{R}^{\frac{1}{2}} & -\gamma I_{ny} \end{array}\right] < 0 \tag{3.66}$$

$$i = 1 \ldots r \tag{3.67}$$

$$\left[\begin{array}{cc} Q & I \\ I & S \end{array}\right] \geq 0 \tag{3.68}$$

The standard formulas for the controller are [15, p53] Eq. 3.69 to Eq. 3.72.

$$A_K(\rho) = -Q(\rho)^{-1}[A^T(\rho) + X(\rho)A(\rho)Y(\rho) - \gamma X(\rho)B(\rho)B^T(\rho) \tag{3.69}$$

$$-\gamma C^T(\rho)C(\rho)Y + X(\rho)LC(\rho)Y(\rho) + \gamma^{-1}C^T(\rho)C(\rho)Y(\rho)$$

$$+C^T(\rho)L^T + X(\rho)\dot{Y}(\rho) + \dot{Q}(\rho)S(\rho)]S^{-T}$$

$$B_K(\rho) = Q^{-1}(\rho)(X(\rho)L - \gamma C^T(\rho)) \tag{3.70}$$

$$C_K(\rho) = -\gamma B^T(\rho)S(\rho)^{-T} \tag{3.71}$$

$$D = 0 \tag{3.72}$$

The LMI problems can be solved using the LMI toolbox [33] or Robust Control toolbox [41] in Matlab. The general solution algorithm requires iterating through the plants, updating the two or three LMI terms, calling the solver mincx(), and then extracting the results using mat2dec().

3.4 Backstepping and Dynamic Inversion (ref. Literature Review)

3.4.1 Feedback Linearisation

Feedback linearisation [31] is a method for linearising a non-linear system by cancelling nonlinear components using feedback. An accurate model of the system dynamics is required in order to cancel the nonlinear terms as the class of techniques use inversion, and perfect inversion is rarely achievable in practice. Robust control techniques can be used to compensate for the inaccuracy in the nonlinear part by incorporating the uncertainty into the linear model. To use this the control input must appear in the equation for the output, and if this is not the case then the output must be differentiated until this is the case.

The largest number of times an output has to be differentiated is referred to as the relative degree of the system.

A common abbreviation is to define the Lie derivative as in Eq. 3.73.

$$\dot{y} = \frac{dh(x)}{dx}f(x) = \mathcal{L}_f h(x) \tag{3.73}$$

The second Lie derivative is defined in similar manner in Eq. 3.74

$$\mathcal{L}_f^2 h(x) = \frac{d}{dx}\left(\frac{dh(x)}{dx}\right) f(x) \tag{3.74}$$

Consider a nonlinear state space system where f(x) and g(x) are vector fields, and are vectors for a given point x, as in Eq. 3.75.

$$\dot{x} = f(x) + g(x)u \tag{3.75}$$

$$y = h(x)$$

The control law to be obtained has the form in Eq. 3.76.

$$u = \alpha(x) + \beta(x)v \tag{3.76}$$

This leads to a state space system of the form in Eq. 3.77 (i.e with the control terms included)

$$\dot{x} = f(x) + g(x)\alpha(x) + g(x)\beta(x)v \tag{3.77}$$

$$y = h(x)$$

The process is to continue taking derivatives of the output, if the derivatives exist, until the input appears at the output. The technique is typically applied to state space type systems so the derivatives tend to exist and be sufficiently smooth until at least the last state where in most SISO cases the real control enters.

$$\dot{y} = \mathcal{L}_f h(x) \tag{3.78}$$

$$\ddot{y} = \mathcal{L}_f^2 h(x) + \mathcal{L}_g \mathcal{L}_f h(x)u \tag{3.79}$$

This leads to v

$$v = \ddot{y} = \mathcal{L}_f^2 h(x) + \mathcal{L}_g \mathcal{L}_f h(x)u \tag{3.80}$$

This can be rearranged to obtain the control input u as in Eq. 3.81. Note that the second term here is an inversion of β

$$u = -\frac{\mathcal{L}_f^2 h(x)}{\mathcal{L}_g \mathcal{L}_f h(x)} + \frac{1}{\mathcal{L}_g \mathcal{L}_f h(x)} v \tag{3.81}$$

Alternatively in terms of α and β as in Eq. 3.82 to Eq. 3.84.

$$\alpha = \mathcal{L}_f^2 h(x) \tag{3.82}$$

$$\beta = \mathcal{L}_g \mathcal{L}_f h(x) \tag{3.83}$$

$$u = \beta^{-1}(-\alpha + v) = \frac{1}{\mathcal{L}_g \mathcal{L}_f h(x)}(-\mathcal{L}_f^2 h(x) + v) \tag{3.84}$$

An alternative approach is to use iterative backstepping. This has the advantage that desirable nonlinearities can be used to help stabilise the system. Backstepping is a sequential iterative approach which lends itself to automation, and involves the obtaining of a stability proof in the form of a Lyapunov function as part of the process. It is also significantly more messy due to the iteration and redefinition of error terms, virtual controls, and Lyuapunov functions.

Chapter 4

System Identification and Modelling

4.1 System Dynamics

The dynamics of a quadrotor are contributed to by the effects of gravity, drag, the fuselage, motors, rotors/propellers, off-centre masses such as batteries and control boards, and potentially in this case also a dynamic mass payload. These effects are combined together to create a non-linear model and then finally a linear model. The dynamics have been well studied in the literature, particularly in [32], and so only a summary of the contributing components will be given.

Fuselage

The fuselage is assumed to be a rigid mass. The construction is such that this is an accurate approximation. Other masses such as batteries and control boards are lumped in with this to calculate the mass and moments of inertia of the fuselage.

$$J_{xx} = Jyy = \frac{1}{4}mr^2 \tag{4.1}$$

$$J_{zz} = \frac{1}{2}mr^2 \tag{4.2}$$

The model description is split into rotational and translational parts, with the rotational part typically being the higher frequency and more complex part to control. The translational part is highly nonlinear at higher angles of operation. The kinematics are similar to a traditional quadrotor but instead of being an X shape the aircraft is an H shape. In the X shape control of pitch requires one motor to increase thrust and the other opposite it to decrease thrust, and the same for roll. In the H shape to pitch up both front motors would have to increase and both rear motors decrease thrust. However the

dynamics are similar enough in principle and one model can be substitute for the other in the development of the proof of concept controllers. The Newton-Euler formalism is used to describe the dynamics of the system, and so the three state variables are ϕ, θ, and ψ. The alternative is the Quaternion which has the advantage of being singularity free, but the disadvantage of making the dynamics description and potentially the reference tracking control more complicated. The state matrix formulation for $\begin{pmatrix} \phi & \theta & \psi \end{pmatrix}$ depends on the choice of first derivative state variables. If $\begin{pmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{pmatrix}$ are chosen then the dynamics are just the identity matrix for integration. If the angular rates $\begin{pmatrix} p & q & r \end{pmatrix}$ are used then the must be rotated into Euler form. This is because $\begin{pmatrix} p & q & r \end{pmatrix}$ are three scalar variables, whereas $\begin{pmatrix} \phi & \theta & \psi \end{pmatrix}$form a vector of three related variables, where two rotations about one axis can be expressed as one rotation about the third axis.

4.1.1 Rotational Dynamics

The angular acceleration terms are obtained by expanding out Eq. 4.3 for a rigid mass with moments M from sources such as motor force differences acting as inputs, and resulting in angular velocities $\begin{pmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{pmatrix}$ and accelerations $\begin{pmatrix} \ddot{\phi} & \ddot{\theta} & \ddot{\psi} \end{pmatrix}$ about a point, damped by inertia terms J.

$$J\dot{\omega} + \omega \times J\omega = M \tag{4.3}$$

The inertia terms are given by Eq. 4.4.

$$J = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \approx \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{4.4}$$

In Eq. 4.5 to Eq. 4.7, the angular rate vector ω becomes $\begin{pmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{pmatrix}$, and the inertia matrix J is broken into its component form. Drag terms K_r are also added.

$$\ddot{\phi} = -k_r\dot{\phi} - \dot{\theta}\dot{\psi}\left(\frac{I_{zz} - I_{yy}}{I_{xx}}\right) + \frac{1}{I_{xx}}\tau_\phi \tag{4.5}$$

$$\ddot{\theta} = -k_r\dot{\theta} - \dot{\phi}\dot{\psi}\left(\frac{I_{xx} - I_{zz}}{I_{yy}}\right) + \frac{1}{I_{yy}}\tau_\theta \tag{4.6}$$

$$\ddot{\psi} = -k_r\dot{\psi} - \dot{\phi}\dot{\theta}\left(\frac{I_{yy} - I_{xx}}{I_{zz}}\right) + \frac{1}{I_{zz}}\tau_\psi \tag{4.7}$$

For $x = \begin{pmatrix} \phi & \theta & \psi \end{pmatrix}$ there are two potential cases. If gyroscopic rates p,q,r are used are above the following dynamics of Eq. 4.8 are used.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} \tag{4.8}$$

Otherwise the following non-linear form in Eq. 4.9 is used in which the angular rates are rotated into Euler form.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} p + q\sin(\phi)\tan(\theta) + r\cos(\phi)\tan(\theta) \\ q\cos(\phi) - r\sin(\phi) \\ \cos(\theta)^{-1}\left[q\sin(\phi + r\cos(\phi)\right] \end{bmatrix} \tag{4.9}$$

The control distribution maps torques for each axis and total force onto motor forces. Here T_{tot} is the total force from all four motors, d is drag, and $m_i : \quad i = 1 \ldots 4$ are individual motor forces.

$$T_{tot} = m_1 + m_2 + m_3 + m_4 \tag{4.10}$$

$$\tau_\phi = -l \cdot m_2 + l \cdot m_4 \tag{4.11}$$

$$\tau_\theta = -l \cdot m_1 + l \cdot m_3 \tag{4.12}$$

$$\tau_\psi = d \cdot m_1 - d \cdot m_2 + d \cdot m_3 - d \cdot m_4 \tag{4.13}$$

4.1.2 Translational Dynamics

The translational dynamics control the position, velocity and acceleration of the aircraft in local space relative to the starting point. The movement is controlled by rotating the force vector using the rotation subsystem, and then maintaining a constant height which causes the aircraft to translate. In the following d_x and d_y are drag terms, m is the total mass, T_{tot} is the total thrust, and (x, y, z) define the position.

$$\ddot{x} = T_{tot} \cdot [\cos\psi \sin\theta \cos\phi + \sin\psi \sin\phi] - \frac{d_x}{m}\dot{x} \tag{4.14}$$

$$\ddot{y} = T_{tot} \cdot [\cos\phi \sin\theta \sin\psi - \cos\psi \sin\phi] - \frac{d_y}{m}\dot{y} \tag{4.15}$$

$$\ddot{z} = T_{tot} \cdot [\cos\theta \cos\phi] - \frac{1}{m}\dot{z} \tag{4.16}$$

It has been noted that these are relative to the start point, but this can be translated into more global coordinate systems for localisation using GPS.

4.2 Linearised Model (Full)

Linearised system dynamics include the inertia terms and the rotational drag. The first model includes only rotational terms, but the second model also includes height terms. These are models that are suitable for linear control systems in the inner loop. The model is defined in Eq.

$$x = \begin{bmatrix} \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix} \tag{4.17}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -k_r & -\left(\frac{I_{zz}-I_{yy}}{I_{xx}}\right) & -\left(\frac{I_{zz}-I_{yy}}{I_{xx}}\right) \\ 0 & 0 & 0 & -\left(\frac{I_{xx}-I_{zz}}{I_{yy}}\right) & -k_r & -\left(\frac{I_{xx}-I_{zz}}{I_{yy}}\right) \\ 0 & 0 & 0 & -\left(\frac{I_{yy}-I_{xx}}{I_{zz}}\right) & -\left(\frac{I_{yy}-I_{xx}}{I_{zz}}\right) & -k_r \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix}, \; D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Linearised dynamics augmented with height.

$$x = \begin{bmatrix} \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} & h & \dot{h} \end{bmatrix} \tag{4.18}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -k_r & -\left(\frac{I_{zz}-I_{yy}}{I_{xx}}\right) & -\left(\frac{I_{zz}-I_{yy}}{I_{xx}}\right) & 0 & 0 \\ 0 & 0 & 0 & -\left(\frac{I_{xx}-I_{zz}}{I_{yy}}\right) & -k_r & -\left(\frac{I_{xx}-I_{zz}}{I_{yy}}\right) & 0 & 0 \\ 0 & 0 & 0 & -\left(\frac{I_{yy}-I_{xx}}{I_{zz}}\right) & -\left(\frac{I_{yy}-I_{xx}}{I_{zz}}\right) & -k_r & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{I_{xx}} & 0 & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{zz}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{m} \end{bmatrix}, \; D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Chapter 5

Robust Control Synthesis

5.1 Control Specifications and Design

The specifications for the control system are

1. Stabilise the quadrotor in pitch and roll orientations
2. Reject disturbance inputs to the roll and pitch
3. Be robust against uncertainties in attitude control
4. Track reference angle in yaw
5. Track reference angle in pitch and roll, to move at a new position at some velocity

Controller Structure

A two loop control structure has been chosen, with an inner controller operating at higher update rates stabilising the rotation dynamics against disturbances and tracking rotation angles, and an outer controller for position reference tracking. The robust controller of the inner loop is the main focus of this dissertation, the outer controllers are simpler by comparison.

Fig: 5.1 shows the outer loop controller configuration.

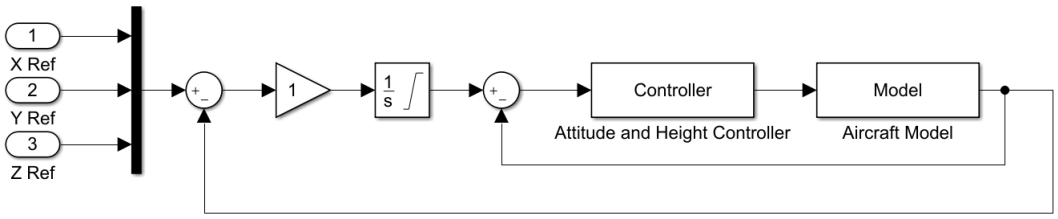


Figure 5.1: Outer Control Loop

Fig: 5.2 shows the inner loop controller configuration.

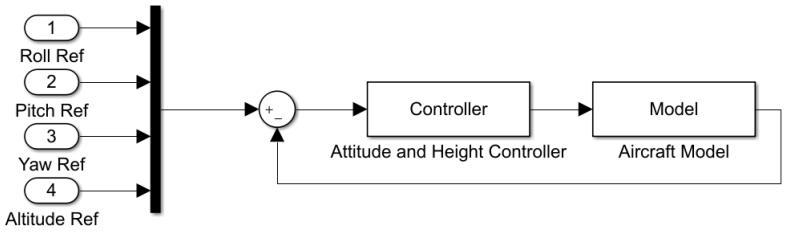


Figure 5.2: Inner Control Loop

5.2 Cont1: Proof of Concept Controller

As the project was split into two streams an initial controller was developed using a simplified model. The purpose of this controller was to demonstrate that the synthesis process was valid for models of the proposed type. The state space model has drag and inertia modelled but cross coupling effects are not included. This has been done to simplify the design process. The model is defined in Eq. 5.1 and Eq. 5.2.

$$\dot{x} = Ax + Bu \tag{5.1}$$

$$y = Cx + Du$$

where

$$A = \begin{bmatrix} 0 & 0 & 0 & 0.9900 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9900 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9900 \\ 0 & 0 & 0 & -7.1429 & 0 & 0 \\ 0 & 0 & 0 & 0 & -7.1429 & 0 \\ 0 & 0 & 0 & 0 & 0 & -18.1818 \end{bmatrix}, \; B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 109.2857 & 0 & 0 & 0 \\ 0 & 109.2857 & 0 & 0 \\ 0 & 0 & 278.1818 & 0 \end{bmatrix} \tag{5.2}$$

$$C = \begin{bmatrix} 57.2958 & 0 & 0 & 0 & 0 & 0 \\ 0 & 57.2958 & 0 & 0 & 0 & 0 \\ 0 & 0 & 57.2958 & 0 & 0 & 0 \end{bmatrix}, \; D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The weights were chosen to have a similar cutoff frequency to the quadrotor, but with a steeper roll-off and less gain. The gain was chosen experimentally to achieve less overshoot

and a stability margin close to 3. Care was also taken to not over-optimise the weights and controller to a point that they would not be realistic. The weights settled on are Eq. 5.3 and Eq. 5.4.

$$W_1 = \frac{1.3}{0.5s + 5.9} \tag{5.3}$$

$$W_2 = 0.4 \tag{5.4}$$

The shaped plant is defined as in Eq. 5.5.

$$G_s = W_1 G W_2 \tag{5.5}$$

The singular values of the original and shaped plants are shown in Fig: 5.3.

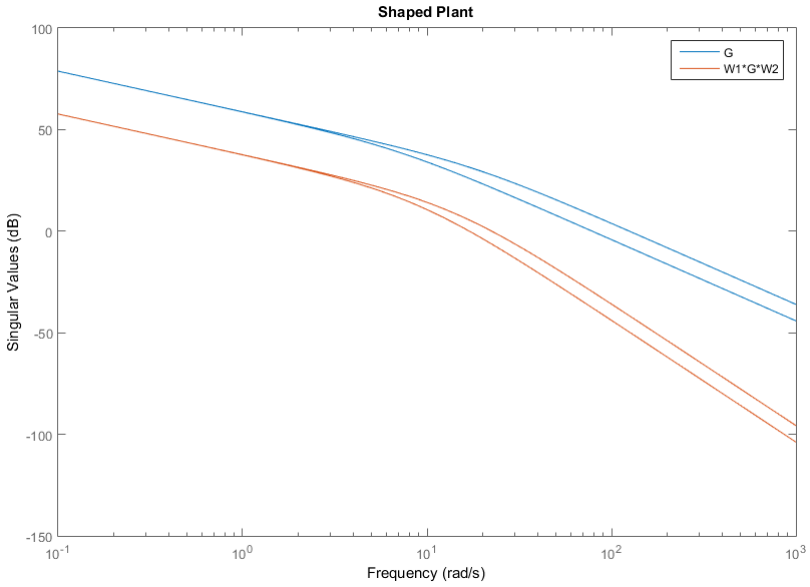


Figure 5.3: Controller 1: Shaped Plant

The controller is then synthesised using the following code

```
[K,CL,GAM,INFO] = ncfsyn(G,W1,W2)
K = -K;
```

This results in a controller with a stability margin of $\gamma = 3.1574$. This indicates robustness to 30% generalised uncertainty.

The controller matrix values are shown in Eq. 5.6. From these it can be seen that there are no very large or very small values, and such values are often indicative of an

over-optimised or difficult to numerically implement design. Note that precision of the values has been reduced for cleaner formatting.

$$K_a = \begin{bmatrix} -45.6 & 0 & 0 & -1.7 & -33.7 & 0 & 15.9 & -0.1 & 0 \\ 0 & -45.6 & 0 & -33.7 & 1.7 & 0 & 0.1 & 15.9 & 0 \\ 0 & 0 & -60.3 & 0 & 0 & -44.0 & 0 & 0 & 17.8 \\ 1.5 & 29.0 & 0 & -11.3 & 0 & 0 & -0.4 & -7.1 & 0 \\ 29.0 & -1.5 & 0 & 0 & -11.3 & 0 & -7.1 & 0.4 & 0 \\ 0 & 0 & 38.7 & 0 & 0 & -18.4 & 0 & 0 & -8.0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -11.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11.8 \end{bmatrix} \quad (5.6)$$

$$K_b = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.8 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.8 \end{bmatrix}, \; K_d = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$K_c = \begin{bmatrix} -10.5 & -0.1 & 0 & -0.3 & -5.4 & 0 & 3.8 & 0 & 0 \\ 0.1 & -10.5 & 0 & -5.4 & 0.3 & 0 & 0 & 3.8 & 0 \\ 0 & 0 & -12.0 & 0 & 0 & -6.1 & 0 & 0 & 3.8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The sensitivity (S), complementary sensitivity (T) and controller gain function (KS) singular value plots S, T and KS respectively are shown in Fig: 5.4. These give a lot of insight into the performance of the controller. The maximum singular value of T is small relatively at high frequency indicating very good robustness characteristics, which further supports the robustness suggested by the robust stability margin. The model does not have full actuator models, so the system appears to have good reference tracking performance at 20 rad/sec where actually the gain is likely to be better at 3 to 6 rad/sec. The KS loop shape indicates low control usage, and a concentration of control effort around the bandwidth where performance is desired.

```
% loop functions
```

```
loops = loopsens(G,K);
sigma(loops.So, loops.To, loops.CSo);
legend('S', 'T', 'KS');
```

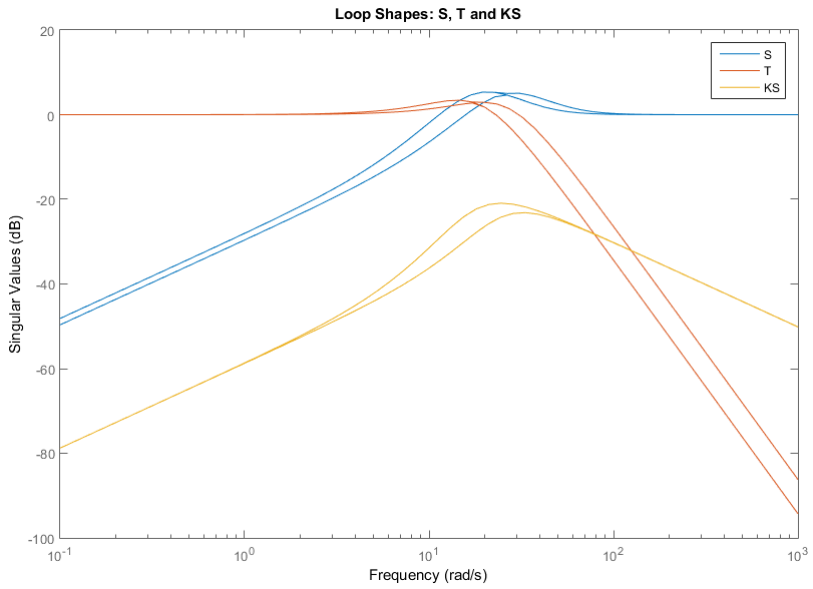


Figure 5.4: Controller 1: S, T and KS

It can be seen in the pole zero diagram in Fig: 5.5 that there are no high frequency poles which could otherwise cause implementation problems.

```
pzplot(K)
```

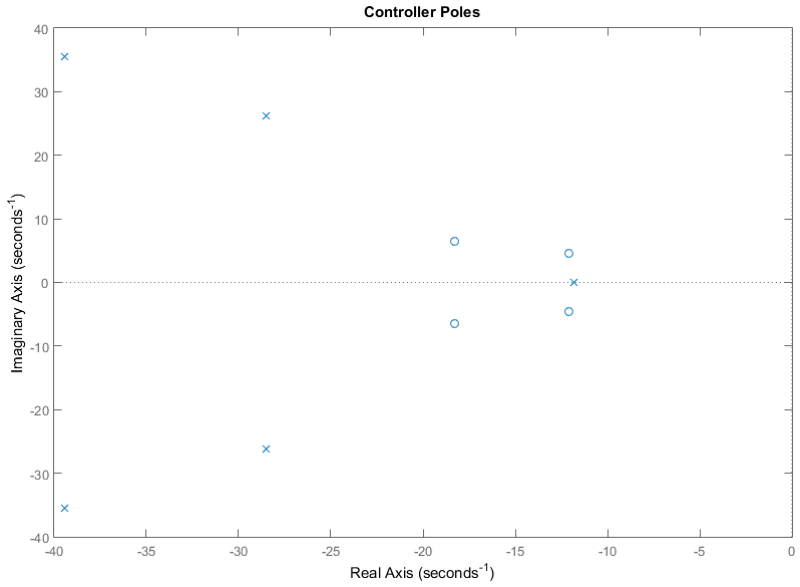


Figure 5.5: Controller 1: Pole Zero Diagram

The step response of the control system is shown in Fig: 5.6

`step(feedback(G,K),3)`

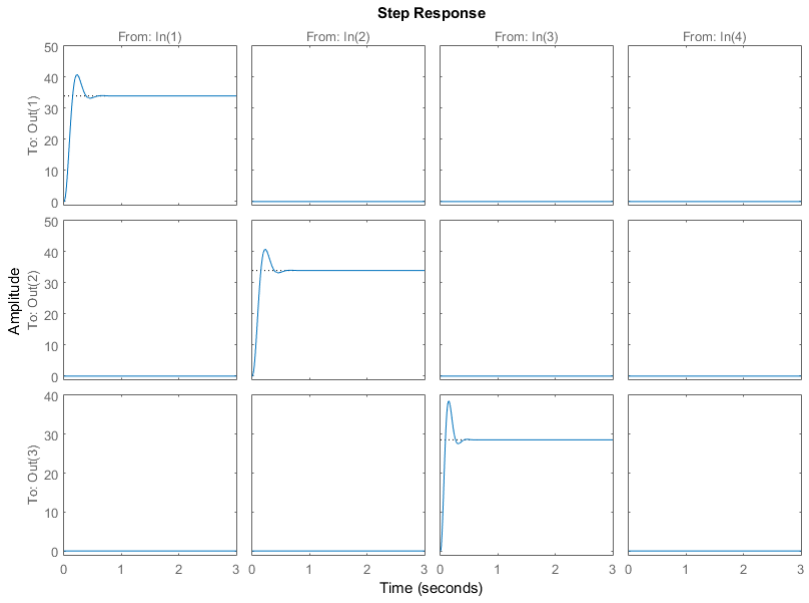


Figure 5.6: Controller 1: Step Response

Controller Order Reduction

Controller order reduction [43] is used in this case to maintain a consistent size of controller matrices to make Simulink simulation easier, otherwise numerous changes would be required if the controller state space model matrices changed in size. The controller order cannot be reduced further beyond 9^{th} order without significantly degrading the performance of the controller. The reduction was done using the following code, with the default method of balanced truncated model reduction via square root method (balancmr), but other methods such as the Hankel minimum degree approximation and Schur method are also available.

```
[K2, redinfo2] = reduce(K,9);
```

For example the step response for a 7^{th} order controller is shown in Fig: 5.7, in which it can be seen that performance is retained mostly in input channels 1 and 2, but the system is no longer controllable in channel 3 (yaw). Note that the fourth channel, total thrust, was never in use.

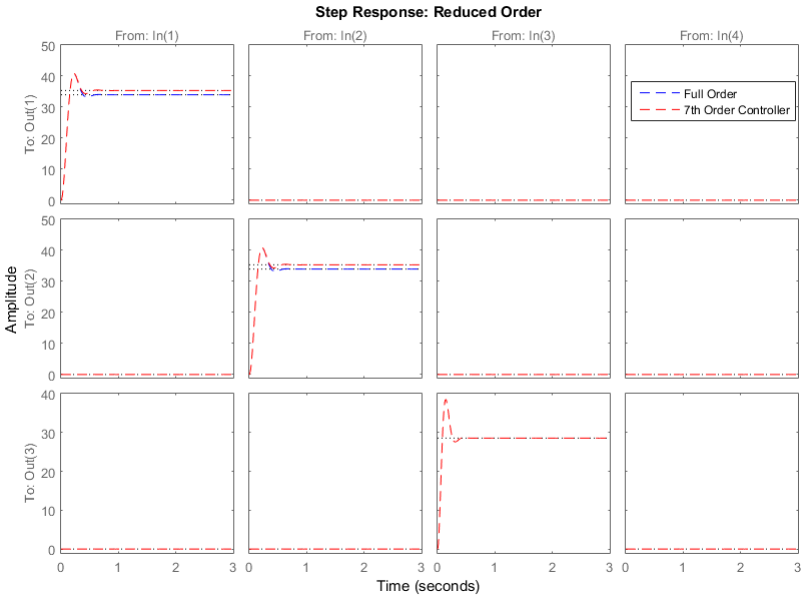


Figure 5.7: Controller 1: Step Response (9th Order)

5.3 Cont2: Proof of Concept Controller with coupling

In this case a controller is synthesised for the model case with coupling effects, but using the same weights as before. The model used is given in Eq. 5.7 and Eq. 5.8.

$$\dot{x} = Ax + Bu \tag{5.7}$$

$$y = Cx + Du$$

where

$$A = \begin{bmatrix} 0 & 0 & 0 & 0.9900 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9900 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9900 \\ 0 & 0 & 0 & -7.1429 & 0.6071 & 0.6071 \\ 0 & 0 & 0 & 0 & -7.1429 & -0.6071 \\ 0 & 0 & 0 & 0 & 0 & -18.1818 \end{bmatrix}, \; B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 109.2857 & 0 & 0 & 0 \\ 0 & 109.2857 & 0 & 0 \\ 0 & 0 & 278.1818 & 0 \end{bmatrix} \tag{5.8}$$

$$C = \begin{bmatrix} 57.2958 & 0 & 0 & 0 & 0 & 0 \\ 0 & 57.2958 & 0 & 0 & 0 & 0 \\ 0 & 0 & 57.2958 & 0 & 0 & 0 \end{bmatrix}, \; D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The singular values of the original and shaped plants are shown in Fig: 5.8

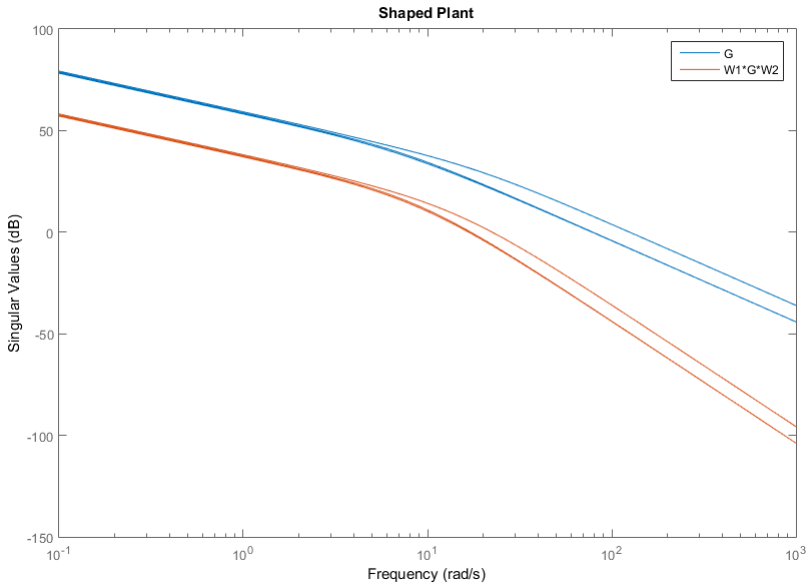


Figure 5.8: Controller 2: Shaped Plant

This results in a controller with a stability margin of $\gamma = 3.1648$, compared to $\gamma = 3.1648$ in the first controller. This indicates robustness to 30% of generalised uncertainty. The controller gains are shown in Eq. 5.9, and again no extreme values are present.

$$K_a = \begin{bmatrix} -139.1 & -0.9 & -0.1 & 111.4 & 1.1 & -0.2 & 68.4 & 5.9 & -6.1 & -40.4 \\ 0.3 & -46.2 & -0.2 & -2.7 & 34.2 & -0.4 & -2.2 & 11.0 & -11.2 & 3.2 \\ 0 & 0.2 & -45.7 & -0.1 & -0.4 & -33.8 & -0.1 & -11.3 & -11.1 & 0 \\ -43.6 & -3.1 & 0 & -20.8 & 2.0 & 0 & -19.3 & 0.3 & -0.3 & 2.8 \\ 0.3 & -29.4 & 0.2 & -1.7 & -11.3 & 0.1 & 0.2 & 5.0 & -5.0 & 1.4 \\ 0 & 0.2 & 29.1 & 0 & -0.1 & -11.2 & 0 & 5 & 5 & 0 \\ 23.5 & 2.0 & 0 & 17 & 0.8 & 0 & -21 & -0.3 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11.8 \end{bmatrix} \tag{5.9}$$

$$K_b = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.8 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.8 \end{bmatrix}, \; K_d = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$K_c = \begin{bmatrix} 1.6 & -7..3 & 7.4 & -0.1 & 3.8 & 3.8 & -0.2 & 3.8 & 0 & 0.5 \\ -1.6 & 7.3 & 7.4 & 0.2 & -3.8 & 3.9 & 0.3 & 0 & 3.8 & -0.5 \\ -10.7 & -2.1 & 0 & -1.9 & 1.1 & 0 & 0 & 0.5 & -0.5 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The sensitivity, complementary sensitivity and controller gain function singular value plots S, T and KS respectively are shown in Fig: 5.9.

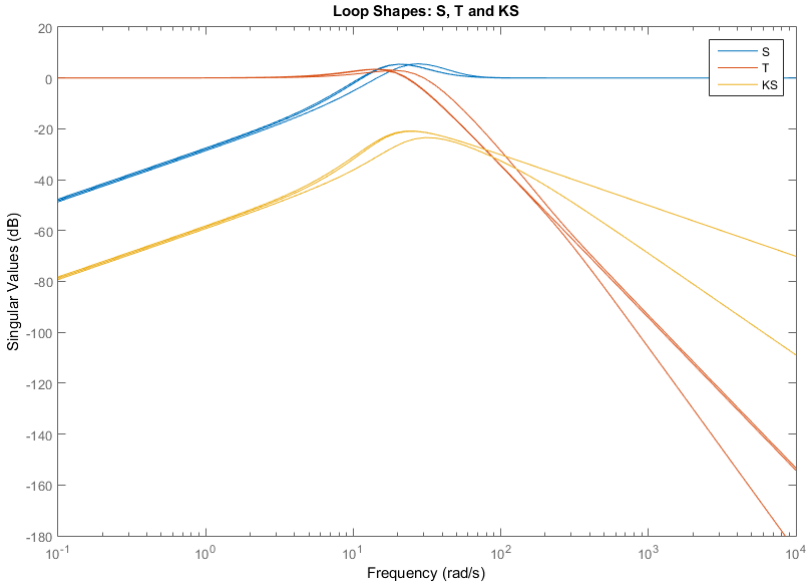


Figure 5.9: Controller 2: S, T and KS

It can be seen in the pole zero diagram in Fig: 5.10 that there is one pole at approximately 3 times the frequency of the highest frequency pole of the first controller. This could potentially be an issue in implementation.

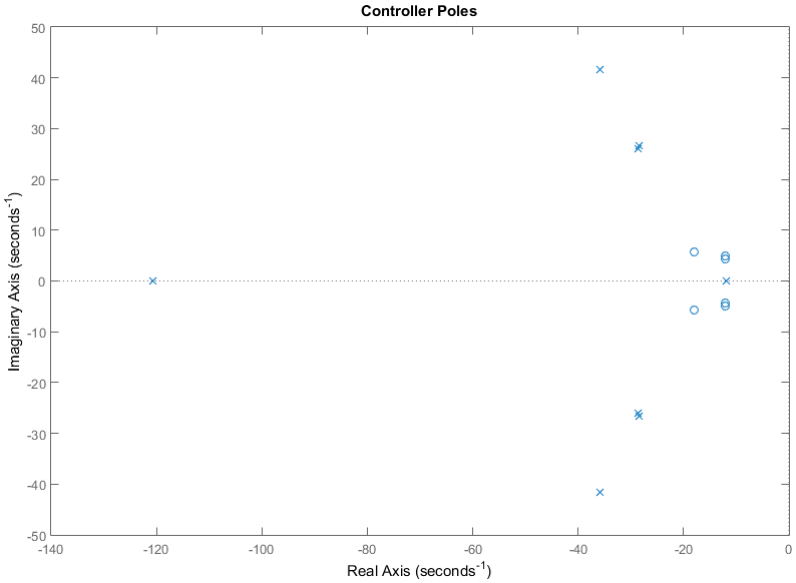


Figure 5.10: Controller 2: Pole Zero Diagram

The step response of the control system is shown in Fig: 5.11.

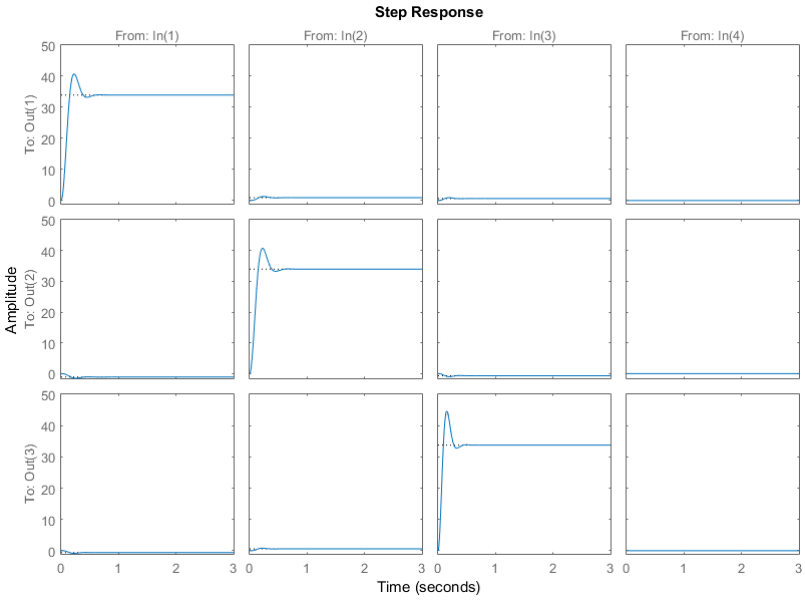


Figure 5.11: Controller 2: Step Response

Simulated Uncertainty

The robust control synthesis method here uses generalised uncertainty through the left coprime factorisation which is an advantage, but also makes it harder to visualise the robustness of the resulting controller. To make the robustness clearer an uncertain model is created with a modelled uncertainty of $\pm 10\%$ in the inertia coefficients I_{xx}, I_{yy}, and I_{zz}.

The uncertain coefficients are defined using ureal from the Robust Control Toolbox [41] which creates a real parametric uncertainty. The example shown is with a nominal value of 0.0028.

```
Ixx = ureal('Ixx',0.0028,'percentage',[-10  10]);
```

The uncertain model is created using a uss object. The rest of the model remains the same.

```
uG = uss(A,B,C,D)
```

The model can be verified by evaluating it at its nominal values. Alternatively Monte Carlo analysis can be used using usample to simulate the closed loop system with random values of the uncertain values within the given bounds.

```
CLP = feedback(uG.NominalValue,K);
CLP = feedback(uG,K);
step(usample(CLP,10))
```

A Monte Carlo analysis of the step response with 10 samples is shown in Fig: 5.12, and shows that the step response is relatively unaffected by the uncertainty.

```
step(usample(CLP,10))
```

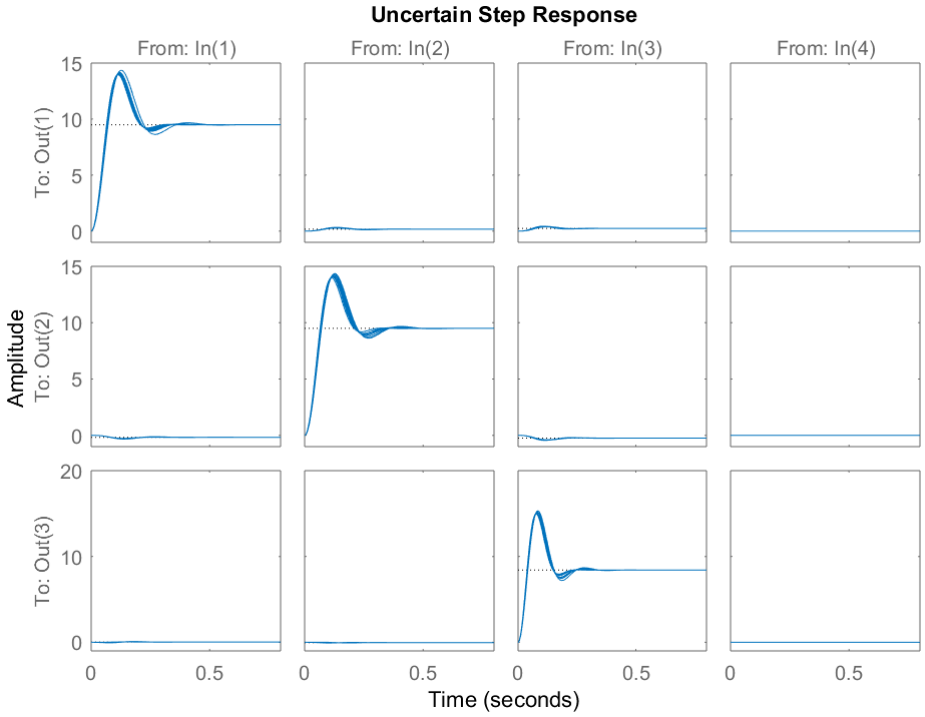


Figure 5.12: Controller 2: Uncertain Step Response

Applying the same technique to the singular value plot of the closed loop in Fig: 5.13 gives similar results.

```
sigma(usample(CLP,10))
```

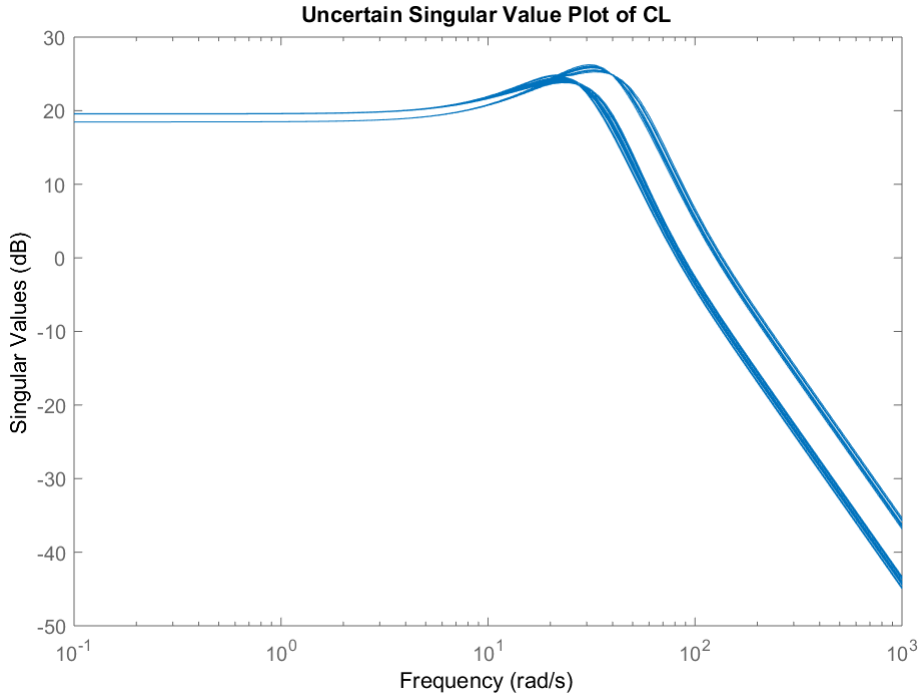


Figure 5.13: Controller 2: Uncertain Singular Value Diagram

As real perturbations were applied it is appropriate to apply Mu analysis. This is relatively straightforward in Matlab 2017 [41] requiring only the following line of code.

```
[stabmarg,wcu] = robstab(CLP);
```

This returns three values, the upper and lower bounds of Mu and the critical frequency of the system. In this case the frequency is infinite, and the upper and lower bounds of Mu are 10 and 7.1559 respectively. The lower bound on Mu represents the point least of robustness against the modelled uncertainty. The value of 7.2 corresponds to robustness against 720% of the modelled uncertainty. This can either be interpreted as a generous scenario or a very robust controller.

Simulink Simulation

A nonlinear simulation was used to evaluate the disturbance rejection performance of the controller. No uncertainty was modelled. In both cases an angular disturbance of $\frac{\pi}{16}$ was used with a time period of 4 seconds and a duty cycle of 10%. A diagram of the simulation configuration is shown in Fig: 5.14.

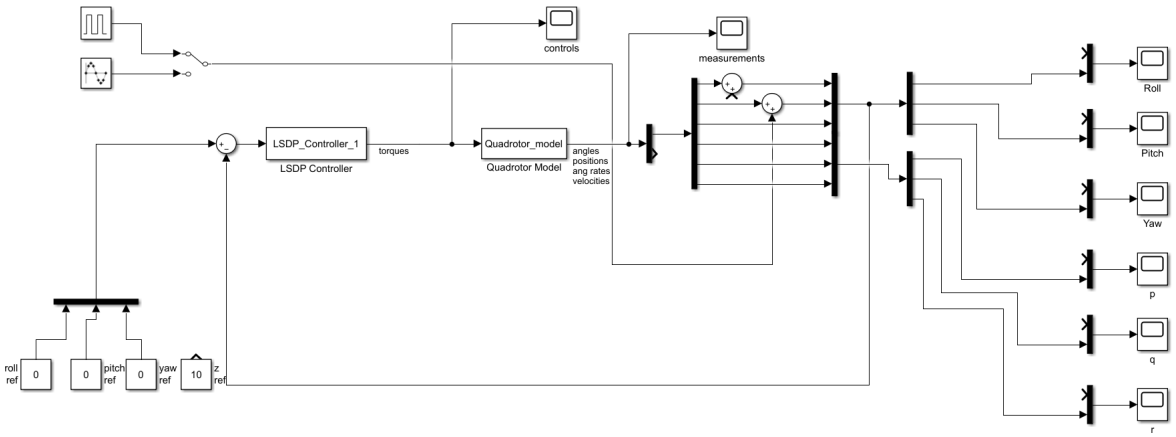


Figure 5.14: Controller 2: Simulink Simulation

The response to a roll disturbance is shown in Fig: 5.15.

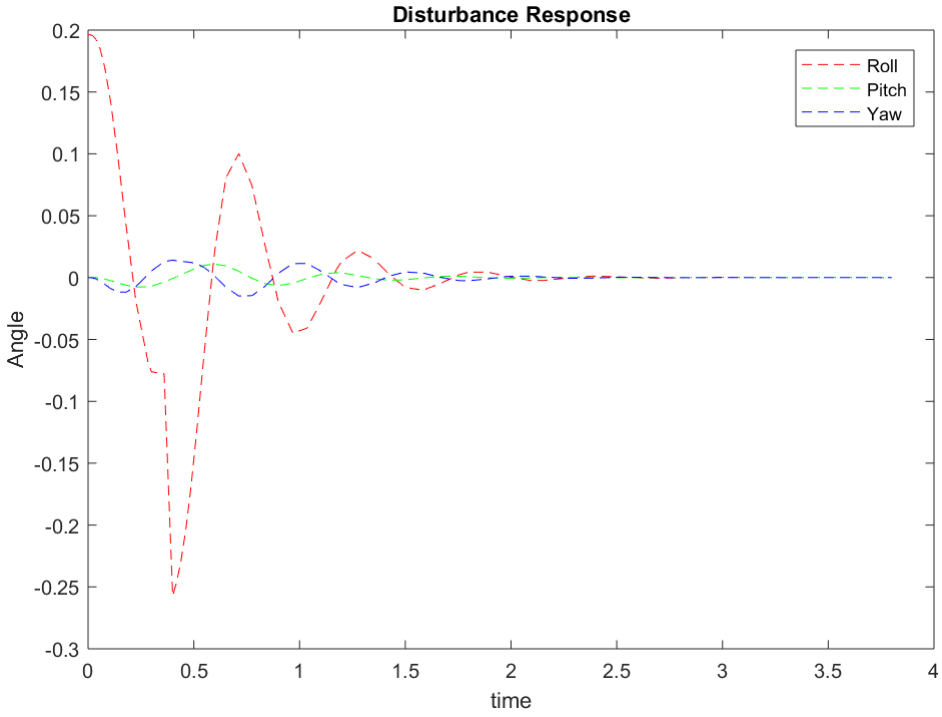


Figure 5.15: Controller 2: Disturbance Response to Roll

The response to a pitch disturbance is shown in Fig: 5.16.

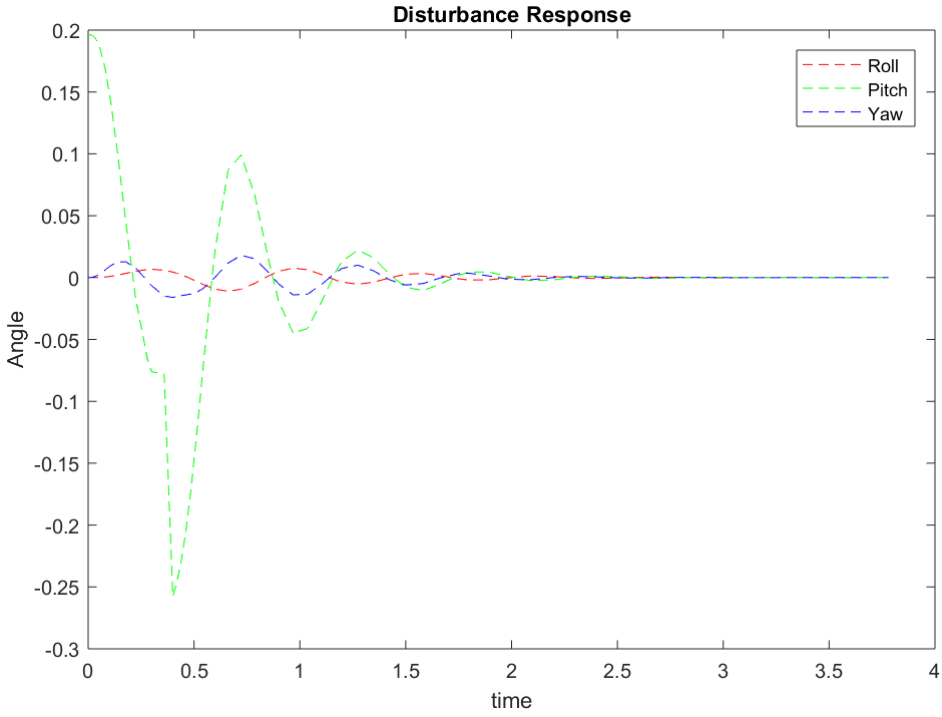


Figure 5.16: Controller 2: Disturbance Response to Pitch

5.4 Cont2: Actuator Addition

Although the exact actuator model isn't available it is possible to make an approximation. Using the controller 2 model above, it is augmented with a first order model with a cutoff frequency at 3 rad/sec. The frequency response is shown in Fig: 5.17.

$$G_{act} = \frac{1}{s+3} \tag{5.10}$$

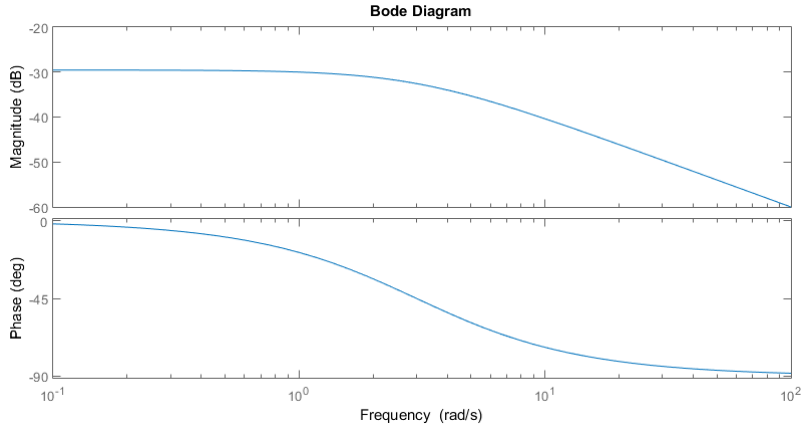


Figure 5.17: Bode plot of actuator

Open loop step singular values of the augmented model are shown in Fig: 5.18.

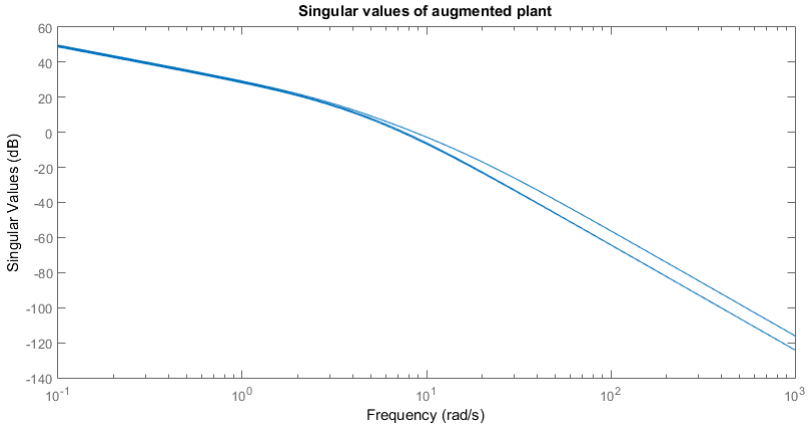


Figure 5.18: Open loop singular values of the augmented model

The robustness level was improved in synthesis from the original controller, with $\gamma = 2.0217$.

The addition of actuators limits the amount of high frequency actuation available to the controller by approximately 2 orders of magnitude. This has relatively little effect on performance. The step response is slower, but also more realistic. The addition of the actuators increases the order of the model, and so also increases the order of the controller. The means that the model order reduction techniques can no longer be applied to obtain a 9th order controller without seriously degrading the controller. The new controller is 13th order which is still acceptable, but does mean that the Simulation code would have to be adjusted, or the model order reduction set to reduce all controllers to 13th order.

The step response is shown in Fig: 5.19 and the singular values of S, T and KS are shown in Fig: 5.20. The S and T loops in Fig: 5.20 are around 2-3 rad/sec, this is approximatly 10 rad/sec lower in frequency than the original such as in Fig: 5.9.

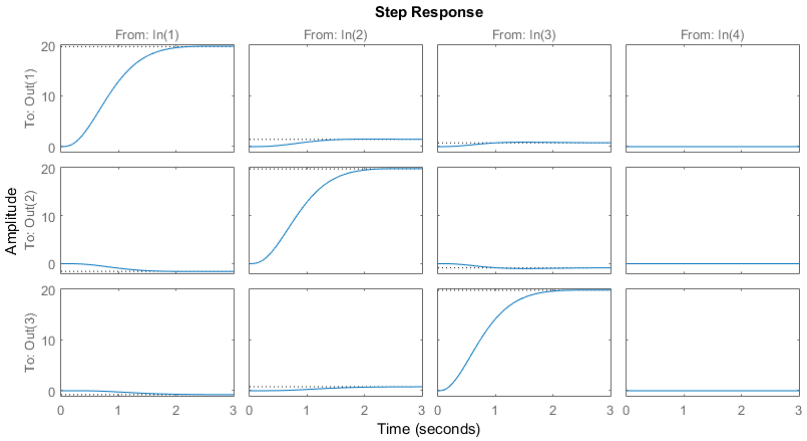


Figure 5.19: Closed loop step response

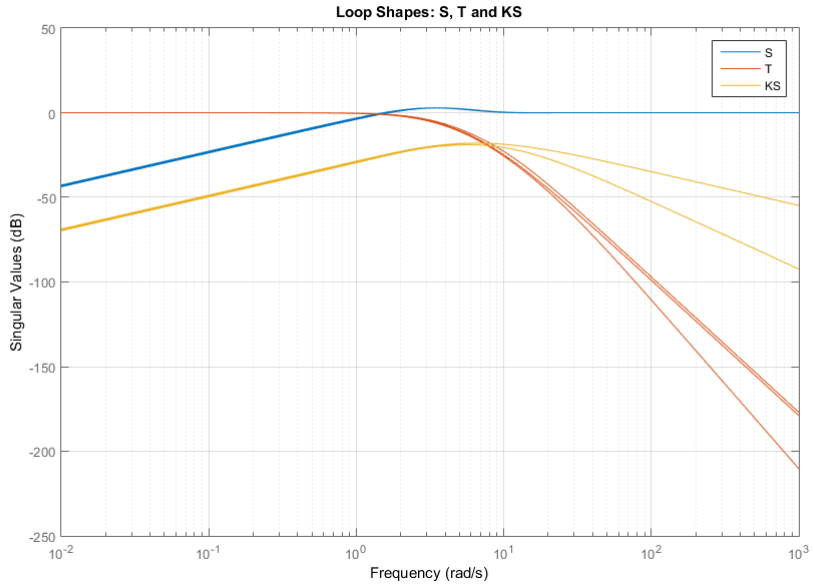


Figure 5.20: Closed loop singular values

5.5 Simulink Simulation

The nonlinear simulation is implemented in Matlab using two level-1 Matlab S functions, one for the model and one for the controller. The S functions are quite flexible and can be applied to both discrete and continuous systems. Each S function sets the initial conditions for its equations, and then at each iteration computes an update to the set of first order nonlinear state equations used to model the system. The controller S-function uses imported controller matrix values, and the aircraft model uses hard coded values. The controller matrices are generated by the controller synthesis functions. The size of the input and output vectors accepted by the block is defined by the initial conditions. These blocks can then be used any other Simulink object. They are combined with standard elements like gains, summers, scopes and signal generators to create the closed loop model.

An example of an S-function, the controller simulation block, is shown in Fig: 5.21.

```
function [sys,x0,str,ts] = LSDP_Controller_1(t,x,u,flag)
switch flag
    case 0
        %
        str = [];
        ts = [0 0];
        szs = simsizes;
        szs.NumContStates = 9;
        szs.NumDiscStates = 0;
        szs.NumOutputs = 4;
        szs.NumInputs = 3;
        szs.DirFeedthrough = 1;
        szs.NumSampleTimes = 1;
        sys = simsizes(szs);
        x0 = [zeros(9,1)];
    case 1
        sys = deriv_quad_LSDP(t,x,u);
    case 2
    case 3
        if exist('Kc','var') == 0
            Kc = importdata('Kc.mat');
        end
        if exist('Kd','var') == 0
            Kd = importdata('Kd.mat');
        end
       sys = Kc* x + Kd* [u(1) u(2) u(3)]';
    case 4
    case 9
end

end
function dxdt = deriv_quad_LSDP(t,x,u)
   if exist('Ka','var') == 0
        Ka = importdata('Ka.mat');
   end
   if exist('Kb','var')==0
        Kb = importdata('Kb.mat');
   end
   % new version with imported controller
   dxdt = Ka* x + Kb* [u(1) u(2) u(3)]';
end
```

Figure 5.21: S-Function for the Controller

Chapter 6

LPV Controller ▲

6.1 Introduction

The Matlab LMI toolbox provides a means to design gain scheduled $\mathcal{H}_\infty$ controllers through the hinfgs() command, but this is used with a signal based synthesis type configuration.

The objective I am trying to achieve is to use coprime factorised models, and to obtain a $\mathcal{H}_\infty$ controller for the coprime factorised LPV model, but Matlab provides no means to do this. I am free to choose any method I wish, however it would be desirable to leverage the LMI toolbox where possible. There are two approaches for the synthesis, one is to use the Tensor Polytopic (TP) or Affine (PD) models, the other is to extract instances of the Parameter Varying (PV) model at a given points and synthesise controllers for them. The former is technically better for stability proofs, allows smooth switching over the parameter space, but is more complex to implement both in system and for control synthesis. The latter has discontinuities on the parameter space, and is less elegant, but is also more in keeping with how most gain scheduling controllers are implemented in practice. Another particularly interesting advantage is that the ARE solutions can be used instead of LMIs in the latter case. Chen [24] took the former approach.

The process overview is

1. Create a Parameter Varying (PV) model
2. Extract instances (gridding)
3. Compute the solutions to the coprime factorisation at grid points
4. Form the CPF model using the global solution from (3)
5. Compute the $\mathcal{H}_\infty$ solution for each grid point
6. Form the $\mathcal{H}_\infty$ controller for each grid point
7. Implement the controller and test in closed loop with the respective model

6.2 Creating a Polytopic Model

A parameter dependent model was constructed using the LMI control toolbox [33]. This allows models to be constructed in Affine [24, p.27] (PD) form, and then be converted to tensor polytopic [24, p.28] (TP) "vertex" for computations.

To create an LPV model the parameter description must be constructed using the pvec function. This can either be created as a polytopic model or as affine using 'box'. The vector in the second argument defines the upper and lower bounds on the parameter, and the vector in the third argument defines the upper and lower bounds on the parameter rate.

pv = pvec('box',[Zmin Zmax]);

The model is then described as an **s** model. S_0 is the nominal model, and S_1 is the first parameter varying model associated with the first varying parameter. These are in affine form, but can be used in polytopic form. Only the first model is actual meaningful as an ltisys model, the others are just descriptions of the additional dynamics with respect to the particular varying parameter.

s0 = ltisys(A,B,C,D);

The final model is constructed using the parameter description pv, and the parameter models S_i.

pdG = psys(pv,[s0 s1]);

The affine model is then converted to polytopic form using the aff2pol command

tpG = aff2pol(pdG);

Information about the model can be obtained using minfo

minfo(tpG)
10 rows 28 cols: regular MATLAB matrix

6.3 Grid Based LPV

Grid Based LPV is one method to implement the coprime factorisation of the LPV plant. It was also the method used by Jianchi Chen [15]. I had originally intended to perform a coprime factorisation of the LPV model itself, but this proved to be very difficult, so I too resorted to gridding, though I started from a "true" polytopic model as a basis, whereas Chen [15] used discrete models. Also I used the built in Matlab tools for $\mathcal{H}_\infty$ synthesis as it is quicker.

The code blocks are numbered and this section follows them. The full code is given in Appendix C. The first step (1) was the creation of a polytopic model as per Chapter 6.2. The model used in this case was that from the "proof of concept-1" controller. There is a single varying parameter but the range of variation is very small to make the synthesis easier. The objective here is just to demonstrate the synthesis method works correctly.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0.99 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.99 & 0.99 \\ 0 & 0 & 0 & -7.14 & 0.60 & 0.60 \\ 0 & 0 & 0 & -0.6 & -7.14 & -0.60 \\ 0 & 0 & 0 & 0 & 0 & -18.18 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 109.28 & 0 & 0 \\ 0 & 109.28 & 0 \\ 0 & 0 & 278.18 \end{bmatrix} \tag{6.1}$$

$$C = \begin{bmatrix} 57.29 & 0 & 0 & 0 & 0 & 0 \\ 0 & 57.29 & 0 & 0 & 0 & 0 \\ 0 & 0 & 57.29 & 0 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.2}$$

```
Zmin=1; Zmax=4;
pv = pvec('box',[Zmin Zmax]);
s0 = ltisys(A,B,C,D);
s1 = ltisys([0 0 0 0 0 0; 0 0 0 0 0 0; 0 0 0 0 0 0; 0 0 0 0.9 0 0; 0 0 0 0 0.7 0; 0 0 0 0 0 0
0.5],... zeros(6,3),zeros(3,6),zeros(3,3),0);
pdG = psys(pv,[s0 s1]);
tpG = aff2pol(pdG);
minfo(tpG)
```

(2) The next step is gridding where the LPV model was then evaluated at discrete points around the parameter to space to create a surface of models. Another option was to just write the discrete models out directly, but this has the advantage that it can be used with the Matlab psys models. The density of the grid used depends on the resolutions at which the model is known. In this implementation the maximum and minimum values are extracted from the parameter vector model and divided by the resolution value.

The parameter description is known, but for completeness it is extracted from the LPV model. This is done using psinfo() with the 'par' parameter which returns a matrix describing the varying parameters and their variation rates. Columns 2 and 3 contain the minimum and maximum value of each parameter, and this is how the limits are determined for each parameter.

```
pvs = psinfo(tpG,'par');
```

Extracting the model at a given grid point is also not obvious. The model is polytopic, not affine. Any point must first be converted into polytopic form using polydec, with the first argument the parameter description, and the second argument the grid point in affine form. This can then be passed to psinfo, along with the argument 'eval' which evaluates the LPV model at that grid point. I didn't have to use the polytopic form of the model because I am writing my own code, however it is the preferable form.

gp = polydec(pv,gpd);
pdm = psinfo(tpG,'eval',gp);

The result, pdm, is an ltisys model. For any method other than gridding direct extraction of the polytopic models would be required which is done using pinfo with 'sys' instead, and an index corresponding to which polytopic model is to be extracted. This would be required if trying to compute a coprime factorisation of the LPV model through a smooth mapping. The -inf indicates to Matlab that it is not just a normal matrix.

pdm = psinfo(tpG,'sys',2);

An example of such a polytopic model is shown in Eq. 6.3.

sk = psinfo(tpG,'sys',1)

$$\begin{bmatrix}
0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -6.24 & 0.60 & 0.60 & 109.28 & 0 & 0 & 0 \\
0 & 0 & 0 & -0.60 & -6.44 & -0.60 & 0 & 109.28 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -17.68 & 0 & 0 & 278.18 & 0 \\
57.29 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 57.29 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 57.29 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -Inf
\end{bmatrix} \tag{6.3}$$

(3) Once the model grid has been extracted the pre and post weights for the $\mathcal{H}_\infty$ loopshaping procedure have to be applied. This is just a series connection of the models, but first the components (A,B,C,D) have to be extracted and a new bigger matrix created for the result as the augmented matrices are larger. The weights applied are the same for all grid points. The B, C and D matrices from the final iteration are used as the fixed matrices for all grid points. It is a requirement in LPV synthesis that those matrices be fixed in order for a solution to exist as otherwise an infinite number of LMIs would have to be solved.

(4) The coprime factorisation. In this step an LMI is created for each model, one final LMI is created as a constraint on the solution variables, optimisation is attempted and the answer extracted. The solution results in two values, P and Z, each of which is a vector. This step does not actually create the factorisation, it just solves an $\mathcal{H}_2$ optimisation problem which gives values for a formula which gives the factorisation. This is much the same as in the Algebraic Riccati Equation method. More detail is now given about the code for the solution. Note that the code is from Chen's [15] code block.

LMI code begins with setlmis() which creates a new LMI system

setlmis([]);

LMIVar is used to define the form of the solution being sought, in this case a column vector with as many rows as the B matrix. The 1 in the first argument indicates that the solution has to be a symmetric block diagonal matrix, zero, or scalar.

P=lmivar(1,[rowp 1]);
Z=lmivar(1,[rowp 1]);

The first LMI set, one for each model, is then created. A new LMI is defined using newlmi, and then the LMI is constructed using lmiterm. The first argument to lmiterm is a vector, the first element indicates the LMI to which the term belongs, elements two and three indicate the row and column number in the matrix, and the fourth term indicates the LMI variable. The other arguments define the left side, constants, the right hand side, and any special functions. Only the lower or upper triangular part of the LMI has to be described because as they are Linear Matrix Inequalities, they are also symmetric.

The most first lmiterm is the most interesting. This indicates the variable P, the constant A from the model at index i, and 's' for the symmetrisation of the expression.

$$PA + A^T P \tag{6.4}$$

lmiterm([jj,1,1,P], 1,A(:,:,i),'s');

The second lmi term adds $-C^T C$ to the end of the first term, with no solution variable involved, leading to the following expression

$$PA + A^T P - C^T C \tag{6.5}$$

lmiterm([jj,1,1,0], -C'*C);

The third and fourth terms put the following expression into the matrix element (2,1)

$$B^T P - D^T C \tag{6.6}$$

lmiterm([jj,2,1,P],B',1);
lmiterm([jj,2,1,0],-D'*C);

The fifth term puts the -R term into the LMI element (2,2)

lmiterm([jj,2,2,0], -R);

The second LMI sets a condition that Z and P be positive. This is achieved by using lmiterms to put Z and P on the diagonal and I on the off-diagonal. The negative in the first element of the first argument to lmiterm causes the Z to have to be greater than or equal to 0.

$$\begin{bmatrix} Z & I \\ I & P \end{bmatrix} > 0 \tag{6.7}$$

lmiterm([-jj 1 1 Z],1,1);
lmiterm([-jj 2 1 0],1);
lmiterm([-jj 2 2 P],1,1);

For completeness, the full LMIs to solve the $\mathcal{H}_2$ problem are given in Eq. 6.8.

$$\begin{bmatrix} PA_i + A_i^T P - C^T C & PB - C^T D \\ B^T P - D^T C & -R \end{bmatrix} < 0,\ i = 1 \dots r \tag{6.8}$$

$$\begin{bmatrix} Z & I \\ I & P \end{bmatrix} > 0 \tag{6.9}$$

Once the LMIs to be solved have been defined the solution variables are extracted from the LMI description using mat2dec and the solver mincx is called. The Matlab LMI toolbox [33] has three generic solvers, gevp for solving generalised Eigenvalue problems, feasp for solving feasibility problems, and mincx for solving convex minimisation problems with linear constraints. Mincx has 3 arguments used, the first is the LMI system, the second is the solution variables, and the third is an options vector for (1) relative accuracy, (2) a limit on the number of iterations, and (4) a more optimal stopping condition.

lmisys=getlmis;
c = mat2dec(lmisys,zeros(rowp), eye(rowp));
options = [1e-2 200 1e5 10 0]
[copt,xopt] = mincx(lmisys,c,options)

(5) Now that the solutions have been obtained the coprime factorisation formula can be applied. The solution values have to be extracted from the decision variable form used by the convex solver using dec2mat.

Popt=dec2mat(lmisys,xopt,1);

Zopt=dec2mat(lmisys,xopt,2);

The value of Popt is then used in the formula for L, which enters the formula for the coprime factorisation. B, C and D are from the model, and Rt is based on D.

LL=-(B*D'+inv(Popt)*C')*inv(Rt);

The coprime factorisation is then formed by iterating through all of the models

```
Acop=A(:,:,i);
Bcop=[-LL*sqrt(Rt) B(:,:)]
c1=[zeros(size(C));C(:,:)];
c2=C(:,:);
Ccop=[c1;c2];
d11 = [zeros(size(D)); sqrt(Rt)];
d12 = [eye(size(D)); sqrt(Rt)];
d21 = sqrt(Rt);
d22 = D(:,:);
Gcop(:,:,i)=pck(Acop,Bcop,Ccop,Dcop);
```

The resulting model has the form of Eq. 6.10, hence the subscripts on B, C and D terms.

$$G_{cpf} = \left[\begin{array}{c|cc} A_i & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] \tag{6.10}$$

The formula for the coprime factorisation is given in Eq. 6.11.

$$G_{cpf} = \left[\begin{array}{c|cc} A_i & L \cdot R^{\frac{1}{2}} & B_2 \\ \hline \begin{bmatrix} 0 \\ C \end{bmatrix} & \begin{bmatrix} 0 \\ R^{\frac{1}{2}} \end{bmatrix} & \begin{bmatrix} I \\ R^{\frac{1}{2}} \end{bmatrix} \\ C_2 & R^{\frac{1}{2}} & D \end{array}\right] \tag{6.11}$$

(6) Synthesise a $\mathcal{H}_\infty$ controller using standard tools

The final step is to solve the controller problem. In the case of Chen [24] he wrote a set of LMIs to solve the $\mathcal{H}_\infty$ control problem with pole placement constraints, but this project is taking a different approach. As the models are in a suitable form a standard LMI $\mathcal{H}_\infty$ control solver can be called on it. The weights have already been included earlier so signal based weights do not have to be applied. Two control design methods were used. The first used the hinflmi() function which synthesises a standard $\mathcal{H}_\infty$ controller, and the second h2hinfsyn() synthesises a mixed $\mathcal{H}_\infty$ controller with pole placement constraints.

These are detailed further in the subsections. In both cases this is iterated over all of the models and the controller and optimal γ values are collected.

```
r = [3 3];
[gopt1,K1] = hinflmi(P,r);
```

6.3.1 Standard $\mathcal{H}_\infty$ Controller

The standard $\mathcal{H}_\infty$ controller was used initially to help verify that the coprime factorised model was correct, and was then developed into a full controller.

The controller was synthesised using hinflmi() which uses LMI methods internally. It takes two parameters, the first is the model of the system in the form of an ltisys model, and the second is a vector containing the size of the measurement and control vectors y and u respectively. It returns the controller and robustness level γ, and optionally the solutions to the two Riccati equations. The objective is similar to that used for the proof of concept controller in that it attempts to minimise the infinitive norm, but uses signal based techniques instead of coprime factorisation. Coprime factorisation has already been applied though, but this was done in such a way that the the result was a ltisys form model which is compatible with the signal based techniques.

The first round of results considers the nominal model with just parameter 1, so the smallest possible LPV component, and as close to nominal as possible. The loop shapes S, T and KS in Fig: 6.1 all have suitable responses with S high pass, T low pass, and KS largest around the frequency of operation.

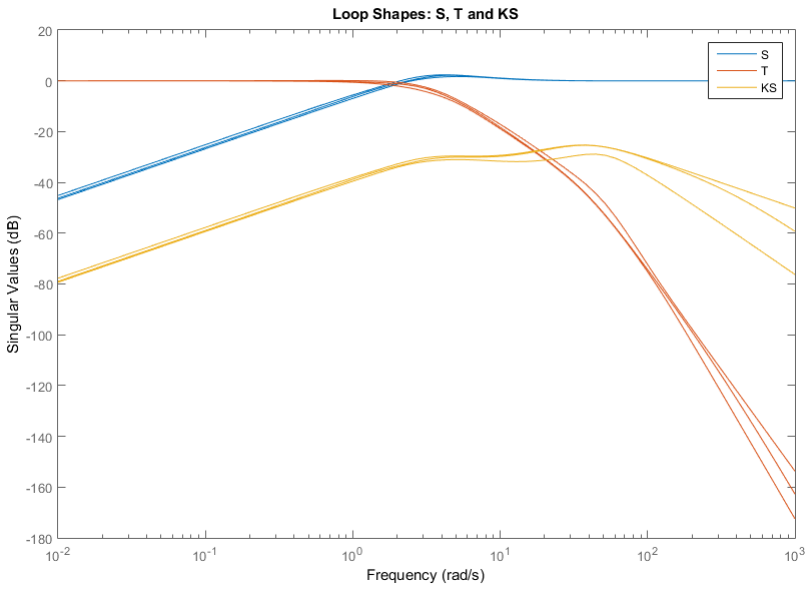


Figure 6.1: LPV Controller: S, T and KS

The step responses are much slower than those in the proof of concept $\mathcal{H}_\infty$ controller, but instead have very little overshoot and display an approximately first order response. Coupling between the channels continues to be minimal. The step response is shown in Fig: 6.2

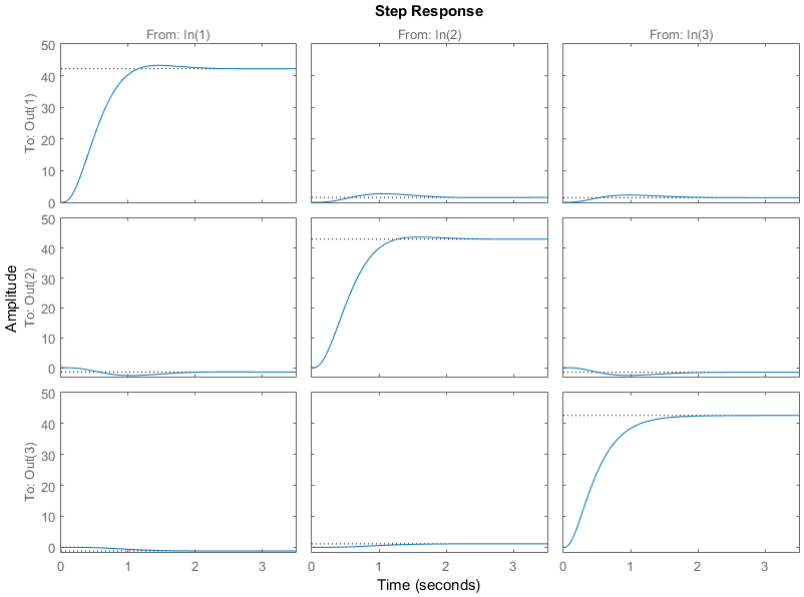


Figure 6.2: LPV Controller: Step Response

The full pole zero plot in Fig: 6.3 (left) shows that there is a very high frequency pole placed at 80000 rad/sec. This is a common side effect of using $\mathcal{H}_\infty$ LMI control techniques as the specifications are met with the use of unrealistic pole placement. A pole at such a high frequency is clearly impossible implement. The pole zero plot in Fig: 6.4 (right) focuses on the remaining poles, and this too shows a quite high frequency pole at 360 rad/sec, with the other poles placed well below 100 rad/sec which is much more realistic for implementation. One option would be to disregard these high frequency poles, and this is probably a viable option, although in the second controller a better approach will be taken.

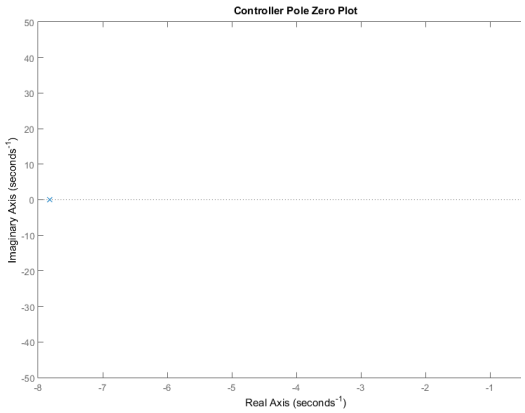


Figure 6.3: LPV Controller: All Poles

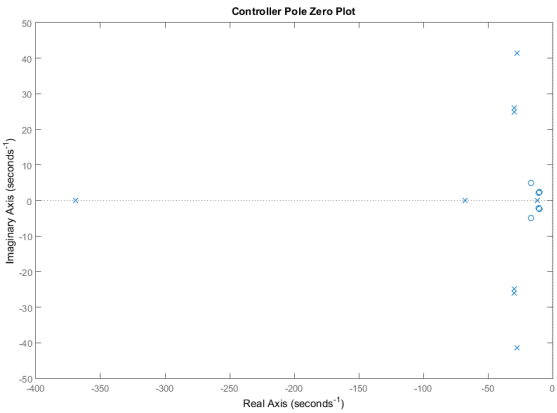


Figure 6.4: LPV Controller: Low Frequency Poles Only

The controller gain plot in Fig: 6.5 shows that control usage is focused around the frequency range of operation. The main purpose of this was as a quick way to visually compare the LPV controller with the proof of concept $\mathcal{H}_\infty$ controller.

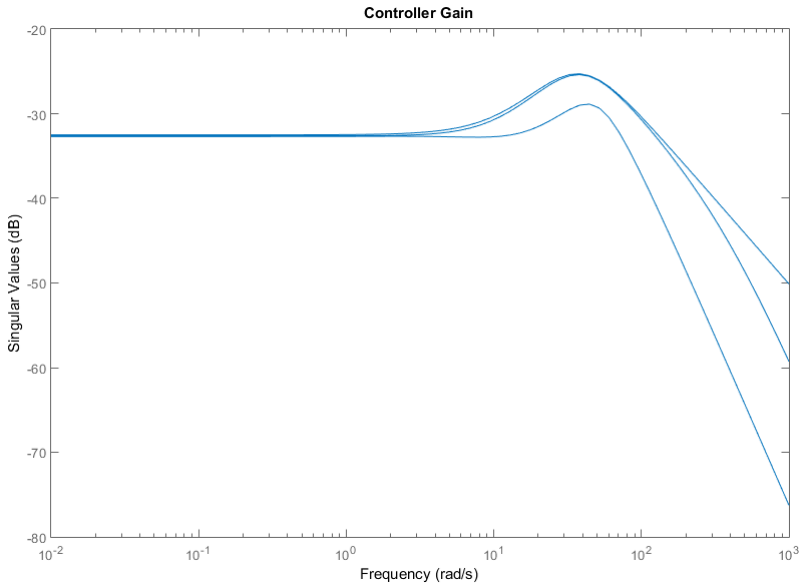


Figure 6.5: LPV Controller: Controller Gain

The second round of results considers the full range of parameters, and controller are synthesised for each parameter model. Fig: 6.6 is of the γ values for each parameter. The γ values increase as the parameter increases, which indicates that as the model departs further from the nominal model the robustness becomes worse, but the difference between the upper and lower values is relatively negligible.

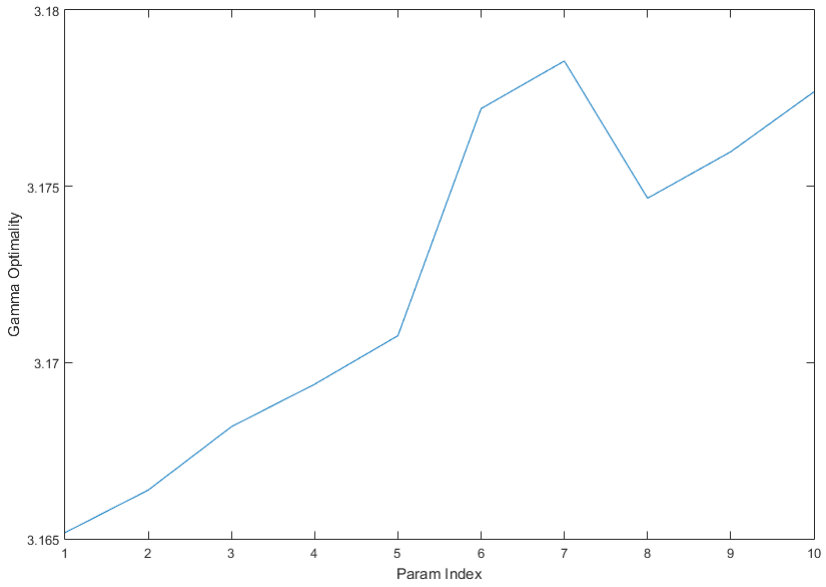


Figure 6.6: LPV Controller: γ Optimality vs Parameter Index

The following plots are generated for all of the LPV parameters, and so each i'th controller is paired with the i'th gridded model. A loop iterates using the following code with hold on and hold off used to update the graph.

```
for i=1:size(A,3)
        step(CL)
        hold on
        end
hold off
```

The S, T and KS loops are shown in Fig: 6.7 for all parameters. The very high frequencies after 10^4 are not relevant, and in some of the other plots have been removed. The variation between the singular values of KS for each parameter (orange) are noticeable below 10^1 and are likely due to the increasing amount of the LPV component that is included. Some difference can also be seen in the loop shape of T (red). S (blue) is quite consistent, except around 3 rad/sec.

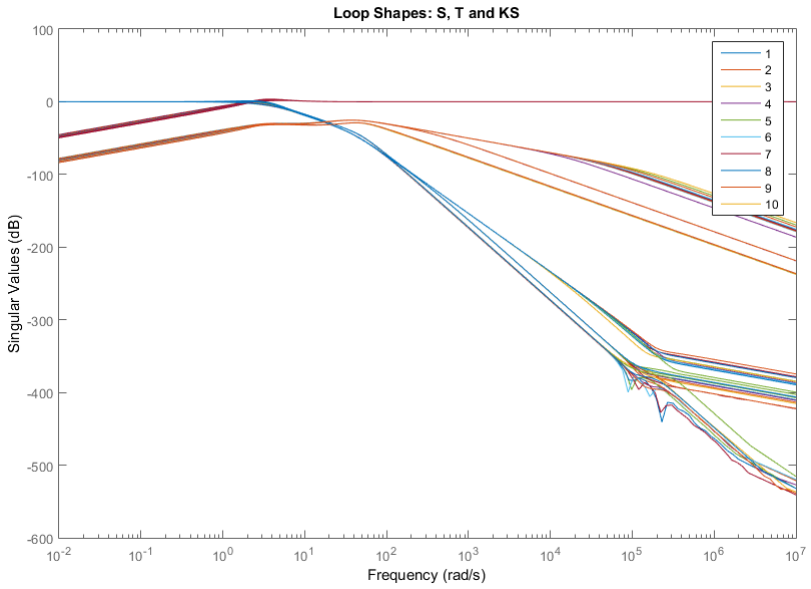


Figure 6.7: LPV Controller: S, T and KS for all Parameters

The multiple step response plot in Fig: 6.8 shows significant variation between the different operating parameters, with the final value varying between 42 and 52.

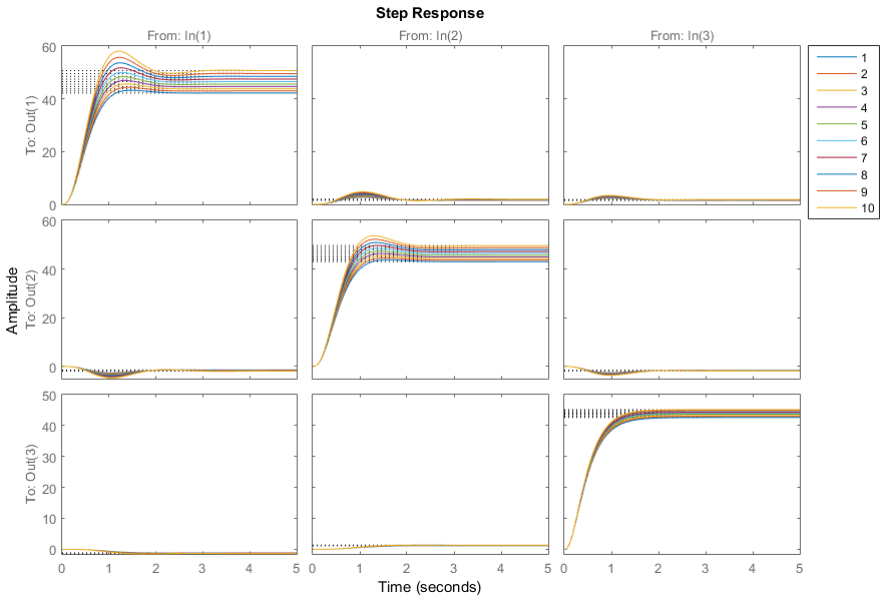


Figure 6.8: LPV Controller: Step Response for all Parameters

The controller gain plot in Fig: 6.9 doesn't have any significant differences except at higher frequencies which are not meaningful anyway.

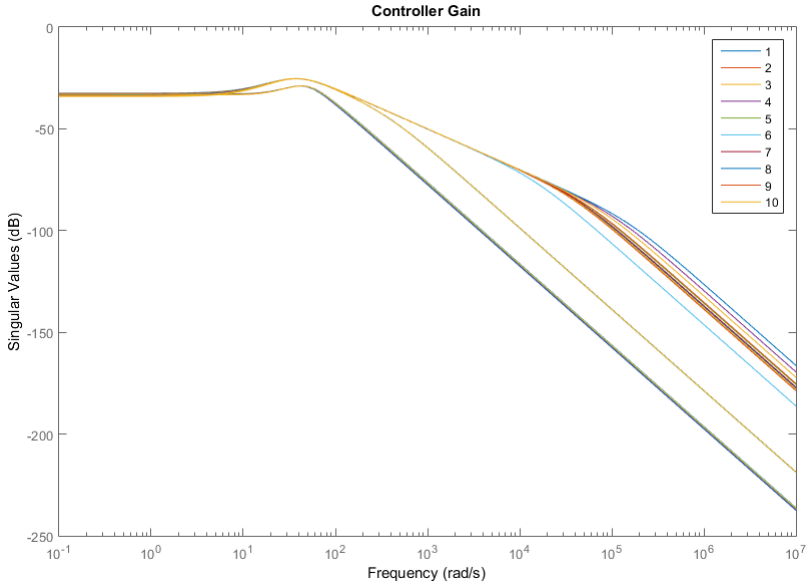


Figure 6.9: LPV Controller: Controller Gain for all Parameters

6.3.2 $\mathcal{H}_\infty$ Controller with Pole Placement Constraints (Mixed Synthesis)

The $\mathcal{H}_\infty$ with pole placement controller was synthesised using hinfmix [33, p.98]. This is similar to hinflmi in that it uses LMIs to solve the control problem, but is significantly more advanced and uses additional LMIs to implement the $\mathcal{H}_2$ and pole placement constraints [33, p.100]. These are optional features and are selected as part of the synthesis procedure. Although it is not used here, $\mathcal{H}_2$ control is a method for optimal control and is best described as the frequency domain counterpart of the linear quadratic control problem which is solved using AREs. It's main purpose in control is to find a stabilising controller which minimise a performance index typically to minimise control usage and eliminate deviation from some equilibrium value.

Four parameters are passed to hinfmix. The first is the ltisys plant. The second is a three element vector containing the dimension of the $\mathcal{H}_2$ outputs, the number of measurements and the number of control inputs. The third argument defines which norms are to be used to measure the the control optimisation objectives. The fourth argument defines an LMI region on the complex plane.

```
r = [0 3 3];
hinfmix(P,r,obj,region1)
```

The vector in the third argument has four elements. The first two define whether the $\mathcal{H}_\infty$ or $\mathcal{H}_2$ norms respectively are to be used, and if both then what weighting should be given to each. The last two elements define an upper bound on the value of each norm

during synthesis. If $\mathcal{H}_2$ specifications are included then the first element of the r vector must indicate how many outputs are to be optimised using $\mathcal{H}_2$. These will be the the highest subscripts of the C matrix. In this case only $\mathcal{H}_\infty$ was used so the vector had the following form

obj = [0 0 1 0];

If both $\mathcal{H}_2$ and $\mathcal{H}_\infty$ objectives are specified then the plant format is effectively placed in the format of the diagram in Fig: 6.10 [33]. Note that the optimisation is from w to z in all cases. The $\mathcal{H}_\infty$ norm is used on $w \rightarrow z_\infty$, and $\mathcal{H}_2$ norm on $w \rightarrow z_2$.

$$\begin{bmatrix} \dot{x} \\ zinf \\ z2 \\ y \end{bmatrix} = \begin{bmatrix} A & B_1 & B_2 \\ C_i & D_{i1} & D_{i2} \\ C_2 & D_{21} & D_{22} \\ C_y & D_{y1} & D_{y2} \end{bmatrix} \begin{bmatrix} x \\ w \\ u \end{bmatrix} \tag{6.12}$$

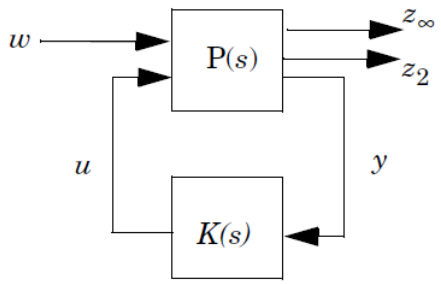


Figure 6.10: $\mathcal{H}_2/\mathcal{H}_\infty$ Mixed Synthesis Control

Pole placement constraints are used to limit where the poles can be placed on the complex plane, and are used here to eliminate high frequency poles. The constraints are defined using a command line tool in Matlab called lmireg in the following manner.

region1 = lmireg

The following menu is then displayed
Select a region among the following:
h) Half-plane
d) Disk
c) Conic sector
e) Ellipsoid
p) Parabola
s) Horizontal strip

m) Matrix description of the LMI region

q) Quit

In this case a disc was placed on the complex plane centred at $(0, j0)$ with radius 200, permitting poles to be placed at up to 200 rad/sec. This resulted in the following LMI region description

$$region = 1.0e2 \cdot \begin{bmatrix} -2 + j0.02 & 0 & 0 & 0 \\ 0 & -2 & 0.01 & 0 \end{bmatrix} \tag{6.13}$$

After 34 iterations the algorithm converged to $\gamma = 3.656185$ in under 1 second. The resulting controller is a 13x13 matrix.

Full details of the LMIs used by the functions can be found in the LMI Control Toolbox documentation [33].

The following plots show that the $\mathcal{H}_\infty$ controller with pole placements performs very similarly to the controller without the constraints, but the benefit of not having high frequency poles present. The first round of results considers the nominal model with just parameter 1, so the smallest possible LPV component, and as close to nominal as possible. As a result these controllers are much more suitable for implementation than the standard $\mathcal{H}_\infty$ controllers. However, both sets of controllers are highly developmental and depend on a very experimental coprime factorisation method so should still be used with caution.

Figure 6.11 shows the loop shapes of S, T and KS.

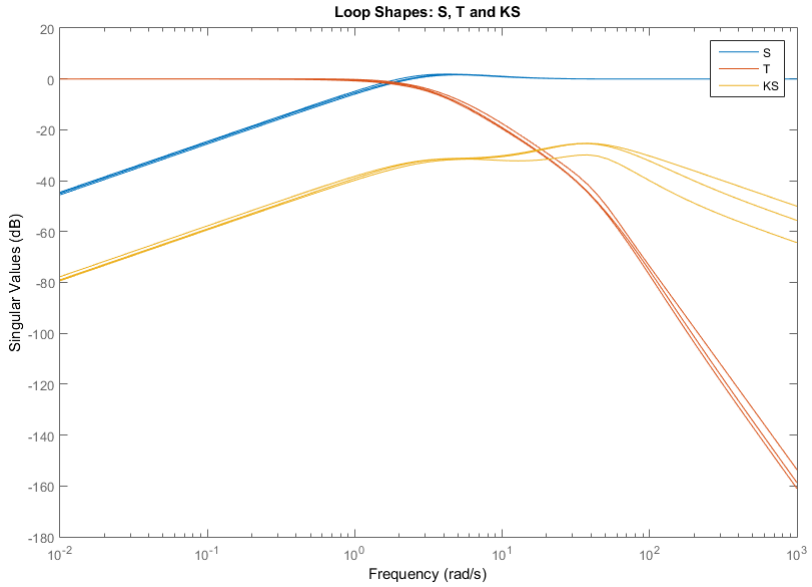


Figure 6.11: LPV Controller with PPC: S, T and KS

Fig: 6.12 shows the step response.

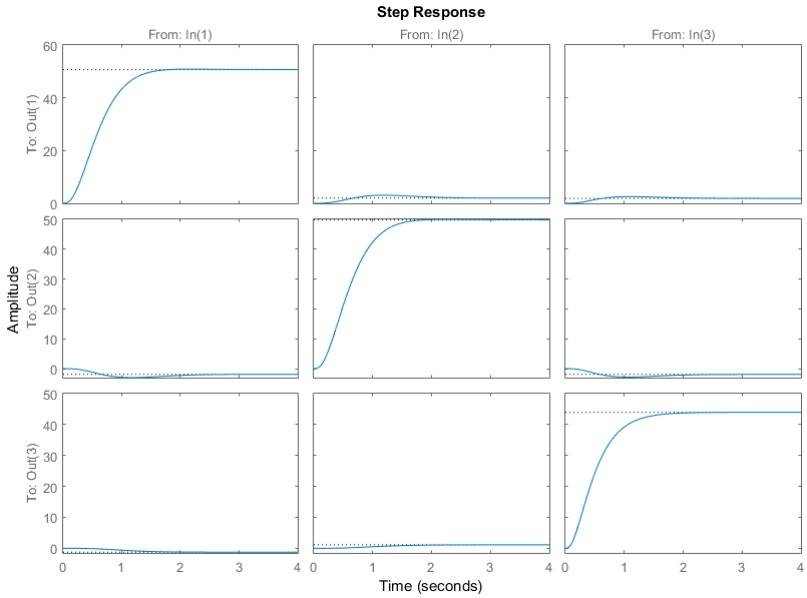


Figure 6.12: LPV Controller with PPC: Step Response

As can be seen in Fig: 6.13, the poles have been contained within the 200 rad/sec circle.

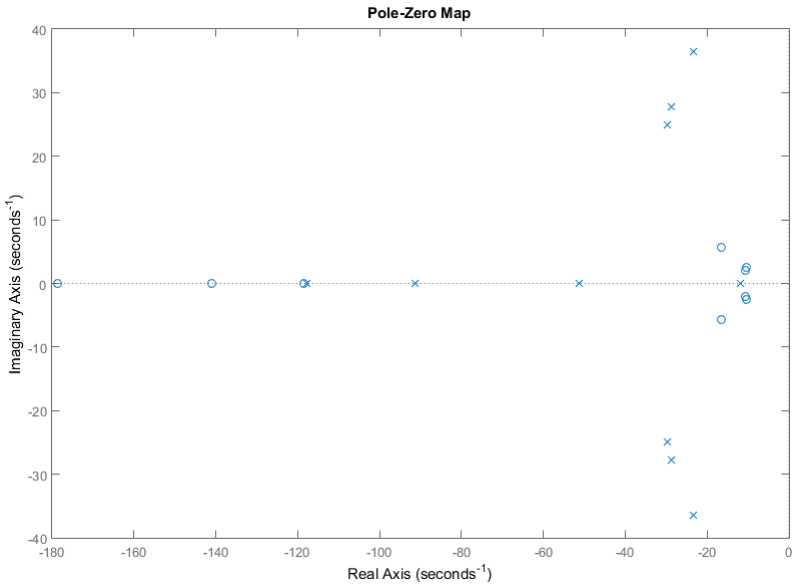


Figure 6.13: LPV Controller with PPC: Pole Zero Diagram

Fig: 6.14 shows the controller gains.

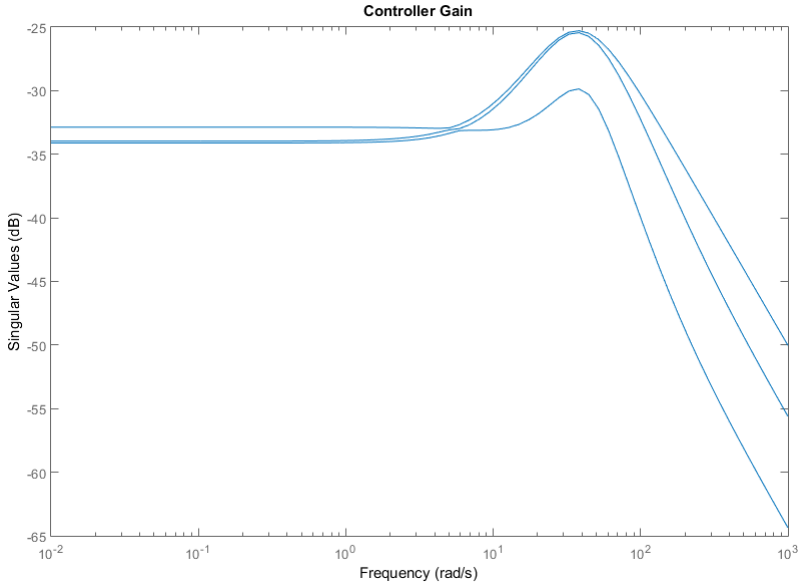


Figure 6.14: LPV Controller with PPC: Controller Gain

The second round of results considers the full range of parameters, and controllers are synthesised for each parameter model. The γ values for the different parameter values are shown in Fig: 6.15.

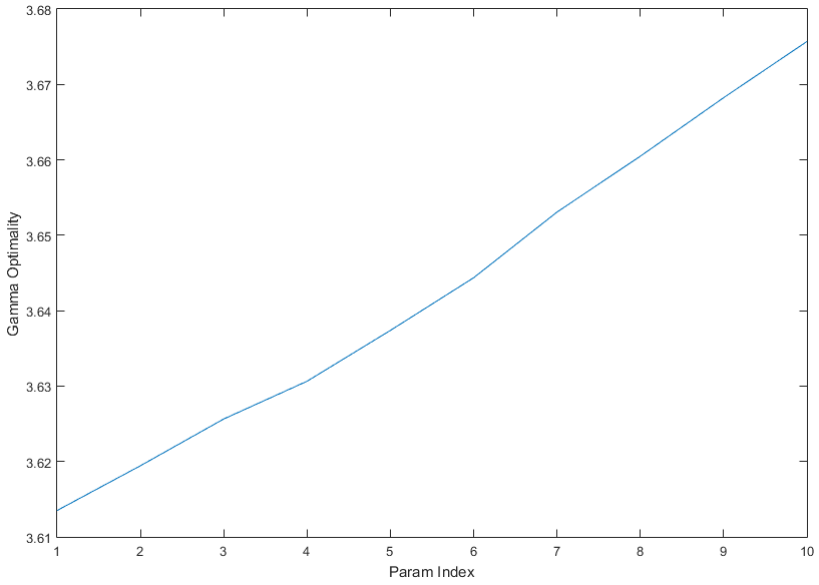


Figure 6.15: LPV Controller with PPC: γ vs Parameter Index

Fig: 6.16 shows the singular value plots of S, T and KS for the closed loop system at each parameter value.

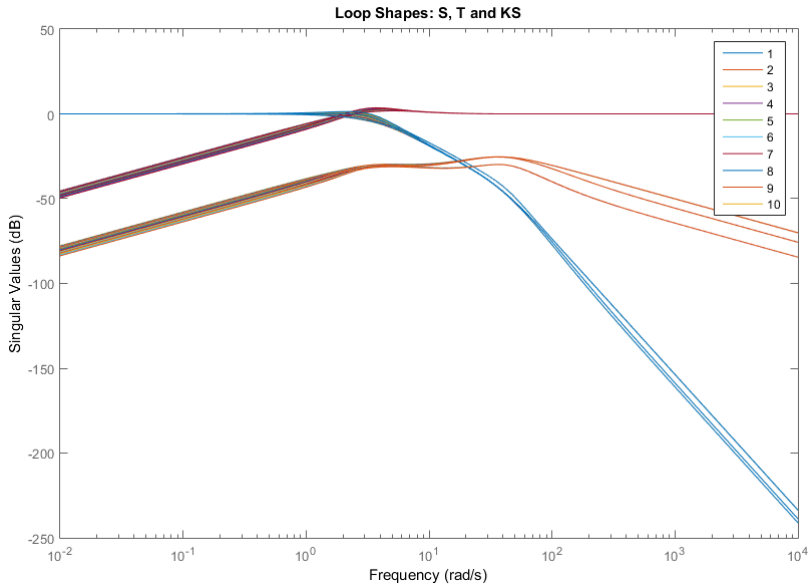


Figure 6.16: LPV Controller with PPC: S, T and KS for all Parameters

Fig: 6.17 shows the step response of the closed loop system at each parameter value.

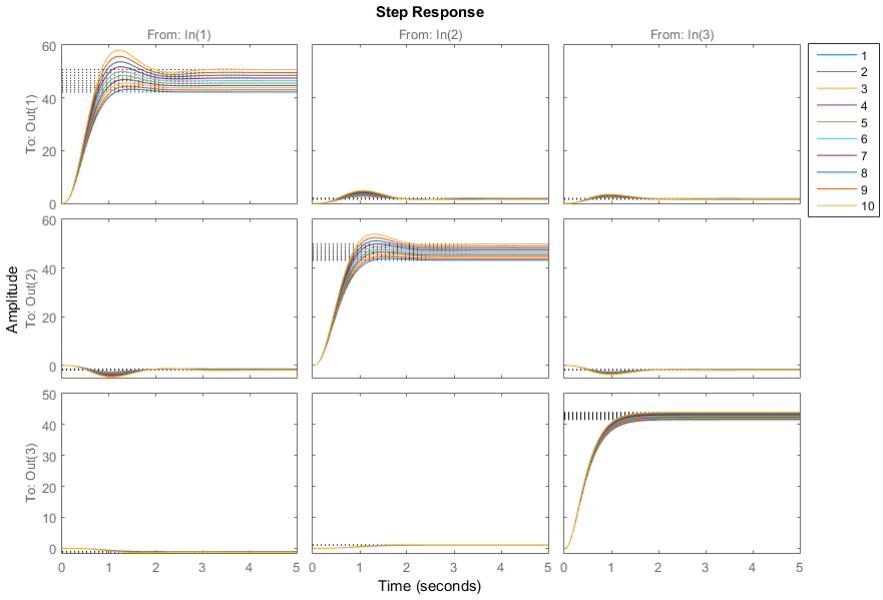


Figure 6.17: LPV Controller with PPC: Step Response for all Parameters

Fig: 6.18 shows the controller gain for the different parameter values.

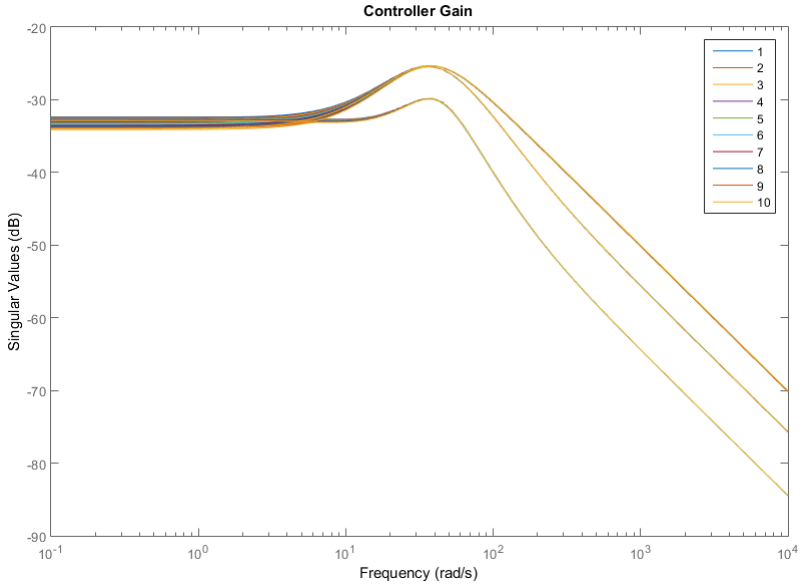


Figure 6.18: LPV Controller with PPC: Controller Gain for all Parameters

Chapter 7

System Implementation

7.1 Python Program

To do real world tests, and to do good quality system identification a real world implementation of the control system is required. The basis for this is a Raspberry Pi 3 [36] as the CPU. This was chosen as it is readily available, relatively inexpensive, has I2C and SPI connections, and supports a wide range of programming languages as it is Linux based. It runs a copy of Raspbian.

A first some attempt was made to do this implementation in Java, using J4Pi [37] which is a low level API for the Raspberry Pi. The motivation for this is that Java includes a library called Swing which makes nice looking GUIs. I am more familiar with it having used it at undergraduate level modules. There seems to be some limitation in J4pi which limits the update rate, which means that the IMU sample frequency is too low to be useful to me. This could be related to the number of system all permitted by Linux in a given time period.

The alternative is to use Python. This has low level APIs and matrix mathematics packages and so is a suitable alternative to Java. Two external libraries were used, one for the PCA9685 servo controller, and RTIMULib2 [38] which is a IMU interfacing and processing library with a built in sensor fusion filter [39]. The main benefit of this approach is that it delivers working code faster. Although I had never used Python before I was able to get an outline demo working with each component using an old board very quickly. There was a problem though, the old board used an MP9250, but the newer boards all used more recently bought IMUs, and they all seemed to be MPU9255 IMUs. The RTIMULib2 library, in Python form, does not seem to support that IMU. I tried modifying the source and rebuilding both the C and Python versions from source but the problem persisted. There was not enough time to source MPU9250 and put those on spare PCBs.

This left me with no alternative but to write my own IMU interface library. The first

part of this is setting up the IMU, the second part is reading the measurements. The first part involves working through the register documentation, calculating necessary values for filters and other registers, setting the registers, and then checking some to verify that the setup was correct. The second part involves reading the values, putting them into the correct byte order, and then converting from twos complement format represented as an unsigned integer into a signed integer. I couldn't find any library to do the conversion, so that also had to be written and tested. Finally conversion factors had to be applied to obtain units. The next step would be to apply sensor fusion but there was insufficient time to do this.

I then by chance found an MPU9250 IMU when at home and attempted to use that with RTIMULib2 but the latency was found to be too great at approximately 1.5 seconds to yield useful measurements for control and system identification. Appendix D shows the Python code using RTIMULib2 and the associated flight control and system and identification functions which were implemented.

7.2 Hardware

The PCB shown in Fig: 7.1 supports SPI MPU9250 IMU (orange), I2C MPU9250 (red), and 4 laser rangefinder / PCA9685 servo controller I2C connections (blue). The Raspberry Pi 3 [36] is mounted below. IMUs require a stable voltage source, this is provided by an LC filter (purple) with a large time constant formed by an inductor and a large capacitor on the IMU side. The microcontroller, an ATMega328P, measures RC signals and converts them into binary but this hasn't been used.

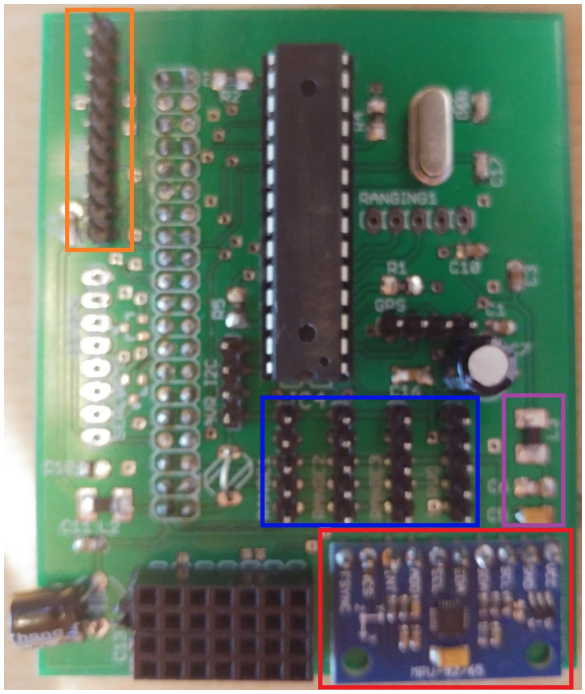


Figure 7.1: New Version of the FCS PCB

Aircraft The aircraft used for testing is shown in Fig: 7.2. It's purpose is to lift small polystyrene cubes representative of boxes. Although one obvious application is in light logistics, it could also be used as a mothership for swam robots. The mixed ground and air modes combine the efficiency of ground travelling with the obstacle avoidance of flight.

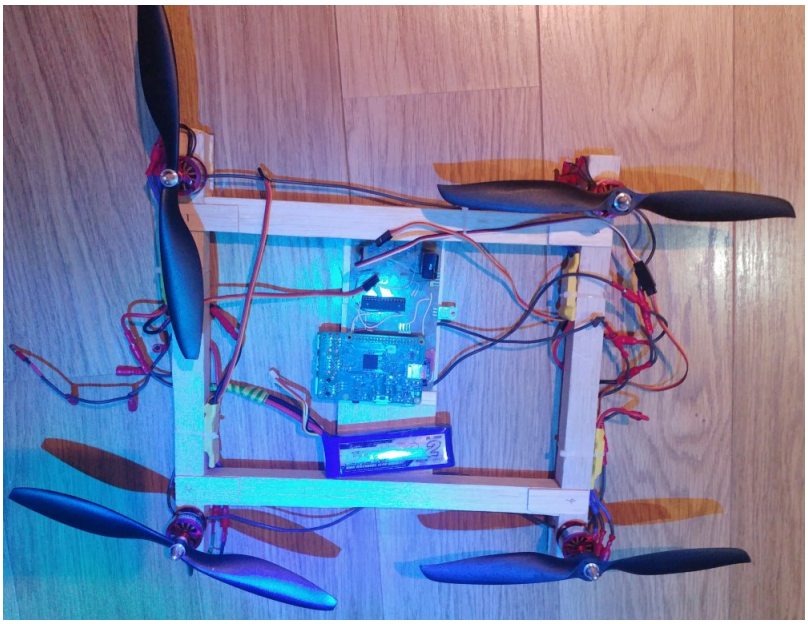


Figure 7.2: Demonstration Aircraft

The motors are CF2812s with 30A ESCs. These were used because they are small and quite powerful, meaning I can make a smaller aircraft. The bigger it is, the harder it is to move around.

Chapter 8

Conclusion

In this project a control system for a quadrotor was successfully developed using H-infinity robust control techniques and a coprime factorised model. Analysis was performed of two potential inner loop controllers including robustness analysis with an uncertainty model and a nonlinear simulation of controller performance.

The extended task of developing software that could be used to develop a LPV coprime factorised model to which robust control algorithm can be applied was also completed. Additionally an example LPV plant and controller were developed and analysed.

In the hardware stream an unusual type of quadrotor aircraft was created for an exploration task of mixed land and air travel. It is designed to carry a payload of either cargo or swam robots. A circuit board for the control system was also designed and created. Software for the flight control system is mostly complete but limited by an incompatibility in one of the libraries for the IMU.

Research Questions (Repeated)

1. How applicable are robust control techniques to robotic applications. What are future directions of research at the interface of control systems and robotics.

2. How well suited are Linear Parameter Varying robust control techniques for control of quadrotor aircraft. Are the benefits of parameter varying control justified by the increased complexity.

3. How do control synthesis techniques for LPV models compare? With particular reference to static non-varying controllers, SQLF based, and PDLF based controllers.

8.1 LSDP Controller

Of the LSDP controllers in Chapter 5, controller 2 underwent the most analysis and so the conclusion focuses on that controller. These were the first controllers developed but are also in many ways the most realistic controllers for implementation due to their

lower complexity compared to LPV. Controller 2 demonstrated good disturbance rejection performance, and this was verified by Simulink simulation. The Simulink simulation was found to be useful as a way of verifying the results obtained through linear control analysis techniques like singular value decomposition and step response plotting.

The performance in all aspects is likely to quite optimistic due to the lack of actuator models, and so a useful improvement would be their inclusion. These could be implemented as first or second order models but would require the Python software to be completed to enable more accurate system identification to be carried out. Further improvements that could be made include sensor noise models and gust disturbance models as they are the most significant external influences on the control system. Sensor noise can be estimates from datasheets and real world tests. The gust model is a more standard model in aerospace design. But, time is limited so not everything can be done. Reference tracking could also be implemented quite easily.

The use of generalised uncertainty with coprime factorisation gives good robustness properties without excessive complexity. To demonstrate the robustness properties of the controller a set of uncertainties more representative of those in the real system was constructed and the controller evaluated against them. In these tests it performed well, with robustness against an order of magnitude more than the 10% uncertainty modelled for inertia uncertainty. The model order reduction also proved to be a useful addition for maintaining consistent controller dimensions so that Simulink S-function did not have to be changed between controller redesigns. It also has some computational benefits but these were a side effect.

This does go some way to answering the first research question. The benefits of robust control techniques are significant in robotic applications where models may be well understood the operating environment may not be as well understood. Robust control techniques can provide a level of robustness against parameter uncertainty as well as these disturbance input type uncertainties. For R&D lab based efforts though simpler methods like LQG are probably preferable because a prototype can be made in much shorter timescales, and it will be easier to verify as safe. In terms of future directions for research in robust techniques in robotics this is likely to be in complex problems like the Mars Curiosity lander's Skycrane (see Fig: 8.1).

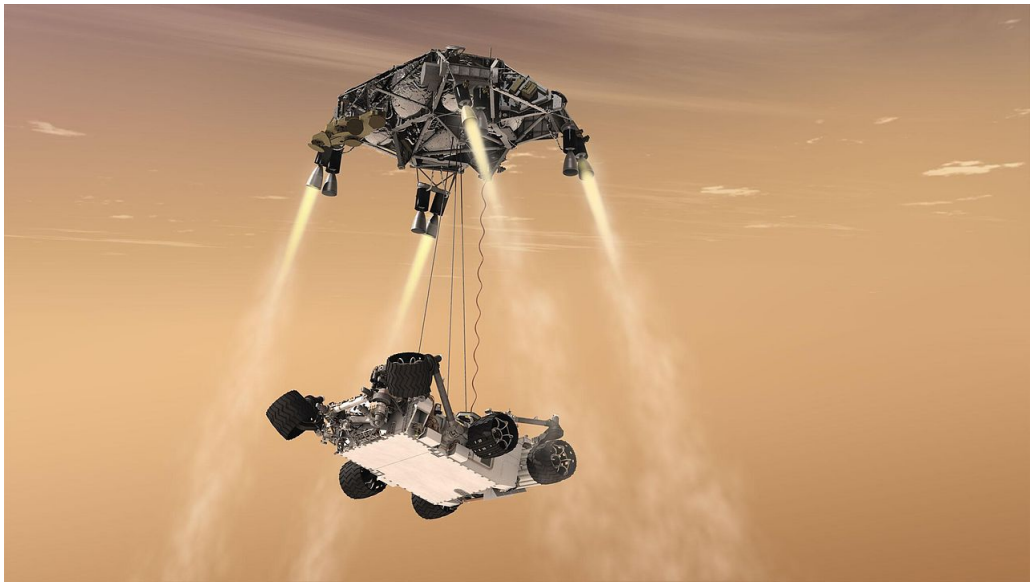


Figure 8.1: NASA Skycrane. Image courtesy of NASA/JPL [40]

8.2 LPV Controller

The objective set for this part of the project was very difficult, almost unrealistically difficult for a masters project. However, it was surprisingly successful. The code for the factorisation was the most difficult and time consuming task. Particular problems were figuring out how the coprime factorisation originated from the LMI solutions, how to incorporate the weighting matrices, and how to make the LPV models work with the coprime factorisation method. I had originally intended to apply the factorisation to the model of the form of Eq. 8.1.

$$G = A_0 + A_1 v_1 + A_2 v_2 \tag{8.1}$$

but this proved very difficult to do, and so I had to evaluate the model as discrete points with the gridding method. I had also expected to get a parameter for each factorisation from the LMIs, but I actually got one, which was shared with all of the LMIs.

The LPV robust control methods, both with and without pole placement constraints were able to deliver excellent results comparable to the standard H-infinity loopshaping controller. The pole placement contraints are an important addition as they make the controller more practical for implementation. Synthesis with both methods was fast so it can be concluded that the method is sufficiently efficient for most realistic control problems.

From the work done in this dissertation the use of LPV techniques does seem to be excessively complicated for most robotic applications. Strong robustness benefits can be

obtained using the LSDP techniques above and these would in most cases be better choice. With that said, LPV techniques have huge potential in the control of reconfigurable and self assembling robots. In such applications the changes in the robot are not well described by uncertainty as the model becomes very conservative. Adaptive control techniques are an option, but LPV gives the addition of robustness as well. Another potential option would be to gain schedule LSDP controllers.

The LPV can be implemented using the hinfgs function in Matlab if coprime factorised models are not required. Coprime factorisation was mostly used here because it is an opportunity to work with more experimental control techniques. If using the coprime factorised approach then an SQLF is easier to obtain, whereas hinfgs, which lack coprime factorisation, does support for PDLF.

8.3 Implementation

The implementation part of the project was the most problematic, particularly the software. At the very core of this was latency in the measurements which in turn make system identification and control much more difficult to achieve. The delays can be modelled using for example Pade delays, but modelling something doesn't automatically mean the controller will be able to compensate for it. The problem was first observed in the Java version which was due to update rates of the IMU. The Python version using RTIMULib2 was then used as an alternative but it has delays in the filter of approximately 1.5 seconds which for a quadrotor is unacceptable. The final attempt was to write a custom IMU driver for the IMU9255 with a custom written sensor data fusion algorithm, but there was insufficient time to complete this. Ideally the fusion algorithm and sensor measurements would be performed on a coprocessor as the instructions have to be executed in real time and Linux is not a real time operating system.

There are some positive outcomes to this. I am now able to write software in Python, as well as Java, C and Matlab. I am also more familiar with writing I2C drivers.

8.4 Project Management

The development principle applied here was to work for short to medium durations in different working environments, applying myself to varied tasks each day. This approach tends to deliver the best results when working in technical disciplines with complex problems to solve. Research forms an important part of this project and reading of this form is invariably most effective away from a desk.

The time estimates for the project were reasonably good, but there were some unexpected differences. The LPV controller was not expected to be completed, but surprisingly

worked much more quickly than was expected and was finished by the end of July. The project and dissertation were approximately 80% finished by the end of July. An additional task in the form of Atrium Tracker V2 was proposed in one project meeting, this was an unplanned task and required approximately one week of further development in August, and a further week of documentation shared with the flight control dissertation. I received a lot of helpful feedback from my supervisor during project meetings. Development was relatively straightforward. I was ill for approximately 1 1/2 weeks in Augusts firstly with food poisoning and secondly with a dental abscess but this did not cause any delays to my project.

8.5 Future Work

The LPV controller is a very complex control approach and not well suited to this type of aircraft in most cases. Realistic platforms, like airliners and fighter jets, are not readily available, so small RC type aircraft make for suitable demonstration platforms. The aircraft control problem could be solved with simpler control means and then be used for light logistics, robot navigation and mapping or for swam robotics. In these areas I feel it would genuinely help to bridge the gap between lab robotics and real world applications.

The concept of combining advanced control methods with robotics, the interface of the two also holds some promise for future development. Reconfigurable robots would be enhanced with control laws that are adaptive not just in the operating conditions sense (as per MRAC), but also adaptive to the current configuration of the robot.

Chapter 9

Atrium Tracker (v2)

In the project specification it was intended to write a paper to accompany this. However, writing such a paper would simply duplicate this dissertation in a more abbreviated form. Unfortunately this also means reading and writing the same thing twice. So as an alternative the paper is going to be the Atrium Tracker v2 paper which describes the work done on Atrium Tracker v2 in early August. Also note that the references for Atrium Tracker v2 are at the end of this chapter.

Atrium Tracker v2

Thomas Pile, 21048743

Sheffield Hallam University

This paper describes the development of the Atrium Tracker project, and the application of Neural Networks to frame differenced videos. The objective of the project is to process a set of videos of people walking through the Hertha Ayrton Atrium in the Engineering at Sheffield Hallam University to determine how many people in total passed through. This document describes the new version of the Atrium Tracker (v2) software and is the first version so far that can perform a count, albeit with errors. The approach taken has some similarities to previous versions but is essentially a complete rewrite.

The software can be divided into several main functions: Pre-processing the video, identifying a target object, tracking an object and post-processing. The software implementation is divided into two classes: Tracking and Detecting. The pre-processing uses frame differencing as a basis with some image enhancements. The object tracking stage uses a built-in Matlab cascade object detector [1] trained to detect human shapes. 9.1 gives an example of the video images being processed.

Figure 9.1: Hertha Ayrton STEM Centre

Detection

The detection stage uses a built-in detector called a "people detector" [2], which combines Histogram of Oriented Gradients (HOG) [3] features with an artificial Neural Network to detect human shapes in an upright pose. Alternative built-in detectors such as face and torso detectors based on the Viola-Jones algorithm [4] were tried in the previous version but the detector only performed well under good conditions with high resolution images and a clear front on angle. Viewed from five to ten meters the resolution of the face was typically insufficient to distinguish it from superficially similar background objects. Similarly faces captured at higher angles did not exhibit the necessary features to be recognised.

Custom cascade detectors [1] were attempted for the purpose, but some problems were found in training the detector. The most significant problem was that a large set of positive and negative training samples were required for the training process, and they had to well represent the range of potential operating conditions. Without this data a detector would have high false positive rate due to low order, or there would be insufficient samples to train a high order detector. To aid in the generation of samples some automation was used to vary lighting and false objects but this was still insufficient.

The "people detector" [2] was eventually made to work, although initial results were not good. The starting point was an image scaled by 50% from 1920×1080 to 960×540, converted to the Luminance/Chrominance YCbCr colour space, and then to greyscale. This was done to decrease memory and processing requirements, and to trade higher resolution for more image quality. A low false positive rate could then be achieved in

tests, but only by setting the ClassificationThreshold of the detector to 2 or more causing many valid detections to be missed and so demanding a lot from the tracking code. Figures 9.2 and 9.3 show the results of the detector with a threshold of 1.5, and Figures 9.4 and 9.5 with a detection threshold of 0.3. Comparison between these figures illustrates the problem.

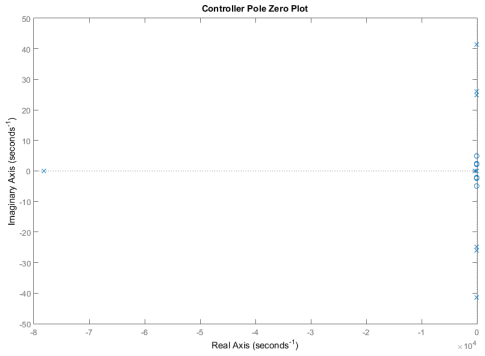


Figure 9.2: People, Threshold=1.5

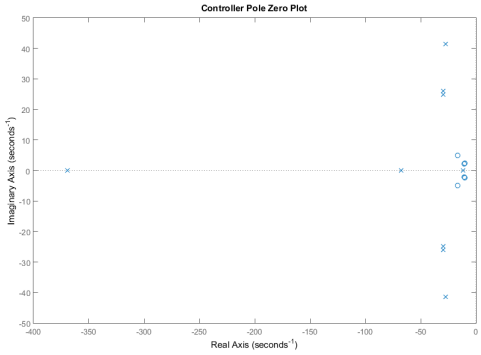


Figure 9.3: Locations, Threshold=1.5

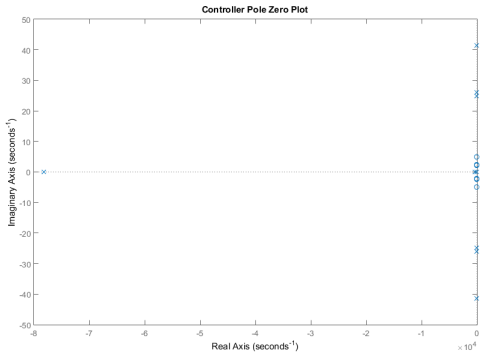


Figure 9.4: People, Threshold=0.3

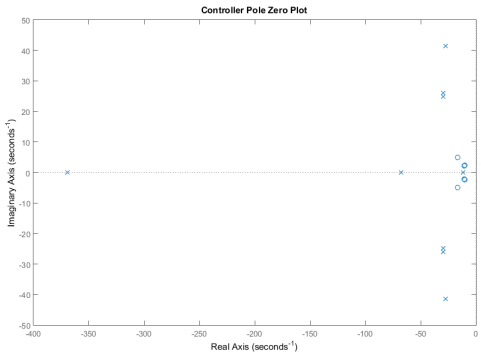


Figure 9.5: Threshold, Threshold=0.3

If frame differencing is applied to every second frame the detector is still able to detect people. This is an interesting and unexpected development because frame differencing tends to return only a general outline corrupted with blurring in the form of thin repeating lines. The greyscale image intensity values were stretched to enhance the contrast of the people using the imadjust function [5] with the input mapping vales set to between 0.01 and 0.2 (range upper bounded at 1), and mapped at the output to the full greyscale range. This is approximately the lowest 20% of the intensity value. This development meant that the detector threshold could be reduced to between 1.5 and 1.7, as the frame differencing eliminates the static objects responsible for the false positives. Figures 9.6 and 9.7 show the result of this method with a threshold of 1.7.

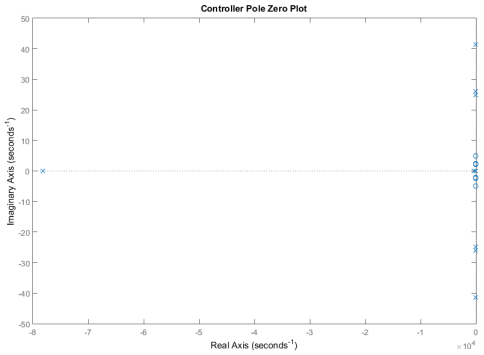


Figure 9.6: People, Threshold=1.7

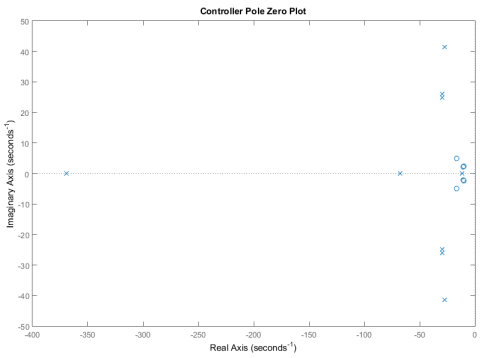


Figure 9.7: Locations, Threshold=1.7

The final stage was to increase the frame skip from 2 frames to 4 frames which resulted in additional noise which the neural network detected as a person in many causes. This was resolved by adding a light median filter using the default 3×3 window. An added benefit of the doubled skip quantity is a doubling of the processing rate of the videos. Combined this meant that the detector threshold could be reduced to 0.3. Figures 9.8 and 9.9 show the false detections caused by a threshold of 0.3 without the additional measures applied, and Figures 9.10 and 9.11 show the improved results.

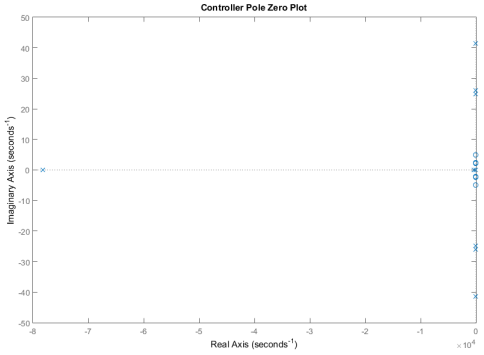


Figure 9.8: People, Without improvements applied

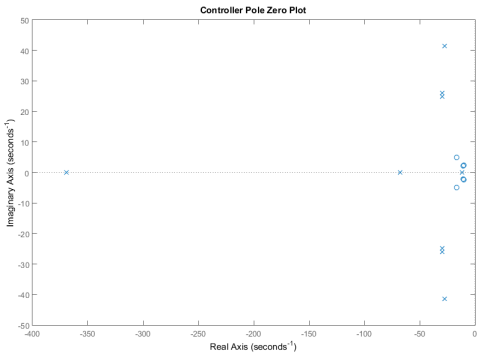


Figure 9.9: Locations, Without improvements applied

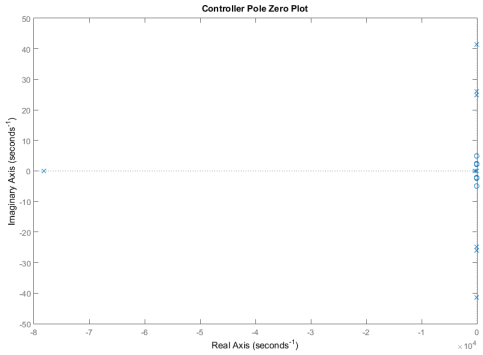


Figure 9.10: People, Improvements applied

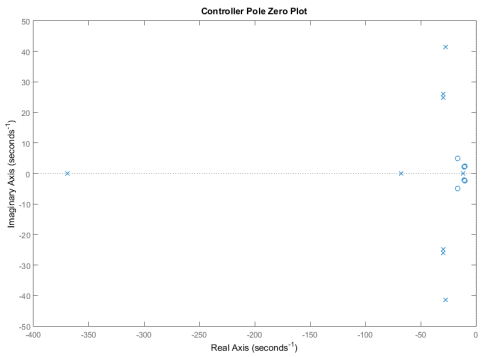


Figure 9.11: Locations, Improvements applied

Deblurring algorithms [6] including Gaussian, Sobel and Motion were each applied to the differenced image but improvements were difficult to identify, and a significant increase in CPU power was required. With many hours of video to process such dramatic increases in CPU power are an important factor and so these deblurring techniques cannot be recommended in this application.

Tracking

The tracking stage is based on a path forming approach where at each iteration location points are grouped with previous end points to form paths through the scene. Derived information including lengths of paths, scene transit times, and frequency of occurrence can be used to identify valid paths, and hence to count people.

The detector [2] produces a $n \times 4$ array, with n detected locations. This is checked against a list of known false positives and then passed to the tracking code. The tracking problem is to match each set of new points to previous sets of points to form pathways through the scene. The total number of people is then just the total number of paths. This problem is made more difficult with large gaps between points due to an overly conservative detector. Mismatching paths at crossover points is acceptable but breaks in the path are undesirable as this creates a false count if the path cannot be recovered.

A match-update cycle is used to process locations. When a set of locations is received the tracker iterates through all tracks, then projects the last known location forwards in time and finds the projected track location closest to each detected object. If no suitable match is found, then a new track is created. If a match is found, then the index of the matching track is used in the update stage. Only distance is used a criterion so far because a combination of a change in direction and a long period between updates causes too many broken paths. The distance criterion is a tuneable threshold in the tracker, with 150 being used in this case.

The update stage uses a Kalman filter [7] to estimate the magnitude and direction of

the velocity vector between the locations which is then used to better allocate locations to paths. As more locations are received the Kalman filter converges to a more accurate estimate for velocity and allows more accurate forward projection. An initial estimate is made for direction, and this then updated when a second location is received, which improves convergence rates. The accuracy of the Kalman filter improves over time by computing the covariance of the state, a value which is to be minimised.

The length of the path at each iteration is stored so that post-processing can remove unrealistic short paths. With further work it might even be possible to reallocate them to another path. An update counter is kept and when it reaches a threshold a track is defined as historic and no longer included in paths. This path-based element makes people sitting still or not moving very much simpler to filter out and prevents them from repeatedly triggering the counter.

The counting performance and false positive rejection rate of this method is good, and without excessive processing cost. The weakness of this method is that groups of people are difficult to track, although with some modifications improvements could be realised.

Identifying False Positives

False positives can be generated automatically for the blacklist by counting each point and marking the most common locations as bad. In some cases this may be undesirable, such as a person standing still, but such a person would be dropped by the tracker anyway as their path would time out.

References

[1] Mathworks (2018) “vision.CascadeObjectDetector: - Detect objects using the Viola-Jones algorithm”. Available online at: https://uk.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html?s_tid=doc_ta [Accessed 13 Aug. 2018].

[2] Mathworks (2018) ”vision.PeopleDetector: Detect upright people using HOG features”. Available online at: https://uk.mathworks.com/help/vision/ref/vision.peopledetector-system-object.html [Accessed 13 Aug. 2018].

[3] N. Dalal and B. Triggs ”Histograms of Oriented Gradients for Human Detection”, INRIA Rhone-Alps, 655 avenue de l’Europe, Montbonnot 38334, France. Available online at: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf [Accessed 13 Aug. 2018].

[4] P. Viola, M. J. Jones (2001) ” Rapid Object Detection using a Boosted Cascade of Simple Features”, Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. Volume: 1, pp.511–518.

[5] Mathworks (2018) ”Imadjust: Adjust image intensity values or colormap”. Available online at: https://uk.mathworks.com/help/images/ref/imadjust.html [Accessed 13 Aug. 2018].

[6] Mathworks (2018) "Deconvreg: Deblur image using regularized filter". Available online at: https://uk.mathworks.com/help/images/ref/deconvreg.html [Accessed 13 Aug. 2018].

[7] R. E. Kalman, "A new approach to linear filtering and prediction problems," J. Basic Eng., vol. 82, no.1, pp.35–45, Mar. 1960.

Chapter 10

Appendix

10.1 Appendix A - Analysis of Chen's Problem

.

To demonstrate the problem that Chen [10] had, I have produced a Matlab script [ml5_mixed_synth_integrator.m] which tries to synthesise a mixed synthesis controller for a pure integrator, using weights from the Matlab help file, and I obtained his rank conditions error. In checking page 31 of Chen's dissertation [10] it can be seen that the errors are indeed the same, and so my hypothesis is likely to be correct.

```
% An integrator system
s = tf('s');
G = 1/s ;
% weights from Matlab help
W1 = 10*(0.1*s+1)/(100*s+1);
W2 = 0.1*(0.1*s+1)/(0.01*s+1);
W3 = 0.01*(10*s+1)/(0.01*s+1);
% attempt to synth
[K,CL,GAM,INFO] = mixsyn(G,W1,W2,W3)
% fails because:
% [a b1 ;c2 d21] does not have full row rank at s=0
% Output argument "r12" (and maybe others) not assigned during call
% to "hinfst".
```

10.2 Appendix B - Matlab Synthesis Output for LPV Controller

Optimization of the $\mathcal{H}_\infty$ performance G :

```
-----------------------------------------------
Solver for linear objective minimization under LMI constraints
Iterations : Best objective value so far
1
2
3
4
5
6
7 388.518464
8 242.745973
9 161.564296
10 98.183848
11 98.183848
12 55.049708
13 55.049708
14 37.609387
15 37.609387
16 37.609387
17 16.764605
18 16.764605
19 8.615020
20 8.615020
21 6.537638
22 6.537638
23 5.112078
24 5.112078
25 4.336833
26 4.336833
27 3.962161
28 3.962161
*** new lower bound: 3.026110
29 3.775821
30 3.775821
31 3.724420
*** new lower bound: 3.594576
32 3.710078
33 3.710078
```

*** new lower bound: 3.608919

34 3.691252

*** new lower bound: 3.656185

Result: feasible solution of required accuracy

best objective value: 3.691252

guaranteed relative accuracy: 9.50e-03

f-radius saturation: 6.153% of R = 1.00e+08

Guaranteed Hinf performance: 3.68e+00

gopt = 3.6757

h2opt = []

10.3 Appendix C - Control Synthesis Code

10.3.1 Cont2: Attitude Controller with Coupling

```
% Simple quadrotor model linearised at one operating point, non-PV
% used for synthesising hinf LSDP controllers
% Based on each axis being independent, small angles, 2nd order DEs

% Thomas Pile, 21048743, SHU
% 21/6/2018

% Jxx = Jyy = 0.25*m*r^2 = 0.25*0.5*0.15^2 = 0.0028
% Jzz = 0.5*m*r^2 = 0.5*0.5*0.02^2 = 1.0000e-04 = 0.0001

% Constants
m = 0.5; kr = -0.02; Ixx = 0.0028; Iyy = 0.0028; Izz = 0.00110;

% Model
% ddrpy: model of the angular acceleration
% Could be modelled as 3 second order DE
a44 = kr*(1/Ixx);
a55 = kr*(1/Iyy);
a66 = kr*(1/Izz);
% off diagonal
a45 = -(Izz-Iyy)/Ixx;
a46 = -(Izz-Iyy)/Ixx;
a54 = -(Ixx-Izz)/Iyy;
a56 = -(Ixx-Izz)/Iyy;
a64 = -(Iyy-Ixx)/Izz;
a65 = -(Iyy-Ixx)/Izz;
% d2
```

```
28 ddrpy = [a44 a45 a46;
29                  a54 a55 a56;
30                  a64 a65 a66];
31
32 % drpy: model of the angular rate
33 % This is often modelled as an integrator, but actually there is a drag
34 % component to consider
35 % first order part of the 3 DE
36 a14 = 0.99; a15 = 0.99; a16 = 0.99;
37 drpy = [a14 0   0;
38                  0   a15 0;
39                  0   0   a16];
40 % combined
41 A = [zeros(3,3) drpy;
42          zeros(3,3) ddrpy];
43
44 % control inputs: roll distrib, pitch distrib, yaw distrib, total
       thrust
45 % scale for torque. 1/0.01Nm which is enough to rotate with unit input
46 % b1 = (1/Ixx)*0.0065; % so an input of 5 would be about right for 0.05
       nm
47 % b2 = (1/Iyy)*0.012; % b3 = (1/Izz)*0.003;
48 % scale so that maximum input (255) is 1
49 b1 = (1/Ixx) *(0.3060);  b2 = (1/Iyy) *(0.3060); b3 = (1/Izz) *(0.3060)
       ;
50 B = [0   0   0   0
51           0   0   0   0
52           0   0   0   0
53           b1  0   0   0
54           0   b2  0   0
55           0   0   b3  0];
56 % B = [0   0   0
57 %       0   0   0
58 %       0   0   0
59 %       b1  0   0
60 %       0   b2  0
61 %       0   0   b3];
62 %
63 % select rpy, xyz as outputs
64 %C = [eye(3)          zeros(3,3)]; % radians. This is good for a max
       output of 1 ~57 degrees
65 C = [57.2958*eye(3) zeros(3,3)];  % degrees. Good for short responses
66 D = zeros(3,4);
```

```
67 G = ss(A,B,C,D);
68
69 % provisional analysis of model
70 pzplot(G)
71 step(G,1)
72 %title('Pitch Coefficient Model Test')
73 %ylabel('angle [degrees]')
74 %xlabel('time [seconds]')
75
76 %%
77 %
78 % control synth
79 s = tf('s');
80 W1 = tf([1.3],[0.5 5.9]);
81 W2 = 0.4;
82 sigma(G,W1*G*W2)
83 legend('G','W1*G*W2')
84 title('Shaped Plant')
85
86 [K,CL,GAM,INFO] = ncfsyn(G,W1,W2)
87 sigma(K)
88 K = -K;
89
90
91 step(feedback(G,K),3)
92 step(CL((4:6),(4:6)),50)
93 % loop functions
94 loops = loopsens(G,K);
95 sigma(loops.So,loops.To,loops.CSo);
96 legend('S','T','KS');
97 title('Loop Shapes: S, T and KS')
98 % controller poles
99 pzplot(-K)
100 title('Controller Poles')
101
102 % TF from ref to error (I+KG)^-1 = Si
103 sigma(loops.Si)
104
105 % controller order reduction
106 % reduce controller order
107 [K2, redinfo2] = reduce(K,9);
108 [Ka Kb Kc Kd] = ssdata(K)
109
```

```
110 % export controller
111 %[Ka Kb Kc Kd] = ssdata(K);
112 save('Ka.mat','Ka');
113 save('Kb.mat','Kb');
114 save('Kc.mat','Kc');
115 save('Kd.mat','Kd');
116 size(Ka)
117 K2 = inv(W1)*K;
118
119 % reduced order controller
120 step(feedback(G,K),'b--',feedback(G,K2),'r--',3)
121 legend('Full Order','7th Order Controller')
122 title('Step Response: Reduced Order')
123
124 % loop functions
125 loops_reduced = loopsens(G,K2);
126 sigma(loops_reduced.So,loops_reduced.To,loops_reduced.CSo);
127 legend('S','T','KS');
128 title('Loop Shapes: S, T and KS - Reduced Order')
129
130 %%
131 % Uncertainty modelling
132 Ixx = ureal('Ixx',0.0028,'percentage',[-10 10]);
133 Iyy = ureal('Iyy',0.0028,'percentage',[-10 10]);
134 Izz = ureal('Izz',0.00110,'percentage',[-10 10]);
135
136 a44 = kr*(1/Ixx);
137 a55 = kr*(1/Iyy);
138 a66 = kr*(1/Izz);
139 a45 = -(Izz-Iyy)/Ixx;
140 a46 = -(Izz-Iyy)/Ixx;
141 a54 = -(Ixx-Izz)/Iyy;
142 a56 = -(Ixx-Izz)/Iyy;
143 a64 = -(Iyy-Ixx)/Izz;
144 a65 = -(Iyy-Ixx)/Izz;
145 ddrpy = [a44 0   0;
146                  0   a55 0;
147                  0   0   a66];
148 ddrpy = [a44 a45 a46;
149                  a54 a55 a56;
150                  a64 a65 a66];
151 a14 = 0.99;
152 a15 = 0.99;
```

```
153 a16 = 0.99;
154 drpy = [a14 0   0;
155                 0   a15 0;
156                 0   0   a16];
157 A = [zeros(3,3) drpy;
158         zeros(3,3) ddrpy];
159 b1 = (1/Ixx) *(0.3060);
160 b2 = (1/Iyy) *(0.3060);
161 b3 = (1/Izz) *(0.3060);
162 B = [0  0  0  0
163         0  0  0  0
164         0  0  0  0
165         b1 0  0  0
166         0  b2 0  0
167         0  0  b3 0];
168 C = [57.2958*eye(3) zeros(3,3)]; % degrees. Good for short responses
169 D = zeros(3,4);
170 % form uss model
171 uG = uss(A,B,C,D)
172 % verify nominal model
173 CLP = feedback(uG.NominalValue,K);
174 step(CLP,5)
175 % uncertain model
176 CLP = feedback(uG,K);
177 % uncertain time response
178 step(usample(CLP,10))
179 title('Uncertain Step Response')
180 % uncertain singular values
181 sigma(usample(CLP,10))
182 title('Uncertain Singular Value Plot of CL')
183
184 % mu analysis
185 % LowerBound: 7.1559
186 % UpperBound: 10.0000
187 % CriticalFrequency: Inf
188 [stabmarg,wcu] = robstab(CLP);
```

10.3.2 Coprime Factorised LPV Model with H-Infinity Controller Code

```
1 % Thomas Pile
2 % MSc Control and Robotics, Sheffield Hallam
3 % LMI code from Jianchi Chen, Uni of Leicester (~2010)
4 % LMI solution for coprime factorisation from Prempain, also UoL
5 % 23 May 2018
```

```
6  % This code uses LMIs and an algorithm by Prempain to produce a left
7  % coprime factorisation of a plant by solving an H2 optimisation
       problem.
8
9  %%
10 % 1. Make the GS model using LPV LMI model
11
12 % This is from my proof of concept controller 1
13 A = ... [0       0          0     0.9900          0          0;
14           0       0          0          0     0.9900          0;
15           0       0          0          0          0     0.9900;
16           0       0          0    -7.1429     0.6071     0.6071;
17           0       0          0    -0.6071    -7.1429    -0.6071;
18           0       0          0          0          0   -18.1818];
19 B = ...           [0          0          0;
20                    0          0          0;
21                    0          0          0;
22                  109.2857       0          0;
23                    0       109.2857      0;
24                    0          0     278.1818];
25 C = ...
26 [57.2958 0        0        0 0 0;
27 0         57.2958 0        0 0 0;
28 0         0        57.2958 0 0 0];
29 D = ...       [0      0      0;
30                  0      0      0;
31                  0      0      0];
32 Zmin=1; Zmax=4;
33 pv = pvec('box',[Zmin Zmax]);
34 % affine model:
35 s0 = ltisys(A,B,C,D);
36 s1 = ltisys([0 0 0 0 0 0;
37                          0 0 0 0 0 0;
38                          0 0 0 0 0 0;
39                          0 0 0 0.9 0 0;
40                          0 0 0 0 0.7 0;
41                          0 0 0 0 0 0.5],...
42                           zeros(6,3),zeros(3,6),zeros(3,3),0);
43 pdG = psys(pv,[s0 s1]);
44 tpG = aff2pol(pdG);
45 minfo(tpG)
46
47 %%
```

```
48 % 2. Extract the linear models at grid points
49 % sizes of the main matrices
50 szA = size(A);
51 szB = size(B); % 6 3
52 szC = size(C); % 3 6
53 % This needs to be updated to use the counting method later
54 numverts = 2;
55 A = zeros(szA(1),szA(2),numverts);
56 % extract B, C, D
57 pdsi = psinfo(tpG,'sys',1);
58 B = pdsi(1:szB(1), szA(2)+1:szA(2)+szB(2));
59 C = pdsi(szA(1)+1:szA(1)+szC(1), 1:szC(2));
60 D = pdsi(szA(1)+1:szA(1)+szC(1), szA(2)+1:szA(2)+szB(2));
61 % get the parameter space pvs = psinfo(tpG,'par');
62 % find its size, skip zero sizes, add 1 for the nominal model
63 for szpvs=1:size(pvs,1)
64         if pvs(szpvs,3)==0
65                 break;
66         end
67 end
68 % loop through the vertices (~parameters ish). n is the local vertex
       index
69 for n=1:szpvs-1
70         % loop through the grid
71         gridres = 10;
72         delta = (pvs(n,3)-pvs(n,2))/gridres;
73         for ipvs=1:gridres % i.e grid split into 10 parts, ipvs is the
               grid index for this vertex
74                 % increase grid point value
75                 gpd = pvs(n,2)+(delta*ipvs);
76                 % extract model at the point. polydec converts to
                       polytopic form
77                 gp = polydec(pv,gpd);
78                 pdm = psinfo(tpG,'eval',gp);
79                 % it psinfo return an ltisys, this extracts the A
                       matrix from it
80                 A(:,:,ipvs) = pdm(1:szA(1),1:szA(2));
81         end
82         % If multiple vertices were supported A(:,:,:) could be
               repackaged here
83         % into the form A(col,row,grid,vertex)
84         % A2 = zeros(szA(1),szA(2),szpvs-1,gridres); % move this!
85         % A2(:,:,n,:) = A(:,:,:);
```

```
86 end
87
88 %%
89 % 3. Apply weights
90 % Design weights s = tf('s');
91 W1 = tf([1.3],[0.5  5.9]);
92 W2 = 0.4;
93
94 % apply using series and ss
95 A1 = A; % A will need to be a different size anyway
96 clear('A')
97 for i=1:size(A1,3)
98         Gs = ss(A1(:,:,i),B,C,D);
99         Gs = series(series(Gs,W2),W1);
100        % put back in the pv 'A' matrix
101        [Ai,Bi,Ci,Di] = ssdata(Gs);
102        % recreate
103        if exist('A','var')==0
104                A = zeros(size(Ai,1), size(Ai,2), size(A1,3));
105        end
106        A(:,:,i) = Ai;
107 end
108 % B,C,D also different, though not fundamentally, just aug'd
109 B = Bi;
110 C = Ci;
111 D = Di;
112
113 %%
114 % 4. Solve coprime factorisation for P, Z
115 % R
116 [rowd,cold]=size(D);
117 Rt=eye(rowd)+D*D';
118 R=eye(cold)+D'*D;
119
120 % initialise the LMIs
121 setlmis([]);
122 [rowp,colp]=size(B);
123 P=lmivar(1,[rowp 1]);
124 Z=lmivar(1,[rowp 1]);
125
126 % Loop through all varying plants in A
127 % one LMI for each Ai matrix ie an A model exists for each vertex
128 % B, C and D have to be fixed, so no looping takes place
```

```
129 for i=1:size(A,3)
130         % first lmi term
131         jj=newlmi;
132         lmiterm([jj,1,1,P], 1,A(:,:,i),'s');
133         lmiterm([jj,1,1,0], -C'*C);
134         lmiterm([jj,2,1,P],B',1);
135         lmiterm([jj,2,1,0],-D'*C);
136         lmiterm([jj,2,2,0], -R);
137 end
138 % second lmi term
139 jj=newlmi;
140 lmiterm([-jj 1 1 Z],1,1);
141 lmiterm([-jj 2 1 0],1);
142 lmiterm([-jj 2 2 P],1,1);
143 lmisys=getlmis;
144 c = mat2dec(lmisys,zeros(rowp), eye(rowp));
145 options = [1e-2 200 1e5 10 0]
146 [copt,xopt] = mincx(lmisys,c,options)
147
148 %%
149 % 5. Form the coprime factorisation for each grid point using the
      solution
150 % P and Z
151 Popt=dec2mat(lmisys,xopt,1);
152 Zopt=dec2mat(lmisys,xopt,2);
153 % this is from Prempain LPV control P2
154 % Note that the coprime factorisation solution Popt enters here!
155 LL=-(B*D'+inv(Popt)*C')*inv(Rt);
156 [rowc,colc]=size(C(:,:,1));
157
158 %---contruct the coprime factorisation design structure
159 % This is where the coprime plant is produced, so it is analagous to G
      in a
160 % ss config. The Prempain formulation is similar but different.
161 for i=1:size(A,3)
162         Acop=A(:,:,i);
163         Bcop=[-LL*sqrt(Rt) B(:,:)];
164         c1=[zeros(size(C));C(:,:)];
165         c2=C(:,:);      Ccop=[c1;c2];
166         d11 = [zeros(size(D)); sqrt(Rt)];
167         d12 = [eye(size(D)); sqrt(Rt)];
168         d21 = sqrt(Rt);
169         d22 = D(:,:);
```

```
170          Dcop=[d11 d12; d21 d22];
171          ss2 = pck(Acop,Bcop,Ccop,Dcop);
172          Gcop(:,:,i)=pck(Acop,Bcop,Ccop,Dcop); % CPF LPV Model
173          Gcop2(:,:,i) = ss(Acop,Bcop,Ccop,Dcop);
174          % try to use modern notation
175 end
176
177 %%
178 % 6. Synthesise a H-infinity controller using standard tools
179 % try to run hinf control on this
180 r = [3 3];
181 for i=1:size(A,3)
182          P = Gcop(:,:,i);
183          [gopt1,K1] = hinflmi(P,r); % testing only
184          %[gopt,K,X1,X2,Y1,Y2] = hinflmi(P,r) % full
185
186          [Katmp, Kbtmp, Kctmp, Kdtmp] = unpck(K1);
187          % controller needs to be augmented with the weights in reverse
                 order
188          %Ks = series(series(K1,W2),W2);      K2 = ss(Katmp,Kbtmp,Kctmp,
                 Kdtmp);
189          K2 = series(series(K2,W1),W2);
190          [Katmp,Kbtmp,Kctmp,Kdtmp] = ssdata(K2);
191          % array of ccontrollers
192          K(:,:,1) = K2;
193          gopt(i) = gopt1;
194          % array of controller components, easier to extract!
195          Ka(:,:,i) = Katmp;
196          Kb(:,:,i) = Kbtmp;
197          Kc(:,:,i) = Kctmp;
198          Kd(:,:,i) = Kdtmp;
199 end
200
201 % compare gamma results
202 plot(1:size(A,3),gopt(:))
203 ylabel('Gamma Optimality')
204 xlabel('Param Index')
205
206 % nominal step response
207 G2 = ss(A(:,:,1),B,C,D);
208 K2 = ss(Ka(:,:,1), Kb(:,:,1), Kc(:,:,1), Kd(:,:,1))
209 CL = feedback(G2,-K2);
210 step(CL)
```

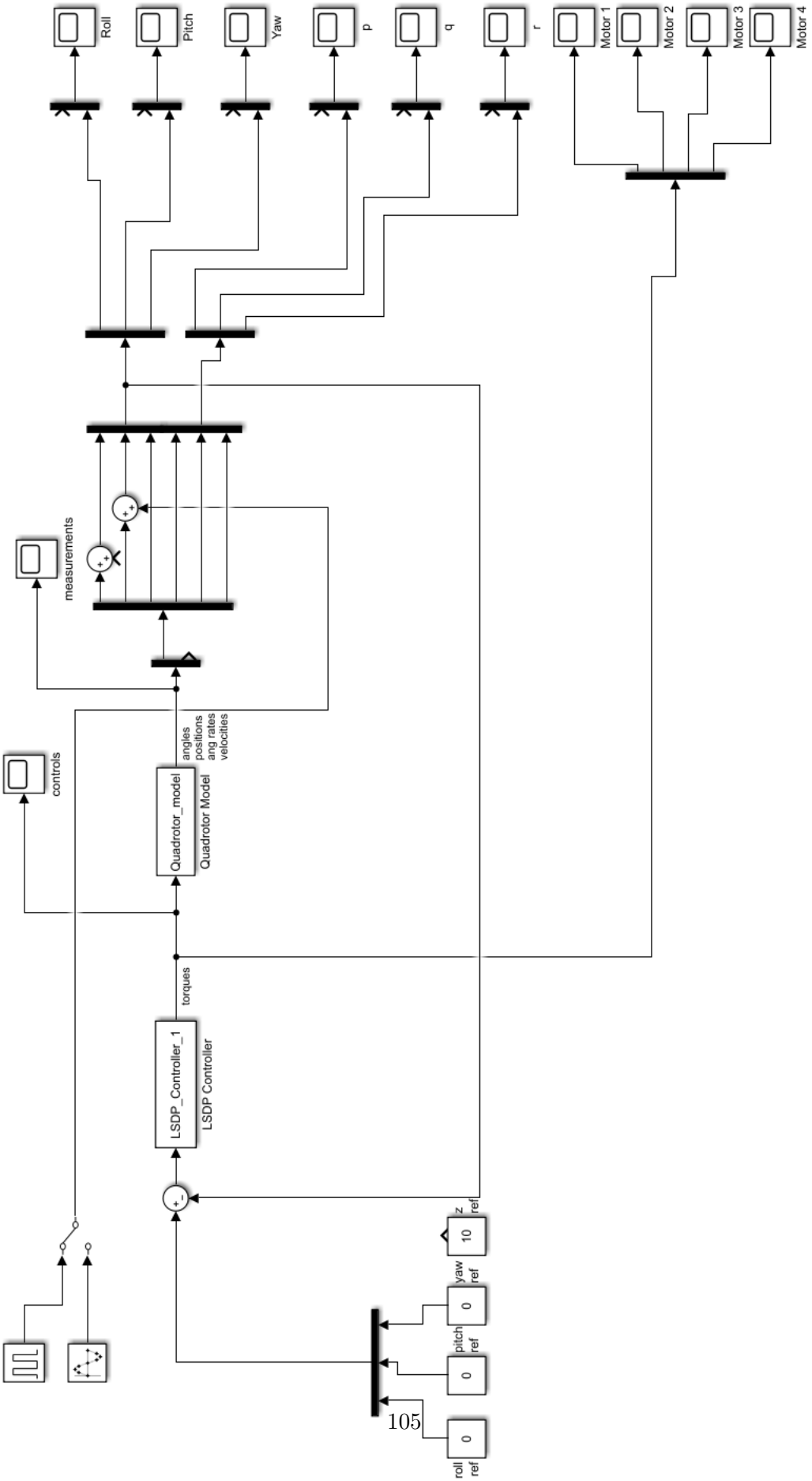
```
211
212 % loop functions
213 loops = loopsens(G2,-K2);
214 sigma(loops.So,loops.To,loops.CSo,{1e-2 1e3});
215 legend('S','T','KS');
216 title('Loop Shapes: S, T and KS')
217 legend('S','T','KS')
218
219 % controller gain
220 sigma(K2,{1e-2 1e3})
221 title('Controller Gain')
222
223 % pole zero plot - all frequencies
224 pzp = pzplot(K2);
225
226 % pole zero plot - lower frequencies
227 pzp = pzplot(K2);
228 pzpopt = getoptions(pzp);
229 pzpopt.XLim = {[-400 0]};
230 setoptions(pzp,pzpopt);
231 title('Controller Pole Zero Plot')
232
233 % multiple responses
234 for i=1:size(A,3)
235         G2 = ss(A(:,:,i),B,C,D);
236         K2 = ss(Ka(:,:,i), Kb(:,:,i), Kc(:,:,i), Kd(:,:,i))
237         CL = feedback(G2,-K2);
238
239         % for step response
240         step(CL)
241         % loop functions
242         % loops = loopsens(G2,-K2);
243         % sigma(loops.So,loops.To,loops.CSo);
244         % legend('S','T','KS');
245         % title('Loop Shapes: S, T and KS')
246         % for controller gain
247         % sigma(K2)
248         % title('Controller Gain')
249         hold on
250 end
251 hold off
252 legend('1','2','3','4','5','6','7','8','9','10')
253
```

```
254 %%
255 % 7. Mixed Synthesis
256 % There are likely to be HF poles placed by the algorithm so to counter
257 % this use pole placement constraints to limit HF poles. Can also place
        a
258 % bound on H2 aka LQR
259 %[gopt,h2opt,K,R,S] = hinfmix(P,r,obj,region,dkbnd,tol)
260 r = [0 3 3]; % dim h2, y, u
261 obj = [0 0 1 0];
262 % h8 up bnd, h2 up bdn, weighting of h8, weighting of h2
263 %region1 = lmireg;
264 % disc, centre at 0, radius 200
265 region1 = 1.0e+02 * ...
266 [-2.0000 + 0.0200i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 +
        0.0000i
267 0.0000 + 0.0000i  -2.0000 + 0.0000i   0.0100 + 0.0000i   0.0000 +
        0.0000i];
268 [gopt,h2opt,K,R,S] = hinfmix(P,r,obj,region1)
269
270 % nominal step response
271 G2 = ss(A(:,:,1),B,C,D);
272 [Katmp, Kbtmp, Kctmp, Kdtmp] = unpck(K);
273 K2 = ss(Katmp,Kbtmp,Kctmp,Kdtmp);
274 K2 = series(series(K2,W1),W2);
275 CL = feedback(G2,-K2);
276 step(CL)
277
278 % loop functions
279 loops = loopsens(G2,-K2);
280 sigma(loops.So,loops.To,loops.CSo,{1e-2 1e3});
281 legend('S','T','KS');
282 title('Loop Shapes: S, T and KS')
283 legend('S','T','KS')
284
285 % controller gain
286 sigma(K2,{1e-2 1e3})
287 title('Controller Gain')
288
289 % pole zero plot - all frequencies
290 pzplot(K2);
291
292 % synth for all parameters
293 for i=1:size(A,3)
```

```
294         P = Gcop(:,:,i);
295         %region2 = lmireg;
296         region1 = 1.0e+02 * ...
297         [-2.0000 + 0.0200i   0.0000 + 0.0000i   0.0000 + 0.0000i
                0.0000 + 0.0000i
298         0.0000 + 0.0000i  -2.0000 + 0.0000i   0.0100 + 0.0000i   0.0000
                + 0.0000i];
299
300         [gopt1,h2opt,K,R,S] = hinfmix(P,r,obj,region1);
301     [Katmp, Kbtmp, Kctmp, Kdtmp] = unpck(K);
302         % controller needs to be augmented with the weights in reverse
                order
303         %Ks = series(series(K1,W2),W2);      K2 = ss(Katmp,Kbtmp,Kctmp,
                Kdtmp);
304         K2 = series(series(K2,W1),W2);
305         [Katmp,Kbtmp,Kctmp,Kdtmp] = ssdata(K2);
306         % array of ccontrollers
307         gopt(i) = gopt1;
308         % array of controller components, easier to extract!
309         Ka(:,:,i) = Katmp;
310         Kb(:,:,i) = Kbtmp;
311         Kc(:,:,i) = Kctmp;
312         Kd(:,:,i) = Kdtmp;
313 end
314
315 % compare gamma results
316 plot(1:size(A,3),gopt(:))
317 ylabel('Gamma Optimality')
318 xlabel('Param Index')
319
320 % multiple responses
321 for i=1:size(A,3)
322         G2 = ss(A(:,:,i),B,C,D);
323         K2 = ss(Ka(:,:,i), Kb(:,:,i), Kc(:,:,i), Kd(:,:,i));
324         CL = feedback(G2,-K2);
325         % for step response
326         %step(CL)
327         % loop functions
328         %loops = loopsens(G2,-K2);
329         %sigma(loops.So,loops.To,loops.CSo);
330         %legend('S','T','KS');
331         %title('Loop Shapes: S, T and KS')
332
```

```
333         % for controller gain
334         sigma(K2)
335         title('Controller Gain')
336
337         hold on
338 end
339 hold off
340 legend('1','2','3','4','5','6','7','8','9','10')
```

10.3.3 Simulink Model



10.3.4 Simulink: LSDP_Controller_1.m

```
function [sys,x0,str,ts] = LSDP_Controller_1(t,x,u,flag)
% Implements H-inf Loop Shaping Controller
% Usage:
% Make a controller in Hinf_Controller_0_1_4.m and save the matrices of K
% as Ka.mat, Kb.mat etc. This is automated in the script.
% Control matrices are loaded in the derivative compute, and output compute
% sections and the relevant values computed.
% All you need to do is set the size of the initial vector and number of
% states (the same), and the size of the input and output vector.
%% Thomas Pile, Sheffield Hallam, 2018
% Tested and is fairly reliable
switch flag % determine which operation to perform
        case 0 % initialistion
        str = [];
                ts = [0 0];
                szs = simsizes;
                szs.NumContStates = 9;
                szs.NumDiscStates = 0;
                szs.NumOutputs = 4;
                szs.NumInputs = 3;
                szs.DirFeedthrough = 1;
                szs.NumSampleTimes = 1;
                sys = simsizes(szs);
                x0 = [zeros(9,1)];
        case 1 % derivatives equation update (compute controller state)
                sys = deriv_quad_LSDP(t,x,u);
        case 2 % discrete equation update
        case 3 % output calculations ( controller output)
                % check if the control matrices have been loaded
                if exist('Kc','var') == 0
                        Kc = importdata('Kc.mat');
                end
                if exist('Kd','var') == 0
                        Kd = importdata('Kd.mat');
                end
                sys = Kc* x + Kd* [u(1) u(2) u(3)]';
        case 4
        case 9
end
end% end csfunc
function dxdt = deriv_quad_LSDP(t,x,u)
        if exist('Ka','var') == 0
                Ka = importdata('Ka.mat');
        end
        if exist('Kb','var')==0
```

```
46                  Kb = importdata('Kb.mat');
47          end    % new version with imported controller
48          dxdt = Ka* x + Kb* [u(1) u(2) u(3)]';
49  end
```

10.3.5 Simulink: Quadrotor_Model.m

```
1   function [sys,x0,str,ts] = Quadrotor_model(t,x,u,flag)
2   % Kinematic and dynamic model of a quadrotor.
3   % Thomas Pile, Sheffield Hallam, June 2018
4   switch flag % determine which operation to perform
5           case 0 % initialistion
6                   str = [];
7                   ts = [0 0];
8                   szs = simsizes;
9                   szs.NumContStates = 12; % 12 continuous states
10                  szs.NumDiscStates = 0;
11                  szs.NumOutputs = 12; % roll,pitch,yaw,p,q,r,x,y,z,dx,dy,dz
12                  szs.NumInputs = 4; % control inputs
13                  szs.DirFeedthrough = 0;
14                  szs.NumSampleTimes = 1;
15                  sys = simsizes(szs);
16                  x0 = zeros(12,1); % initial conditions
17          case 1 % derivatives equation update
18                  sys = deriv_quad(t,x,u);
19          case 2 % discrete equation update
20          case 3 % output calculations
21                  sys = [x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8) x(9) x(10) x
                        (11) x(12)]; % r,p,y   p,q,r  x,y,z  vx,vy,vz
22          case 4 % get time of next variable hit
23          case 9 % termination end
24  end
25  function dx = deriv_quad(t,x,u)
26          r2d = 57.2957795;
27          dx = zeros(12,1);
28          kx = zeros(12,1); % controller initial state
29          g = 9.8;
30          m = 0.5;
31          Jxx = 0.0028; Jyy = 0.0028; Jzz = 0.00110;
32          % Lanzon robust quadrotor form
33          dx(1) = 0.99*x(4);      dx(2) = 0.99*x(5);      dx(3) = 0.99*x(6);
34          dx(4) = (-0.02*(1/Jxx)*x(4)) + (u(1)*(1/Jxx)); % p
35          dx(5) = (-0.02*(1/Jyy)*x(5)) + (u(2)*(1/Jyy)); % q
36          dx(6) = (-0.02*(1/Jzz)*x(6)) + (u(3)*(1/Jzz)); % r
37          dx(7) = 0;      dx(8) = 0;      dx(9) = 0;
38          dx(10) = 0;     dx(11) = 0;     dx(12) = 0;
39  end
```

10.4 Appendix D - Python Flight Controller Code

```
# V4 # Based on the included documentation for RTIMULib2 and PCA9685
# This code:
# 1. Reads IMU measurements and performs sensor fusion using RTIMULib2
# 2. Reads whether the power is on using a program implemented on the Atmega328p [Custom code]
# 3. Controls motors using PWM. Also sets up the motor limits and releases the brakes
# 4. PID is implemented but it's not possible to tune it due to the sensor latency (plant is faster than sensor)
# 5. Writes logs to desktop with flight data for sys ident
# Thomas Pile, Sheffield Hallam Uni, August 2018

from __future__ import division
import time from multiprocessing.dummy
import Pool as ThreadPool
# Import the PCA9685 module.
import Adafruit_PCA9685
import sys, getopt
sys.path.append('.')
import RTIMU
import os.path
import time
import math
import numpy as np # matrix mathematics package
import spidev # SPI

# Global Variables rpyref = np.array([0,0,0])
vxyz = np.array([0,0,0])
xyz = np.array([0,0,0])
headingT = 0
speedKM = 0
out1 = 0
out2 = 0
out3 = 0
out4 = 0
laser1 = 0
laser2 = 0
fcsenabled = 0
pwmupdatedelaycounter = 0
pwmupdatedelay = 100 # have to do this or the AVR goes wrong!

# ------------------------------------------------------------------------
# IMU Setup
# ------------------------------------------------------------------------
SETTINGS_FILE = "RTIMULib"
```

```
43 print("Using settings file " + SETTINGS_FILE + ".ini")
44 if not os.path.exists(SETTINGS_FILE + ".ini"):
45         print("Settings file does not exist, will be created")
46 s = RTIMU.Settings(SETTINGS_FILE)
47 imu = RTIMU.RTIMU(s)
48 print("IMU Name: " + imu.IMUName())
49 if (not imu.IMUInit()):
50         print("IMU Init Failed")
51         sys.exit(1)
52 else:
53         print("IMU Init Succeeded")
54 # this is a good time to set any fusion parameters
55 imu.setSlerpPower(0.01) # measured state influence. 0 to 1. 0.02 orig
56 imu.setGyroEnable(True)
57 imu.setAccelEnable(True)
58 imu.setCompassEnable(True)
59 poll_interval = imu.IMUGetPollInterval()
60 print("Recommended Poll Interval: %dmS\n" % poll_interval)
61
62 #------------------------------------------------------------------
63 # AVR Setup
64 #------------------------------------------------------------------
65 # Setup AVR servo interface
66 # 3.9mhz  = 3900000
67 # 1953mhz = 1953000
68 # 488khz  = 488000
69 # 244khz  = 244000
70 # 122khz  = 122000
71 bus = 0x00
72 device = 0x00 # 0x00 for the small board, 0x01 for the cnc board
73 spi = spidev.SpiDev()
74 spi.open(bus,device)
75 spi.max_speed_hz = 122000
76 spi.mode = 0b00 print("SPI Setup bus:0 dev:0")
77 # open log file
78 log = open('logfile.csv','w')
79 log.write('$, Roll, Pitch, Yaw, Velocity X, Velocity Y, Velocity Z, heading
       T, speedKM, rc_ch1_roll, rc_ch2_thrust, rc_ch3_pitch, fcs_enabled,
       ultrasonic_height, out1, out2, out3, out4, laser1, laser2, dt(nanoseconds
       )\r\n')
80
81 #------------------------------------------------------------------
82 # PCA9686 Servo Setup
83 #------------------------------------------------------------------
84 # Initialise the PCA9685 using the default address (0x40).
85 pwm = Adafruit_PCA9685.PCA9685()
86 # Configure min and max servo pulse lengths
```

```
87  servo_min = 150
88  # Min pulse length out of 4096
89  servo_max = 600
90  # Max pulse length out of 4096
91  # Helper function to make setting a servo pulse width simpler.
92  def set_servo_pulse(channel, pulse):
93          pulse_length = 1000000    # 1,000,000 us per second
94          pulse_length //= 60       # 60 Hz
95          print('{0}us per period'.format(pulse_length))
96          pulse_length //= 4096     # 12 bits of resolution
97          print('{0}us per bit'.format(pulse_length))
98          pulse *= 1000
99          pulse //= pulse_length
100         pwm.set_pwm(channel, 0, pulse)
101         # Set frequency to 60hz, good for servos.
102         pwm.set_pwm_freq(60)
103
104         # File reading
105         #file1 = open("file.txt")
106         #file1.read();
107         #file1.readline()
108         #for eachline in file1:
109         #step through
110         #  print(file1.line())
111
112         # servo values
113         servo1 = 1
114         servo2 = 1
115         servo3 = 1
116         servo4 = 1
117         servo5 = 1
118         motorsetupcomplete = 0;
119         rpyeprev = 0
120
121         while True:
122           if imu.IMURead():
123                 data = imu.getIMUData()
124                 fusionPose = data["fusionPose"]
125                 # transfer the rpy measurements to a better array and
                        convert to degrees
126                 rpy = np.array([math.degrees(fusionPose[0]),math.degrees(
                        fusionPose[1]),math.degrees(fusionPose[2])])
127                 rpye = np.subtract(rpyref,rpy)
128                 print("r_err: %f p_err: %f y_err: %f" % (rpye[0],rpye[1],
                        rpye[2]))
129
130                 # P controller
```

```
131                 dt = 0.04
132         kp = 10 # 0.3
133         ki = 0
134         kd = 30
135         # P
136         up = np.multiply(rpye,kp)
137         # D
138         ederiv = (rpye-rpyeprev)
139         ud = np.multiply(np.divide(ederiv,dt),kd)
140         rpyeprev = rpye
141         # I
142         ui = np.multiply(np.multiply(rpye,ki),dt)
143         u = np.add(up,ui,ud)
144
145         # control distribution
146         # roll
147         if u[0]>0:
148                 out1 = round(abs(u[0]))
149                 #out2 = -round(abs(u[0]))
150         if u[0]<0:
151                 #out1 = -round(abs(u[0]))
152                 out2 = round(abs(u[0]))
153         # check whether power is on
154         packet = [0x31, 0x00]
155         servopoweron = spi.xfer(packet)
156         #print(servopoweron[0])
157
158         # 0->1: motor not set up yet, power detected so doing a setup
159         if(motorsetupcomplete<1):
160                 # if power is on
161                 if(servopoweron[0]==1):
162                         # motor set up for the PCA
163                         pwm.set_pwm(0, 0, 200)
164                         pwm.set_pwm(1, 0, 200)
165                         pwm.set_pwm(2, 0, 200)
166                         pwm.set_pwm(3, 0, 200)
167                         time.sleep(2)
168                         pwm.set_pwm(0, 0, 600)
169                         pwm.set_pwm(1, 0, 600)
170                         pwm.set_pwm(2, 0, 600)
171                         pwm.set_pwm(3, 0, 600)
172                         time.sleep(2)
173                         pwm.set_pwm(0, 0, 200)
174                         pwm.set_pwm(1, 0, 200)
175                         pwm.set_pwm(2, 0, 200)
176                         pwm.set_pwm(3, 0, 200)
177                         time.sleep(2)
```

```
178                     motorsetupcomplete=1 # motor test mode
179                     motorsetupcomplete=1 # normal operation
180                     print("Motor setup Complete")
181             # end
182         # end
183         # 1: setup completed
184         if(motorsetupcomplete==1):
185             #
186             time.sleep(2)
187             pwm.set_pwm(0, 0, 250)
188             pwm.set_pwm(1, 0, 250)
189             pwm.set_pwm(2, 0, 250)
190             pwm.set_pwm(3, 0, 250)
191             time.sleep(2)
192             pwm.set_pwm(0, 0, 200)
193             pwm.set_pwm(1, 0, 200)
194             pwm.set_pwm(2, 0, 200)
195             pwm.set_pwm(3, 0, 200)
196         motorsetupcomplete=2
197         # end
198
199         # read rc input 1
200         # 0x11, 0x12, 0x13, 0x14
201         FROM8BitSERVO = 1
202         packet = [0x11, 0x00]
203         rcin1 = spi.xfer(packet)
204         # subtract base value. For ch1-4 these are 0, 70, 28, 50. top vals
                are 0, 250, 248, 228
205         rcin1 = rcin1[0] - 0
206         rcin1 = rcin1 * FROM8BitSERVO
207         #print(rcin1[0])
208
209         # read rc input 2 - roll
210         # -160 so zero is with stick at the centre
211         packet = [0x12, 0x00]
212         rcin2 = spi.xfer(packet)
213         #print(rcin2[0])
214         rcin2 = rcin2[0] - 160
215         rcin2 = rcin2 * FROM8BitSERVO
216
217         # read rc input 3 - thrust
218         # minus 26 so thrust starts from zero, up to 248 ish
219         packet = [0x13, 0x00]
220         rcin3 = spi.xfer(packet)
221         rcin3 = rcin3[0] - 26
222         rcin3 = rcin3 * FROM8BitSERVO
223         #print(rcin3[0])
```

```
224
225         # read rc input 4 - pitch
226         # centre: 154 to 158 ish
227         packet = [0x14, 0x00]
228         rcin4 = spi.xfer(packet)
229         #print(rcin4[0])
230         rcin4 = rcin4[0] - 154
231         rcin4 = rcin4 * FROM8BitSERVO
232         #print(rcin4[0])
233
234         # rc vales in a vector: thrust, roll, pitch, yaw. yaw isn't working
235         rcin = [rcin3, rcin2, rcin4, rcin1]
236
237         # autopilot enabled check function
238         packet = [0x30, 0x00]
239         spi.xfer(packet)
240         packet = [0x30, 0x00]
241         fcsenabledtmp = (spi.xfer(packet))
242         fcsenabled = fcsenabledtmp[0]
243         #print(fcsenabled)
244
245         # ultrasonic ranging
246         packet = [0x31, 0x00]
247         spi.xfer(packet)
248         packet = [0x31, 0x00]
249         sonicheight = spi.xfer(packet)
250         #print(sonicheight[0])
251
252         # actual output to servos
253         # outx should range 0 to max, then the base value is added to it
              here
254         if(motorsetupcomplete==2):
255                 #
256                 pwmbase = 250
257                 servo1 = int(pwmbase+out1)
258                 servo2 = int(pwmbase+out2)
259                 servo3 = int(pwmbase+out3)
260                 servo4 = int(pwmbase+out4)
261                 # lower limit
262                 if (servo1<210): servo1=210
263                 if (servo2<210): servo2=210
264                 if (servo3<210): servo3=210
265                 if (servo4<210): servo4=210
266                 # upper limit
267                 if (servo1>600): servo1=600
268                 if (servo2>600): servo2=600
269                 if (servo3>600): servo3=600
```

```
270                 if (servo4>600): servo4=600
271                 pwm.set_pwm(0, 0, servo1)
272                 pwm.set_pwm(1, 0, servo2)
273                 pwm.set_pwm(2, 0, servo3)
274                 pwm.set_pwm(3, 0, servo4)
275                 #print('motor 1 out: ', servo1)
276                 #print('motor 2 out: ', servo2)
277                 # if power goes low then go back to motor setup
278                 if(servopoweron[0]==0):
279                         motorsetupcomplete=2
280         # end
281         # Logging, also need this
282         log.write('0, ')
283         log.write('{}'.format(rpy[0])) # roll
284         log.write(', ')
285         log.write('{}'.format(rpy[1])) # pitch
286         log.write(', ')
287         log.write('{}'.format(rpy[2])) # yaw
288         log.write(', ')
289         log.write('{}'.format(vxyz[0])) # velocity X
290         log.write(', ')
291         log.write('{}'.format(vxyz[1])) # velocity Y
292         log.write(', ')
293         log.write('{}'.format(vxyz[2])) # velocity Z
294         log.write(', ')
295         log.write('{}'.format(headingT)) # heading T
296         log.write(', ')
297         log.write('{}'.format(speedKM)) # speed KM
298         log.write(', ')
299         log.write('{}'.format(rcin[1])) # rc_ch1_roll
300         log.write(', ')
301         log.write('{}'.format(rcin[0])) # rc_ch2_thrust
302         log.write(', ')
303         log.write('{}'.format(rcin[2])) # rc_ch3_pitch
304         log.write(', ')
305         log.write('{}'.format(fcsenabled)) # fcs enabled
306         log.write(', ')
307         log.write('{}'.format(sonicheight[0])) # ultrasonic height
308         log.write(', ')
309         log.write('{}'.format(servo1)) # out 1
310         log.write(', ')
311         log.write('{}'.format(servo2)) # out 2
312         log.write(', ')
313         log.write('{}'.format(servo3)) # out 3
314         log.write(', ')
315         log.write('{}'.format(servo4)) # out 4
316         log.write(', ')
```

```
317             log.write('{}'.format(laser1)) # laser 1
318             log.write(', ')
319             log.write('{}'.format(laser2)) # laser 2
320             log.write(', ')
321             log.write('{}'.format(dt)) # dt
322             log.write(' \n')        # sleep
323             dt = poll_interval*1.0/1000.0 # 4ms
324             #print('{}'.format(dt))
325             #time.sleep(dt)
326 #log.close()
```

10.5 Appendix E - Atrium Tracker v2 Code

10.5.1 atrium_tracker_COD_Tester_v3.m

```
1  % Thomas Pile, 21048743, Sheffield Hallam University % August 2018
2  % What it does:
3  % 1. Attempts to build a path of people walking through the Atrium so a
       count can be made
4  % 2. Places markers on screen for the location and estimate. Displays count
       on screen
5  % 3. Generates a video with the above info in it
6  r2d = 57.2958;
7  addyv1 = '227_04_10_17_0002.mp4'; addyv2 = '02_10_17_0004.mp4';
8  addyv3 = 'D:\Atrium_Tracker_Vids\CCTV_235\CCTV_235-20180802T211653Z-001\
       CCTV_235\10_04_18_0000.mp4';
9  v1 = VideoReader(addyv3);
10 skip = 0; % always zero
11 skipqty = 4;
12 % for differencing
13 picold = zeros(540, 960,'uint8');
14 % generate video output
15 vo = VideoWriter('newfile.avi','Motion JPEG AVI'); vo.Quality = 95; open(vo)
       ;
16 % setup the object tracker / counter
17 clear('Track');
18 % test objects
19 obj = [1 1        50 1];
20 % initial track value
21 Track(1).Xpos = -9999; Track(1).Ypos = -9999;
22 Track(1).Xvel = 0; Track(1).Yvel = 0;
23 Track(1).Velvect = 10; Track(1).Phi = 0;
24 Track(1).Dphi = 0; Track(1).Pvel = 0;
25 Track(1).Pphi = 0; Track(1).Historic = 0;
26 Track(1).StaticCount = 0; Track(1).TimeSinceUpdate = 0;
27 Track(1).JourneyLength = 0;
28 % Add known false locations here
```

```
29 Blacklist = [721, 332];
30 % experimental, only for frame differencing
31 detector = vision.PeopleDetector('UprightPeople_96x48');
32 detector.MinSize = [128 64];
33 % 50% frame
34 %detector.MaxSize = [250 125];
35 detector.MaxSize = [200 100];
36 detector.ClassificationThreshold = 0.3; %1.7;
37 figure;
38 ibbox = 1;
39 start = cputime; % loop through all frames
40 while hasFrame(v1)
41         skip = skip+1;
42         v1f = readFrame(v1);
43         % if enough frames have been skipped
44                 if skip==skipqty
45                         skip = 0;
46                         v1fcolour = v1f;
47                 % frame differencing, colour space change, and contrast
48                         % adjustment, and deblurring. bad performance
49                         pic = rgb2gray(rgb2ycbcr(imresize(v1fcolour,0.5)));
50                         pic2 = pic; % so we can save it as picold later
51                         pic = imadjust(pic-picold, [0.01 0.2], []); % frame
                               difference
52                         pic = medfilt2(pic);
53                         bbox = step(detector,pic);
54                 % check points against blacklist
55                 szBB2 = 0;
56                 bbox2 = zeros(size(bbox));
57                 for m=1:size(bbox,1)
58                         for k=1:size(Blacklist,1)
59                                 if sqrt( (Blacklist(k,1)-bbox(m,1))^2 + (
                                       Blacklist(k,2)-bbox(m,2))^2 ) < 5
60                                         % remove from bbox
61                                 else
62                                         % the value if good, keep
63                                         szBB2 = szBB2 +1;
64                                         bbox2(szBB2,:) = bbox(m,:);
65                                 end
66                         end
67                 end
68                 bbox2;          % put X,Y positions into array for tracker
69                 obj = zeros(size(bbox,1), 2);
70                 obj(:,:) = bbox(:,1:2);
71                 % update tracker
72                 [x, Track] = runTracker(obj, Track);
73                 % get Trackers estimates for plotting
```

```
74                  LAR = getTrackEstimates(Track);
75                  % how many people so far
76                  size(Track,2)
77          % label them better
78                  label_str = cell(size(bbox,1),1);
79                  %conf_val = [85.212 98.76 78.342];
80                  index = 1:1:size(bbox,1);
81                  % [x y width height].
82                  %position = [23 373 60 66;35 185 77 81;77 107 59 26];
83                  position = bbox(:,1:4);
84                  for ii=1:size(bbox,1)
85                          label_str{ii} = ['Count: ' num2str(index(ii),'%0.2f
                                ')];
86                  end
87                  if size(bbox,1)==0
88                          % as detected
89                          detectedImg = insertObjectAnnotation(pic2,'rectangle
                                ',bbox,'Person','Color','Green');
90                          % project forward
91                          % display the total count so far
92                          count_str = ['Total People: ' num2str(size(Track,2)
                                -1,'%d')];
93                          detectedImg = insertText(detectedImg,[10 10],
                                count_str,'FontSize',18,'BoxColor',...
94                          'green','BoxOpacity',0.4,'TextColor','white');
95                          % show final image
96                          %imshow(detectedImg);
97                  else
98                          % as detected alt
99                          detectedImg = insertObjectAnnotation(pic2,'rectangle
                                ',bbox,'Person','Color','Green');
100                         % display the total count so far
101                         count_str = ['Total People: ' num2str(size(Track,2)
                                -1,'%d')];
102                         detectedImg = insertText(detectedImg,[10 10],
                                count_str,'FontSize',18,'BoxColor',...
103                         'green','BoxOpacity',0.4,'TextColor','white');
104                 end
105                 writeVideo(vo,detectedImg)
106                 % collect all points, just for debugging
107                 if size(bbox,1)>0
108                         bboxsamples(ibbox).bbox = bbox;
109                         ibbox = ibbox + 1;
110                 end
111                 % end of processing that frame
112                 picold = pic2;
113     end
```

```
114 end
115 cputime-start
116 close(vo)
```

10.5.2 runTracker.m

```
1  function [ret, Track] = runTracker(obj,Track)
2  %runTracker: Object tracker and counter for the image processing toolbox.
3  %   Provided an array of object locations at discrete time steps
4  %the function will attempt to track the objects movement, discriminate
5  %between objects, and count the total objects to pass through the scene.
6  % Thomas Pile, Sheffield Hallam Uni, August 2018
7  % Key points
8  % + kalman filters to track direction and velocity
9  % + measurements are compared with estimated (projected forward) position %
       for each exising track
10 % + tracks are timed out of the frame
11 % + measure journey length to determine which objects are less valid
12 r2d = 57.2958;
13 % tuning values
14 % This determines how far off a point can be, and still be considered
15 % as associated with an existing target track. If its outside this
16 % range a new track is generated.
17 % Bigger: May misassign targets, Smaller: May lose tracks
18 target_assign_threshold = 300;
19 %300;
20 % best guess for velocity
21 velocityguess = 0;
22 % corners of the screen [(xl,yl),(xr,yr)]
23 box = [1,1,540,960];
24 % number of frames before a frame near the edge has "gone"
25 drop_track_timeout = 5;
26 % distance from the edge at which we apply this
27 prox_track_timeout = 5;
28 %persistent Track
29 ret = 1;
30 if isempty(Track)
31         X = 0; % state vector
32         %P = 99999; % covariance
33         % Objects are identified by row number
34         %         X,    Y      vel,   phi,   historic, static_count, dphi,
               Pvel, Phi
35         %         1     2      3      4      5         6             7     8
                  9
36         %Track = [-9999, -9999, -9999, -9999, 0,        0,            0,
               9999, 9999];
37         Track(1).Xpos = -1;          Track(1).Ypos = -1;
38         Track(1).Xvel = 0;           Track(1).Yvel = 0;
```

```
39          Track(1).Velvect = 0;          Track(1).Phi = 0;
40          Track(1).Dphi = 0;          Track(1).Pvel = 0;
41          Track(1).Pphi = 0;          Track(1).Historic = 0;
42          Track(1).StaticCount = 0;          Track(1).TimeSinceUpdate = 0;
43          Track(1).JourneyLength = 0;
44          % Note that 1 is basically an empty ex and should NEVER be matched
45  end
46  % increase the time since update flag for all targets
47  for k=1:size(Track,2)
48          Track(k).TimeSinceUpdate = Track(k).TimeSinceUpdate + 1;
49          % if update count exceeds a threshold, mark as historical
50          if Track(k).TimeSinceUpdate > 30 %60
51                  Track(k).Historic = 1;
52          end
53  end
54  % Find which target in the target vector is most likely to be it
55  for i=1:size(obj,1)% loop through all objects
56          % estimate which target is closest
57          % loop through target vector
58          bestmatch = [];
59          targeterrprev = 9999999999999;
60          for j=1:size(Track,2)
61                  % alternative method
62                  ang2 = Track(j).Phi;
63                  ang = atan2( Track(j).Ypos, Track(j).Xpos);
64                  projectedmag = (Track(j).TimeSinceUpdate*Track(j).Velvect);
65                  [projectedX projectedY] = pol2car(projectedmag,ang);
66                  projectedX = Track(j).Xpos + projectedX;
67                  projectedY = Track(j).Ypos + projectedY;
68
69                  % error between the measurement and the estimate position
70                  targeterr = sqrt( (obj(i,1)-projectedX)^2 + (obj(i,2)-
                        projectedY)^2 );
71                  % angle between last known point and here
72                  estangle = atan2((obj(i,2)-Track(j).Ypos),(obj(i,1)-Track(j)
                        .Xpos));
73                  targeterr;
74                  disp_est_angle = estangle*r2d;
75                  % find index of the smallest
76                  if targeterr<targeterrprev
77                          % check that its within reasonable count
78                          if(targeterr<target_assign_threshold)
79                                  % check that difference between measurement
                                        and
80                                  % past point Track(j) in terms of angle is
                                        less
81                                  % than some value
```

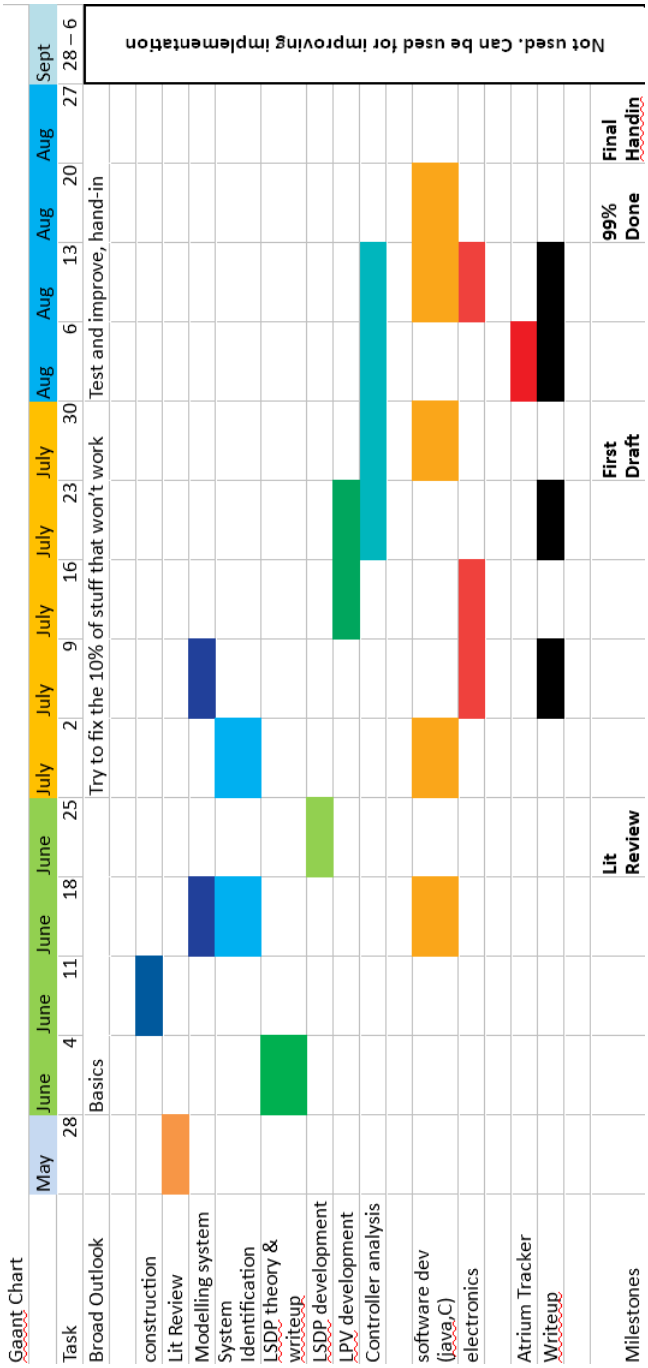
```
82                                  %if abs(estangle)*r2d < 90
83                                  % make sure it isn't a track that has left
                                        the scene
84                                  if Track(j).Historic==0
85                                          % check that the direction of travel
                                                is vaguely similar
86                                          % helps with crossing people
87                                          % skipped for now
88                                          bestmatch = j;
89                                  end
90                                  %end
91                          end
92                  end
93                  targeterrprev = targeterr;
94          end
95          [bestmatch estangle*r2d ang2*r2d]
96          % at the moment a result is always returned from above
97          if isempty(bestmatch)==1 || exist('bestmatch','var')==0
98                  % no target table match was found, add as new
99                  ni = size(Track,2)+1;
100                 Track(ni).Xpos = obj(i,1); Track(ni).Ypos = obj(i,2);
101                 Track(ni).Xvel = 0; Track(ni).Yvel = 0;
102                 Track(ni).Velvect = velocityguess; Track(ni).Phi = 0;
103                 Track(ni).Dphi = 0; Track(ni).Pvel = 0;
104                 Track(ni).Pphi = 0; Track(ni).Historic = 0;
105                 Track(ni).StaticCount=0; Track(ni).TimeSinceUpdate = 0;
106                 Track(ni).JourneyLength = 0;
107         else
108                 % make of probable target reference easier to handle
109                 bm = bestmatch;
110                 % hopefully improved best guess for angle for faster
                        convergence
111                 if Track(bm).JourneyLength==0
112                         Track(bm).Phi = atan2( (obj(i,2)-Track(bm).Ypos), (
                                obj(i,1)-Track(bm).Xpos) );
113                 end
114                 % Kalman filter for magnitude
115                 A = 1; % state transition matrix
116                 H = 1; % connect measurement to state
117                 R = 0.6; % measurement noise estimate
118                 Q = 1.0; % bigger this, quicker convergence
119                 % set vals for kf
120                 X = Track(bm).Velvect;
121                 P = Track(bm).Pvel;
122                 % distance travelled
123                 z = sqrt( (obj(i,1)-Track(bm).Xpos)^2 + (obj(i,2)-Track(bm).
                        Ypos)^2 );
```

```
124                 % update journey length
125                 Track(bm).JourneyLength = Track(bm).JourneyLength + z;
126                 % get velocity est
127                 if Track(bm).TimeSinceUpdate >0 % else inf
128                         z = z/Track(bm).TimeSinceUpdate;
129                 end
130                 % state update
131                 Xn = A*X; % project forward state
132                 Pn = A*P*A'+Q; % project forward covariance
133                 % correct estimate
134                 % calculate kalman gain
135                 K = (Pn*H')/(H*Pn*H'+R); % calculate kalman gain "/" for inv
136                 X = Xn + K*(z-H*Xn); % calculate state
137                 P = (1-K*H)*Pn; % calvulate covariance
138                 % velocity estimates
139                 Track(bm).Velvect = X;
140                 Track(bm).Pvel = P;
141                 % increase static count if its not moving
142                 % this is used to determine if a target has left. It does
                        not
143                 % determine if the target is stationary, we dont care
144                 if Track(bm).Velvect==0
145                         Track(bm).StaticCount = Track(bm).StaticCount + 1;
146                 end
147                 % Kalman filter for direction
148                 A = 1; % state transition matrix
149                 H = 1; % connect measurement to state
150                 R = 0.5; % measurement noise estimate
151                 Q = 1.1; %% bigger this, quicker convergence
152                 % set vals for kf
153                 X = Track(bm).Phi; % best guess is current angle
154                 P = Track(bm).Pphi;
155                 anglemeas = atan2( (obj(i,2)-Track(bm).Ypos), (obj(i,1)-
                        Track(bm).Xpos) );
156                 z = anglemeas; % measured from vector
157                 % state update
158                 Xn = A*X; % project forward state
159                 Pn = A*P*A'+Q; % project forward covariance
160                 % correct estimate
161                 % calculate kalman gain
162                 K = (Pn*H')/(H*Pn*H'+R); % calculate kalman gain "/" for inv
163                 X = Xn + K*(z-H*Xn); % calculate state
164                 P = (1-K*H)*Pn; % calvulate covariance
165                 % directiom estimates
166                 Track(bm).Phi = X;
167                 Track(bm).Pphi = P;
168                 %anglemeas;
```

```
169                 % Update state with info
170                 % location from latest measurement
171                 Track(bm).Xpos = obj(i,1);
172                 Track(bm).Ypos = obj(i,2);
173                 %Track(bm).Dphi = Xn-X; %dphi
174                 % zero its time since update counter
175                 Track(bm).TimeSinceUpdate = 0;
176             end
177 end
178 end
```

10.6 Appendix F - Project Documents

10.6.1 GANTT Chart



10.6.2 Risk Assessment

Sheffield Hallam University

Faculty of Arts, Computing, Engineering and Sciences

For Office Use Only
File Reference:
File Date:
Review Date:
Type of Activity:

Risk Assessment / Method Statement / Sequence of Work Form **RAMS3** (Nov 09)

Written By: (Print name) THOMAS PILE	**Sign:**	**Date:** 11/5/2018
Approved By: (Line Manager/Supervisor) Lyuba ALBOUL	**Sign:**	**Date:** 14/5/2018

* If this RA/MS is issued to several persons, each should sign overleaf - Copies of this RA/MS must be issued to all participating persons.

Activity / Project Title (Course): Robust control of a multirotor robot. MSc Automation, control & Robotics

Location: Sheffield Hallam

(Please use additional sheet if required)

Examples of Common Hazards (Please Tick all that apply) ☑

Machinery / tools ☐	Entanglement ☐	Dust ☐	Falling Objects ☑	Lighting ☐	Striking Object ☑
Chemical Hazard ☐	Cutting Accident ☑	Electrocution ☐	Fire ☐	Manual Handling ☐	Toxic Gases ☐
Climate ☐	Lasers ☐	Explosion ☐	Flooding ☐	Noise ☐	Traffic ☐
Material Ejection ☐	Display Screens ☐	Fall from Height ☐	Lifting Equipment ☐	Abrasion ☐	Slip/Trip Hazards ☐
N-I Radiation ☐	Vibration ☐	Other ☐	(please specify)		

Using the list above and your knowledge of the activity, select the SIGNIFICANT hazards
Give priority to hazards where the numbers of persons at risk, probability or severity are high.

	Significant Hazards Rate the hazards using the Risk Rating Analysis Matrix given in **Table 1** below. Ignore minor hazards covered by normal safe working practices.	Number of Persons at Risk	Probability	Severity	Risk Rating
①	Hit by low mass aircraft	1	2	3	6
②	Hit by propeller	1	2	3	6
③	Lithium Polymer battery fire	1	1	1	1
④	Eye strain working on computer				
⑤	RSI working on computer				

Substances / materials to be used:
Include substances produced as a by-product and identify Risk Phrases from MSDS

Risk Phrase Code:
Refer to **Table 3** / MSDS

Are less hazardous substances available? Yes / No

If 'Yes' reasons for not using them?

Sequence of Work / Method Statement (Please use additional sheet if required)

Mostly Nutlog based. The risks mostly exist in the testing phase but PPE mitigates most of them. I wear impact protected sunglasses most of the time anyway.

Include all controls that reduce the risks from the significant hazards

Control Measures using PPE
(Tick all that apply)

☑	Dust mask ☐	Ear protection ☐	Harness or line ☐	Overalls/Lab Coats ☐
		Eye protection ☑	Face Protection ☐	Respirator ☐
		Gloves ☐	High-vis clothing ☐	Safety Footwear ☐

Other PPE required

Control Measures:
Refer to **Table 2** for an interpretation of the Actions and Timescales required relative to the above individual Risk Rating/s.

1) Dont stand near the quadcopter

2) Dont stand near the quadcopter, and wear eye protection when standing not near the quadcopter.

3) Charge battery in a fireproof container, preferably in a basement.

4) Dont do too much work

5) Write a shorter dissertation

Safety Phrase Code:
Refer to **Table 4** / MSDS

Relevant Regulations

(Please Tick as Applicable) ✓

COSHH Control of Substances Hazardous to Health	☐	**EAWR** Electricity at Work regs	☐	**LEV** Local Exhaust Ventilation	☐	**LOLER** Lifting Operations & Lifting Equipment regs	☐
MH Manual Handling regulations	☐	**NAWR** Noise at Work regs	☐	**PPE** Personal Protective Equipment	☐	**Radiation** Ionising & Non-ionising Radiation	☐

Skills and Competencies

Do all the persons involved in this activity have the skills and competencies required to carry out the activity? - If YES please Tick ✓

If NO please indicate below the skills and competencies to be acquired as part of the activity.
(Please use additional sheet if required)

✓

Resources Required e.g. Plant, Equipment, Tools, etc. (Please use additional sheet if required)

Matlab, Tricotor / Quadrotor, Standard tools.

Emergency Procedures (Please use additional sheet if required)

Contact a member of staff immediately.
Green signs indicate how to contact a local first aider. For medical emergencies ring 888.

Call ambulance, use any ready materials to reduce bleeding.

Safety Phrase Code:
Refer to **Table 4** / MSDS

Waste Disposal Procedures (Please use additional sheet if required)

Additional Information (Please use additional sheet if required)

Date RA/MS To Be Reviewed (within 36 months of the date above)	**Date:**
Date Copy of RA/MS filed with Relevant Technical Team Leader / Advisor	**Date:**
Received / Approved By (Relevant Technical Team Leader / Advisor)	**Name:**

NOTE: Please attach any other information that you think may be relevant to this activity e.g. Handouts, Travel Arrangements, Accommodation, Site Contacts, Mobile & Landline Phone Numbers, etc.

Chapter 11

Glossary

API: Application Programming Interface. A software library of functions.

Autonomous system: A state space system with no control input. The energy in the system is only that from the initial conditions.

Bode: A plot of gain magnitude and phase over frequency. Typically the gain is plotted logarithmically and the phase is plotted linearly.

Coprime: Two values are coprime if the only positive number that divides both of them is 1, and they obey Bezout's identity.

Convex Polytope: A convex set of extreme points. A set is convex if a "straight line" can be draw between any two points in the set. This is easy to imagine in two dimensions but can be generalised to n-dimensional space.

H-Infinity: A norm but also used as short hand for a robust control technique based on the H-infinity ($\mathcal{H}_\infty$) norm

I2C: Communications standard. Supports multiple peripheral devices. Relatively slow at 400KHz.

Java: Interpreted programming language. Mostly used for enterprise applications.

Lie Derivative: The derivative of one vector field with respect to another.

LQG: Linear Quadratic Gaussian. A combination of a Kalman Filter and an LQR controller. The Kalman filter reconstructs the state and gives the controller some robustness against Gaussian white noise.

LQR: Linear Quadratic Regulator. A type of controller which uses optimal pole placement. A cost index is used to define the optimality constraints. Controller is found by solving AREs.

LTR: Loop Transfer Recovery is a technique to give some stability properties to an LQG controller by making it converge to a an LQR controller.

Lyuapunov Stability: Lyapunov stability is a measure of system stability using either Eigenvalues (1st method) or energy functions (2nd method).

Nominal Stability: Stability of an unperturbed model (no uncertainties).

MPU9250: IC package containing a gyroscope, accelerometer and magnetometer. Based on MEMS technology.

Mu analysis: Mu analysis is a technique for analysing the robustness of a system with real uncertainties (as opposed to complex uncertainties).

Mu synthesis: Mu synthesis a technique for synthesising a controller to minimise maximise its robustness against real uncertainties.

Python: Interpreted programming language. Widely used in science, maths and engineering.

Robust Performance: Performance is maintained in the presence of a modelled level of uncertainty.

Robust Stability: Stability is maintained in the presence of a modelled level of uncertainty.

SPI: Communications standard. Requires a select line for each peripheral device. Very fast, 1MHz+.

Tensor Polytopic: A tensor polytopic model is a less conservative subset of a full operating model. Full models of arbitrary varying parameters may have large operating regions for a given axis, but where combinations of extremes are unreachable. Tensor models eliminate these unreachable regions.

Bibliography

[1] H. Kwakernaak and R. Sivan (1972) "Linear Optimal Control systems". ISBN: 0-471-51110-2

[2] H. K. Khalil (2002) "Nonlinear Systems", third edition, Prentice Hall, Upper Saddle River, New Jersey.

[3] K. Zhou., J. Doyle., and K. Glover. (1996) "Robust and Optimal Control".

[4] K. Glover and D McFarlane (1989) "Robust Stabilization of Normalized Coprime Factor Plant Descriptions with &-Bounded Uncertainty", IEEE Trans on Automatic Control, Aug 1989.

[5] R. Van der Weijst., B. van Loon, M. Heertjes, and M. Heemels (2018) "Scheduled Controller Design for Systems With Varying Sensor Configurations: A Frequency-Domain Approach", IEEE Trans on Control Systems Technology, Vol 26, No 2, March 2018

[6] Y. V.Mitrishkin., A. A. Prohorov., P. S. Korenev., and M. I. Patrov (2017) "Robust H1 switching MIMO control for a plasma time-varying parameter model with a variable structure in a tokamak", Volume 50, Issue 1, July 2017

[7] D.W. Gu., P.H. Petkov. M.M Konstantinov. (2013) "Robust control design with Matlab". Second Edition. ISBN: 978-1-4471-4681-0

[8] F. Seve., S. Theodoulis., M. Zasadzinski., M. Boutayeb., and P. Wernert (2014) "Comparison of two H1 loop-shaping robust autopilot structure configurations for a 155mm spin-stabilized canard-guided projectile", 2014 22nd Mediterranean Conference on Control and Automation (MED)

[9] A. Hyde, K. Glover, and G. T. Shanks (1995) "VSTOL First Flight of an H-Infinity Control Law", Journal of Computing and Control Engineering. Accessible at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=372707

[10] Z. Chen (2017) "Model, Simulation and Control of a Helicopter", MSc Dissertation ACSE, University of Sheffeld, UK

[11] D. Hurley (2015) "Modelling, simulation and control of a model helicopter". ACSE, University of Sheffield.

[12] Lanzon, A. (2005) "Weight optimisation in loop-shaping". Automatica, 41(7), pp.1201- 1208.

[13] A. Lanzon (2000) "Weight Selection in Robust Control: An Optimisation Approach", PhD Thesis, Control Group, Department of Engineering, Wolfson College, University of Cambridge

[14] P. John. (2009) "The flapless air vehicle integrated industrial research (FLAVIIR) programme in aeronautical engineering". School of Engineering, Cranfield University. Proc. IMechE Vol. 224 Part G: J. Aerospace Engineering. DOI: 10.1243/09544100JAERO580

[15] J. Chen (2010) "Robust Linear Parameter Varying Control of an Unmanned Aerial Vehicle", PhD Thesis, Department of Engineering, University of Leicester.

[16] J. Chen., D. Gu., I. Postlethwaite., K. Natesan (2008) "Robust LPV Control of UAV with Parameter Dependent Performance", Proceedings of the 17th World Congress, IFAC, Seoul, Korea, July 6-11, 2008

[17] K. Natesan., D. Gu, and I. Postlethwaite (2009) "Design of linear parameter varying trajectory tracking controllers for an unmanned air vehicle", Department of Engineering, University of Leicester, Leicester. DOI: 10.1243/09544100JAERO586

[18] J. L. Lin., I. Postlethwaite, and D. W. Gu. (1993) "D-K iteration: a new algorithm for Mu-synthesis". Automatica, 29:219-244, 1993.

[19] A. Packard and J. C. Doyle. (1993) "The complex structured singular value". Automatica, Special Issue on Robust Control, 29:71-109, 1993.

[20] Z. Liu., C. Yuan., and Y. Zhang (2016) "Adaptive Fault-Tolerant Control of Unmanned Quadrotor Helicopter Using Linear Parameter Varying Control Technique", 2016 International Conference on Unmanned Aircraft Systems, p980-p985

[21] M. K. Mohamed. (2012) "Design and Control of UAV Systems: A Tri-Rotor UAV Case Study". University of Manchester

[22] K. Momamed (2017) "Advanced trajectory tracking for UAVs using combined feedforward-feedback control design", Robotics and Autonomous Systems 96 (2017) pp143-156.

[23] J. Hu and A. Lanzon (2018) “An innovative tri-rotor drone and associated distributed aerial drone swarm control”, Robotics and Autonomous Systems 103 (2018) pp162-174

[24] S. Chumalee. (2010) “Robust Gain-Scheduled H-infinity Control for Unmanned Aerial Vehicles”. PhD Thesis, Cranfield University.

[25] E. Prempain (2006) “Gain Scheduling Hinfinity Loop Shaping Control of Parameter-Varying Systems”, 5th IFAC Symposium on Robust Control Design, Toulouse, France. Department of Engineering, University of Leicester.

[26] G. Zames. (1966) “On the input-output stability of nonlinear time-varying feedback systems”, IEEE Transactions on Automatic Control. Vol AC-11 pp228 and pp465

[27] R.M. Redheffer. (1960) “On a certain linear fractional transformation”, J. Math. and Physics, vol. 29, pp 269-286

[28] O. H. Bosgra., H. Kwakernaak., G. Meinsma. (2008) “Design Methods for Control Systems”, Dutch Institute for Systems and Control. Delft

[29] M. Vidyasagar (1985) “Algebraic and topological aspects of feedback stabilisation”, IEEE Transactions on Automatic Control, Vol. AC-27, pp880-894

[30] S. Skogestad., and I. Postlethwaite (2003 maybe?) “Multivariable Control”

[31] Z. Ding. (2013) “Nonlinear and Adaptive Control Systems”. ISBN: 978-1-84919-574-4

[32] A. Lanzon., A. Freddi., and S. Longhi (2014) “Flight Control of a Quadrotor Vehicle Subsequent to a Rotor Failure”, Journal of Guidance, Control, and Dynamics, Vol. 37, No. 2 (2014), pp. 580-591.

[33] P. Gahinet., A. Nemirovski., A. J. Laub., and M. Chilali (1995) ”LMI Control Toolbox” Version 1. The Mathworks Inc, 3 Apple Drive, Natick, MA, USA.

[34] A. M. Lyapunov (1892) ”The General Problem of the Stability of Motion” (In Russian), Doctoral dissertation, Univ. Kharkov 1892

[35] J. Mawhin (2005) ”Alexandr Mikhailovich Liapunov, The general problem of the stability of motion (1892)”, Universit´e de Louvain, Institut math´ematique, B-1348 Louvain-la-Neuve, Belgium. Available online at: https://bit.ly/2P1b5Q9

[36] Raspberry Pi Foundation (2018) ”Raspberry Pi 3 Model B”. Available online at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[37] Raspberry Pi Foundation (2018) “The Pi4J Project”. Available online at: http://pi4j.com/

[38] R. Barnett (2015) “RTIMULib2”, Github.com. Available online at: https://github.com/richardstechnotes/RTIMULib2

[39] R. E. Kalman, “A new approach to linear filtering and prediction problems,” J. Basic Eng., vol. 82, no. 1, pp. 35–45, Mar.1960

[40] NASA JPL (2018) “ Mars Science Laboratory: Curiosity Rover”. Available online at: https://mars.nasa.gov/msl/

[41] G. Balas., R. Chiang., A. Packard., and M. Safonov (2012) ”Matlab Robust Control Toolbox: Getting Started Guide”. The Mathworks Inc, 3 Apple Drive, Natick, MA, USA.

[42] P. Seiler., U. Topcu., A. Packard, and G. Balas (2009) ”Parameter-Dependent Lyapunov Functions for Linear Systems With Constant Uncertainties”, IEEE Transactions On Automatic Control, Vol. 54, NO. 10, Oct 2009

[43] Glover, K. (1984) “All Optimal Hankel Norm Approximation of Linear Multivariable Systems, and Their Lµ-error Bounds,“ Int. J. Control, Vol. 39, No. 6, 1984, p. 1145-1193