Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2022 Åpen
Forfattere: Sander Hamborg, Daniel Årvik	
Fagansvarlig & Veileder: Damiano Rotondo Medveileder: Didrik Efjestad Fjereide	
Tittel på bacheloroppgaven: Fuzzy control of a 3DOF hover	
Studiepoeng: 20	
Emneord: Fuzzy Logic Control Theory Circuit Design	Sidetall: 59 + vedlegg/annet: 48 Stavanger 13. mai 2022

Acknowledgements

Both of us would like to express our thankfulness for the guidance around our bachelor's project. In particular, supervisor Damiano Rotondo with his experience and work around control engineering, and co-supervisor Didrik Efjestad Fjereide with his knowledge about the 3DOF Hover and the use of MATLAB. Both have been very helpful and responsive in the search for as good a thesis as possible. We would also like to send a special thanks to doctoral student Adrian Ruiz Royo for helping us in the modeling process of the 3DOF Hover.

Jeg vil først og fremst takke Daniel for samarbeidet rundt denne bacheloren. Jeg er stolt av den, og jeg kunne ikke oppnådd en så god besvarelse uten han. Videre vil jeg takke familie og venner for støtten under prosjektet.

- Sander Hamborg

En spesiell takk til Sander for den synergien vi til sammen har, oppgaven hadde ikke vært foruten. I tillegg vil jeg utrekke en takk til linjeforeningen for det gode miljøet de skaper på universitetet, og min familie og venner for alt de har lært meg gjennom årene.

- Daniel Årvik

Abstract

This thesis deals with the design of a fuzzy control system for the 3DOF hover system produced by Quanser. The human mind is great at making decisions based on estimated and non-precise information. We may use words such as "very steep" or "not so hot" when describing angles and temperatures, fuzzy logic is a way of implementing these words into controller design. The results and performance of the fuzzy control system are measured through comparisons with a linear quadratic regulator. As a second approach for controlling the 3DOF hover, weighted linearization for the non-linear system is implemented.

Sammendrag

Denne oppgaven omhandler design av et fuzzy kontrollsystem for 3DOF hover systemet produsert av Quanser. Menneskesinnet er flink til å ta avgjørelser basert på estimert og unøyaktig informasjon. Vi kan bruke ord som "veldig bratt" eller "ikke så varmt" når man skal beskrive vinkler og temperaturer, fuzzy logikk er en måte å iverksette disse begrepene i kontrollerdesign. Resultatene og ytelsen til det ikke-lineære kontrollsystemet er målt gjennom sammenligninger med en lineær kvadratisk regulator. Som en annen tilnærming for å kontrollere systemet, er vektet linearisering for det ikke-lineære systemet iverksatt.

Contents

List of Figures

List of Tables

Abbreviations

DOF	**D**egrees **O**f **F**reedom
LQR	**L**inear **Q**uadratic **R**egulator
TSK	**T**akagi **S**ugeno **K**ang
FIS	**F**uzzy **I**nference **S**ystem
FLC	**F**uzzy **L**ogic **C**ontroller
TSE	**T**aylor **S**eries **E**xpansion
WL	**W**eighted **L**inearization
mf	**m**embership **f**unction
wf	**w**eighting **f**unction

Part I

Introduction

Chapter 1

Aim and motive

1.1 Introduction

According to the project description that was published by the university. Fuzzy logic is the name given to a logic where the truth value of a statement/variable can take any real value between 0 and 1, for example 0.7, in contrast with the Boolean logic, where a statement/variable can only be either true or false, which means definite 0 or definite 1. And a fuzzy control system is a control system based on fuzzy logic where different controllers are blended according to the truth value of a rule. The fuzzy logic control system will be heavily discussed in this thesis.

1.2 History of Fuzzy Control

1965 is the year Lotfi A. Zadeh introduced the term fuzzy logic for the first time independently through the fuzzy set theory. The road to the fuzzy logic term, however, started way back. Some important years in the history and evolution of the fuzzy logic are mentioned here:

1920s, Jan Łukasiewicz and Alfred Tarski The Łukasiewicz logic was first introduced as a three value logic, before going on to be a n-value logic and the infinite value logic. This logic was the start and a completely necessary step towards developing today's fuzzy logic [5].

1974, Ebrahim H. Mamdani and Seto Assilian The first successful application of fuzzy logic to a control on a laboratory-scale [6]. A model steam engine was controlled by inferring the correction of the heat setting from the error and error

change of the speed and pressure. This is the origin of the inference systems that are known as Mamdani systems.

1982, Lauritz L. Holmblad and Jens J. Ostergaard The first industrial application of a fuzzy logic system, implemented on a cement kiln in Denmark [7].

1985, Tomohiro Takagi, Michio Sugeno A mathematical tool for building the fuzzy models of a system is presented by Takagi and Sugeno [8]. The fuzzy models uses either a linear or constant consequent.

1986, Michio Sugeno, Geuntaek Kang The problems surrounding structure identification of fuzzy models are addressed and an algorithm for identifying the structure is suggested [9]. The work builds upon the work Takagi and Sugeno did the previous year.

1987, Matsushita® Fuzzy logic was first available in 1987 when the company Matsushita sold the first consumer product with implemented fuzzy logic. The product was a shower head. Later in 1989 they also released the first washing machine with fuzzy logic [10].

1990, Kazuo Tanaka and Michio Sugeno Regarding the history of Takagi-Sugeno fuzzy controller, Kazuo Tanaka and Michio Sugeno began working on the idea through their seminal work introducing fuzzy model construction based on sector nonlinearity and parallel distributed compensation based on Lyapunov stability [11].

Remark: It is recognized that the fuzzy models in (3.3) to (3.6) are the result of work done by Takagi and Sugeno in 1985, thus being known as the Takagi-Sugeno fuzzy models [8]. Later on, in 1986, Kang and Sugeno did extensive work on structured identification of fuzzy models which is referred to as the Sugeno-Kang fuzzy modeling method [9]. In this report it's been chosen to acknowledge but not differentiate between the two previously stated facts, therefore addressing the system as a Takagi-Sugeno-Kang fuzzy inference system while a naming such as Takagi-Sugeno fuzzy inference system may be justified.

1.3 Goals

The main goals of this thesis can be listed as follows:

- Identify the nonlinear system model
- Compute a linearized state space model of the 3DOF Hover system
- Develop a Linear-Quadratic Regulator controller through the Linear Time-Invariant system with a cost function and optimized weighting matrices
- Implement a Takagi-Sugeno-Kang fuzzy inference system
- Implement weighted linearization of fuzzy logic control
- Compare the Takagi-Sugeno-Kang fuzzy logic controller with the linear controller
- Compare the study of the weighted linearization approach against the Takagi-Sugeno-Kang fuzzy controller
- Conclude on which controller approach suits the 3DOF Hover system the best.

To achieve these goals, results will be collected through simulations in MATLAB and Simulink. Physical study and results are captured through experiments to adjust the Quanser Aero 3DOF Hover which is available in room KE E-455 at the University of Stavanger.

Chapter 2

The 3DOF Hover system

2.1 Description

The producer's webpage describes the 3 Degrees Of Freedom (DOF) Hover seen in Figure 2.1 in the following way [3]:

> *The 3 DOF Hover consists of a planar round frame with four propellers. The frame is mounted on a three degrees of freedom pivot joint that enables the body to rotate about the roll, pitch and yaw axes. The propellers are driven by four DC motors that are mounted at the vertices of the frame. The propellers generate a lift force that can be used to directly control the pitch and roll angles. Two of the propellers are counter-rotating, so that the total torque in the system is balanced when the thrust of the four propellers is approximately equal. The axis angles are all measured using high-resolution encoders. The encoder and motor signals are transmitted through a slip ring mechanism, which allows the yaw axis to rotate continuously about 360 degrees.*

Fig 2.1: The Quanser 3DOF hover test bed.

Each DC motor is a Pittman Model 9234 [12] with an electrical resistance of 0.83Ω and a current-torque constant, $K_t = 0.0182Nm/A$. The supply voltage V_s of the motor is $12V$, but its peak voltage can be brought up to $24V$ without damage. The input voltages V_x, $x = [f, b, r, l]$, are limited by the drivers. Further on considering given values and limits from Quanser Aero's datasheet [3] to help develop the controller.

The minimum and maximum pitch and roll angles are read as $\pm 37.5^\circ \approx \pm 0.6545$ rad, while the yaw angle spins freely 360° around its axis.

The angular velocities p, q, r are found experimentally to be bounded by allowing the motor voltages to reach saturation, with minimum and maximum velocity being approximately ± 1 rad/s.

2.2 State-feedback

A state-feedback system includes the use of a given state vector to calculate the control action for selected system dynamics. In this case, the system is a non-linear MIMO-system, where the state-feedback consists of a gain matrix K determined by a Linear Quadratic Regulator. State-feedback is also used in simpler systems, for example in a SISO-system the gain K would act as a row vector.

Figure 2.2 depicts state-feedback regulation, as there is a connection between the process output y and the control input u. The generation or calculation of the control input u happens inside the regulator [13].

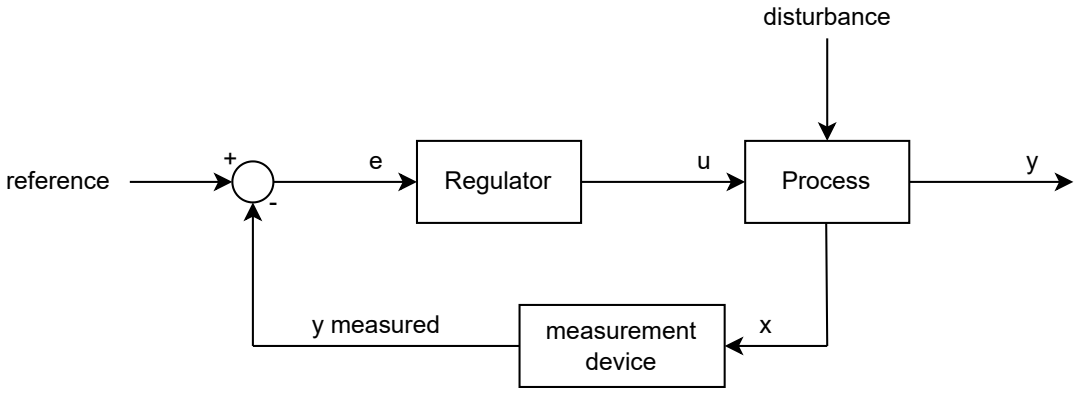


Fig 2.2: State-feedback regulation diagram.

2.3 State-space representation

The state-space representation is a structured way to define the differential equations describing a system. When working in the state-space domain there are methods for linearization, simulation and analysis that are very helpful. Additionally, state-space models are the foundation for topics like observability and controllability [14].

The Equations (2.1) and (2.2) represent the state-space model of a general n'th order system.

$$\dot{x}(t) = \begin{cases} \dot{x_1}(t) = f_1(x_1(t), \ldots, x_n(t), u_1(t), \ldots, u_n(t), t) \\ \vdots \\ \dot{x_n}(t) = f_n(x_1(t), \ldots, x_n(t), u_1(t), \ldots, u_n(t), t) \end{cases} \tag{2.1}$$

$$y(t) = \begin{cases} y_1(t) = g_1(x_1(t), \ldots, x_n(t), u_1(t), \ldots, u_n(t), t) \\ \vdots \\ y_p(t) = g_p(x_1(t), \ldots, x_n(t), u_1(t), \ldots, u_n(t), t) \end{cases} \tag{2.2}$$

Where $f(t)$ are the differential input equations and $g(t)$ are the algebraic output equations.

The state-space representation of a continuous time-invariant system can then be simplified using vector notation, resulting in the system (2.3).

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{2.3}$$

With the following elements:

- time t
- state vector $x(t)$
- output vector $y(t)$
- input/control vector $u(t)$
- state matrix A
- input matrix B
- output matrix C
- feedthrough matrix D

2.3.1 Dynamics and state-space modeling

The 3DOF hover is described by a system of nonlinear Equations (2.4) - (2.9).

$$\dot{\theta_r} = f_1 = p + sin\theta_r tan\theta_p q + cos\theta_r tan\theta_p r \tag{2.4}$$
$$\dot{\theta_p} = f_2 = cos\theta_r q - sin\theta_r r \tag{2.5}$$
$$\dot{\theta_y} = f_3 = \frac{sin\theta_r}{cos\theta_p} q + \frac{cos\theta_r}{cos\theta_p} r \tag{2.6}$$
$$\dot{p} = f_4 = \frac{J_p - J_y}{J_r} qr + \frac{I_r q}{J_r} u_g + \frac{1}{J_r} \tau_r \tag{2.7}$$
$$\dot{q} = f_5 = \frac{J_y - J_r}{J_p} pr - \frac{I_r p}{J_p} u_g + \frac{1}{J_p} \tau_p \tag{2.8}$$
$$\dot{r} = f_6 = \frac{J_r - J_p}{J_y} pq + \frac{1}{J_y} \tau_y \tag{2.9}$$

By isolating the equation describing the roll axis, the following dynamics can be identified,

$$f_4 = \underbrace{\frac{J_p - J_y}{J_r} qr}_{\substack{\text{Coriolis} \\ \text{Effect}}} + \underbrace{\frac{I_r q}{J_r} u_g}_{\substack{\text{Rotor} \\ \text{Gyroscopic} \\ \text{Effect}}} + \underbrace{\frac{1}{J_r} \tau_r}_{\substack{\text{Body} \\ \text{Gyroscopic} \\ \text{Torque}}}$$

where the term describing the rotor gyroscopic effect [15] contains u_g, a sum of voltages as presented in Equation (2.10).

$$u_g(u_1, u_2, u_3, u_4) = K_v(V_r + V_l - V_f - V_b) \tag{2.10}$$

And the torques τ_r, τ_p and τ_y are given by the following Equations (2.11) - (2.13) [16].

$$\tau_r(u_3, u_4) = LK_f(V_r - V_l) \tag{2.11}$$
$$\tau_p(u_1, u_2) = LK_f(V_f - V_b) \tag{2.12}$$
$$\tau_y(u_1, u_2, u_3, u_4) = K_t(V_r + V_l - V_f - V_b) \tag{2.13}$$

The parameters of the 3DOF hover are listed in Table 2.1.

Symbol	Description	Value	Unit
K_t	Torque thrust constant of motor/propeller	0.0036	N m/V
K_f	Force-thrust constant of motor/propeller	0.1188	N/V
L	Distance between pivot to each motor	0.197	m
J_r	Equivalent moment of inertia about roll axis	0.0552	kg m^2
J_p	Equivalent moment of inertia about pitch axis	0.0552	kg m^2
J_y	Equivalent moment of inertia about yaw axis	0.110	kg m^2
I_r	Motor rotor moment of inertia	$6{\cdot}10^{-5}$	kg m^2
K_v	Transformation constant	54.945	rad s/V

Table 2.1: Model parameters according to the producers at [3].

Additionally, the state $x(t)$, input/control $u(t)$ and output $y(t)$ are given by the vectors (2.14) - (2.16).

$$x(t) = [\theta_r, \theta_p, \theta_y, p, q, r]^T \tag{2.14}$$
$$u(t) = [V_f, V_b, V_r, V_l]^T \tag{2.15}$$
$$y(t) = [\theta_r, \theta_p, \theta_y]^T \tag{2.16}$$

Where V_f, V_b, V_r, V_l correspond to the voltage control variables for the front, back, right and left motor, and θ_r, θ_p and θ_y are the Euler angles yaw, pitch and roll as demonstrated in Figure 2.3.

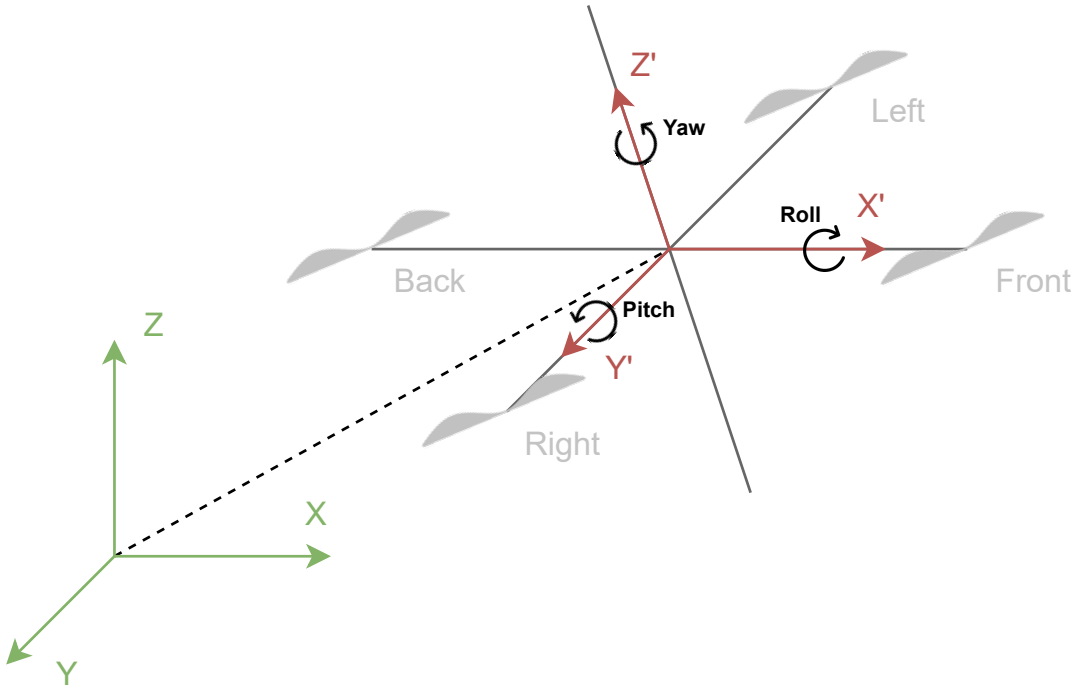


Fig 2.3: Coordinate system. X, Y and Z are the axes in the world frame. X', Y' and Z' are the axes in the body frame.

The attitude is arranged in the Euler angle sequence 1-2-3, often used in aerospace settings, referring to the angles as Tait-Bryan angles. The states p, q and r are the angular velocities in the body frame, which are described by Equation (2.17) [17].

$$[p,q,r]^T = E'_{123}\begin{bmatrix}\dot{\theta}_r\\ \dot{\theta}_p\\ \dot{\theta}_y\end{bmatrix} = \begin{bmatrix}1 & 0 & -sin\theta_p\\ 0 & cos\theta_r & cos\theta_p sin\theta_r\\ 0 & -sin\theta_r & cos\theta_p cos\theta_r\end{bmatrix}\begin{bmatrix}\dot{\theta}_r\\ \dot{\theta}_p\\ \dot{\theta}_y\end{bmatrix} \tag{2.17}$$

Where E'_{123} is the conjugate Euler angle rates matrix, which gives the angular velocities in the body frame of reference when multiplied with the Euler angle rates $\dot{\theta}_r$, $\dot{\theta}_p$ and $\dot{\theta}_y$.

The input Equations (2.4) - (2.6) are derived from Equation (2.18) [17].

$$[\dot{\theta}_r,\dot{\theta}_p,\dot{\theta}_y]^T = [E'_{123}]^{-1}\begin{bmatrix}p\\ q\\ r\end{bmatrix} = \begin{bmatrix}1 & sin\theta_r tan\theta_p & cos\theta_r tan\theta_p\\ 0 & cos\theta_r & -sin\theta_r\\ 0 & sin\theta_r/cos\theta_p & cos\theta_r/cos\theta_p\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{2.18}$$

Where $[E'_{123}]^{-1}$ is the inverse conjugate Euler angle rates matrix, which gives the Euler angle rates when multiplied with the angular velocities p, q and r in the body frame of reference.

2.3.2 Linearization

Based on the nonlinear Equations (2.4) - (2.9), and partially deriving these with respect to the state vector (2.14) we would obtain the state matrix $A(x,u)$ as Equation (2.19).

$$A(x,u) = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_r} & \frac{\partial f_1}{\partial \theta_p} & \cdots & \frac{\partial f_1}{\partial r} \\ \frac{\partial f_2}{\partial \theta_r} & \frac{\partial f_2}{\partial \theta_p} & \cdots & \frac{\partial f_2}{\partial r} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_6}{\partial \theta_r} & \frac{\partial f_6}{\partial \theta_p} & \cdots & \frac{\partial f_6}{\partial r} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & sin\theta_r tan\theta_p & cos\theta_r tan\theta_p \\ 0 & 0 & 0 & 0 & cos\theta_r & -sin\theta_r \\ 0 & 0 & 0 & 0 & \frac{sin\theta_r}{cos\theta_p} & \frac{cos\theta_r}{cos\theta_p} \\ 0 & 0 & 0 & 0 & \frac{J_p - J_y}{Jr} r - \frac{I_r u_g}{Jp} & \frac{J_p - J_y}{J_r} q \\ 0 & 0 & 0 & \frac{J_y - J_r}{J_p} r + \frac{I_r u_g}{Jr} & 0 & \frac{J_y - J_r}{J_p} p \\ 0 & 0 & 0 & \frac{J_r - J_p}{J_y} q & \frac{J_r - J_p}{J_y} p & 0 \end{bmatrix} \tag{2.19}$$

The terms $\frac{I_r u_g}{J_p}$ and $\frac{I_r u_g}{J_r}$ disappear when the voltages corresponding to an equilibrium point $\bar{x} = 0$ are inserted, as Equations (2.20) - (2.21) show.

$$\frac{I_r u_g}{J_p} = \frac{I_r K_v}{J_p}(V_r + V_l - V_f - V_b) = \frac{I_r K_v}{J_p}(0 + 0 - 0 - 0) = 0 \tag{2.20}$$

$$\frac{I_r u_g}{J_r} = \frac{I_r K_v}{J_r}(V_r + V_l - V_f - V_b) = \frac{I_r K_v}{J_r}(0 + 0 - 0 - 0) = 0 \tag{2.21}$$

The input matrix B is found using the nonlinear Equations (2.4) - (2.9), going on to partially derive these with respect to the input vector (2.15) resulting in Equation (2.22).

$$B(x) = \begin{bmatrix} \frac{\partial f_1}{\partial V_f} & \frac{\partial f_1}{\partial V_b} & \cdots & \frac{\partial f_1}{\partial V_l} \\ \frac{\partial f_2}{\partial V_f} & \frac{\partial f_2}{\partial V_b} & \cdots & \frac{\partial f_2}{\partial V_l} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_6}{\partial V_f} & \frac{\partial f_6}{\partial V_b} & \cdots & \frac{\partial f_6}{\partial V_l} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{I_r q}{J_r} K_v & -\frac{I_r q}{J_r} K_v & \frac{LK_f}{J_r} + \frac{I_r q}{J_r} K_v & -\frac{LK_f}{J_r} + \frac{I_r q}{J_r} K_v \\ \frac{LK_f}{J_p} + \frac{I_r p}{J_p} K_v & -\frac{LK_f}{J_p} + \frac{I_r p}{J_p} K_v & -\frac{I_r p}{J_p} K_v & -\frac{I_r p}{J_p} K_v \\ -\frac{K_t}{J_y} & -\frac{K_t}{J_y} & \frac{K_t}{J_y} & \frac{K_t}{J_y} \end{bmatrix} \tag{2.22}$$

Proceeding to find the output matrix C by partially deriving the output Equations (2.16) by the state vector (2.14) resultig in Equation 2.23.

$$C = \begin{bmatrix} \frac{\partial \theta_r}{\partial \theta_r} & \frac{\partial \theta_r}{\partial \theta_p} & \frac{\partial \theta_r}{\partial \theta_y} & \frac{\partial \theta_r}{\partial p} & \frac{\partial \theta_r}{\partial q} & \frac{\partial \theta_r}{\partial r} \\ \frac{\partial \theta_p}{\partial \theta_r} & \frac{\partial \theta_p}{\partial \theta_p} & \frac{\partial \theta_p}{\partial \theta_y} & \frac{\partial \theta_p}{\partial p} & \frac{\partial \theta_p}{\partial q} & \frac{\partial \theta_p}{\partial r} \\ \frac{\partial \theta_y}{\partial \theta_r} & \frac{\partial \theta_y}{\partial \theta_p} & \frac{\partial \theta_y}{\partial \theta_y} & \frac{\partial \theta_y}{\partial p} & \frac{\partial \theta_y}{\partial q} & \frac{\partial \theta_y}{\partial r} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.23}$$

D will be a zero matrix as Equation 2.24 shows, because none of the inputs (2.15) will be found in any of the output Equations (2.16).

$$D = \begin{bmatrix} \frac{\partial \theta_r}{\partial V_f} & \frac{\partial \theta_r}{\partial V_b} & \frac{\partial \theta_r}{\partial V_r} & \frac{\partial \theta_r}{\partial V_l} \\ \frac{\partial \theta_p}{\partial V_f} & \frac{\partial \theta_p}{\partial V_b} & \frac{\partial \theta_p}{\partial V_r} & \frac{\partial \theta_p}{\partial V_l} \\ \frac{\partial \theta_y}{\partial V_f} & \frac{\partial \theta_y}{\partial V_b} & \frac{\partial \theta_y}{\partial V_r} & \frac{\partial \theta_y}{\partial V_l} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.24}$$

Further on, the matrices $A(x)$ and $B(x)$ are linearized around the equilibrium point $\bar{x} = 0$. The constants are given values from Table 2.1, inserting these will return the linearized matrices in (2.25).

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.424 & -0.424 \\ 0.424 & -0.424 & 0 & 0 \\ -0.0327 & -0.0327 & 0.0327 & 0.0327 \end{bmatrix} \tag{2.25}$$

These matrices will be used to develop a controller using the Linear-quadratic regulator approach in the following section.

2.4 Linear–quadratic regulator

The Linear-Quadratic Regulator (LQR) is a feedback controller which optimizes the control of a system. Given a Linear Time-Invariant (LTI) system as presented in the previous Section 2.3. With a cost function as presented in Equation (2.26).

$$J = \int_0^\infty (x^T(t)Qx(t) + u^T(t)Ru(t))\ dt \tag{2.26}$$

The weighting matrices for Q and R are stated in the hover laboratory guide [18].

$$Q = diag(350,\ 350,\ 500,\ 20,\ 20,\ 0)$$
$$R = diag(0.01,\ 0.01,\ 0.01,\ 0.01)$$

The Q matrix penalizes bad performance by increasing its values for each of the states in the state vector (2.14). The R matrix penalizes the effort of the actuators by increasing its values, the inputs, which is the control vector (2.15).

The feedback control law that minimizes the cost is given by Equation (2.27).

$$u(t) = -Kx(t) \tag{2.27}$$

Whereas the gain matrix K is computed by Equation (2.28).

$$K = R^{-1}(B^T P) \tag{2.28}$$

Where R and B are known matrices. P is found by solving a continuous time algebraic Riccati equation, which is a non-linear equation typical to find in optimal control problems. The Riccati equation occurs as in Equation (2.29).

$$0 = A^T P + PA - PBR^{-1}B^T P + Q \tag{2.29}$$

Where P is the unknown. The formula is solved for P, which appears as a symmetric n-x-n matrix. The rest of the matrices are known values at this stage.

Using the weighting matrices Q and R with the computed state-space matrices A and B, the control gain K can be computed through the cost function (2.11). To calculate this, the Matlab function `lqr` can be used.

$$K = \begin{bmatrix} 0 & 132.2876 & -111.8034 & 0 & 36.1989 & -41.3293 \\ 0 & -132.2876 & -111.8034 & 0 & -36.1989 & -41.3293 \\ 132.2876 & 0 & 111.8034 & 36.1989 & 0 & 41.3293 \\ -132.2876 & 0 & 111.8034 & -36.1989 & 0 & 41.3293 \end{bmatrix}$$

While a linearized control system utilizing LQR shows good control performance, it is only effective in operating over a limited operational range. Looking into a non-linear system will help achieve reliable quad-rotor control over a wider operational range. Taking into account the fuzzy-model approach, we should be able to improve the performance of controlling the 3DOF Hover.

2.4.1 Simulation

The state-space model is implemented in `HoverSim.m` and thereafter simulated in `SimulationLQR.m`. Using `ode45` to solve the differential equations that make up the model (2.4) - (2.9), the formerly produced LQR is tested. The natural response of the system is shown in Figure 2.4.

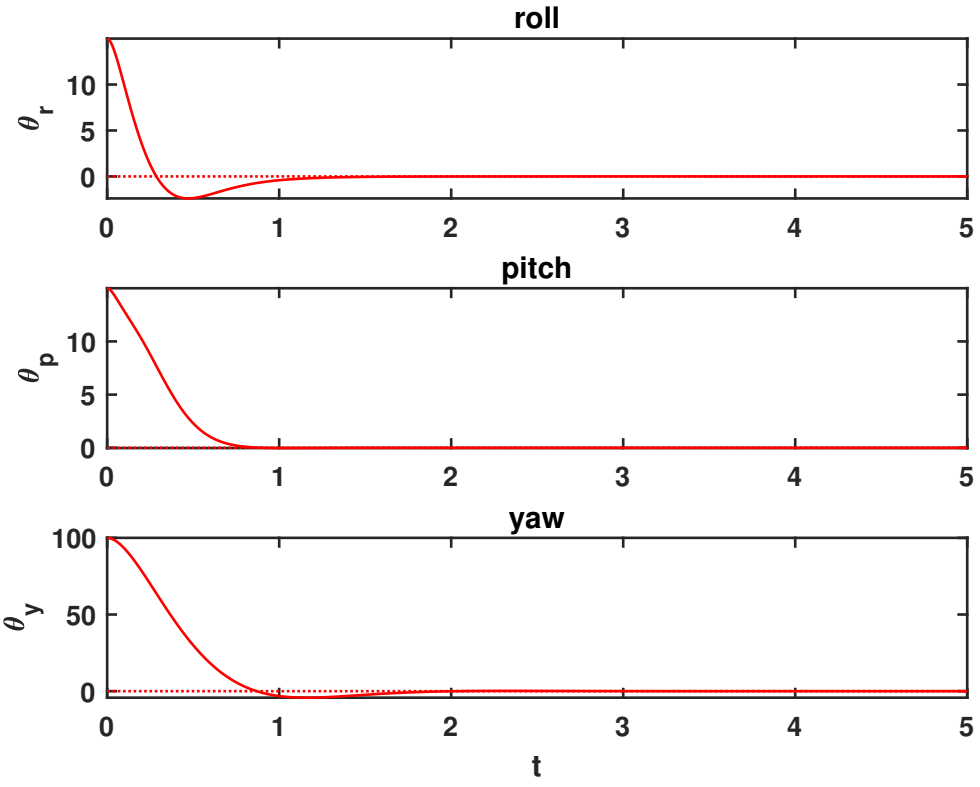


Fig 2.4: The LQR response with initial condition $x(0) = [15°, 15°, 100°, 0, 0, 0]^T$ and reference $ref = 0$.

Figure 2.5 shows the state trajectory with a non zero reference for the same LQR linearized around $\bar{x} = 0$.

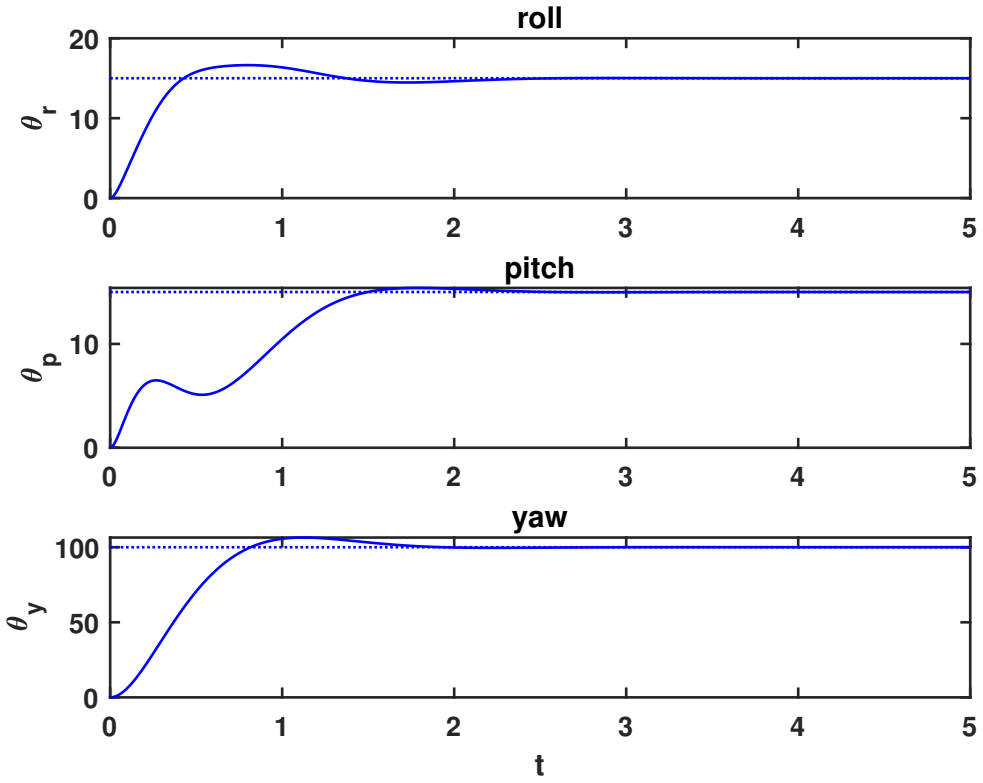


Fig 2.5: The LQR response with initial condition $x(0) = 0$ and reference $ref = [15°, 15°, 100°, 0, 0, 0]^T$.

Now the error $e(t)$ substitutes for the state vector $x(t)$ in the cost function (2.26) and feedback control law giving the cost function in Equation (2.30).

$$J = \int_0^\infty (e(t)^T Q e(t) + u(t)^T R u(t))\ dt \tag{2.30}$$
$$u(t) = -Ke(t)$$

One should not assume however that the controller linearized around $\bar{x} = 0$ will give the lowest quadratic cost in this scenario. A plot comparing simulations with differently linearized controllers is shown in Figure 2.6, indicating that linearizing around the reference might improve performance.

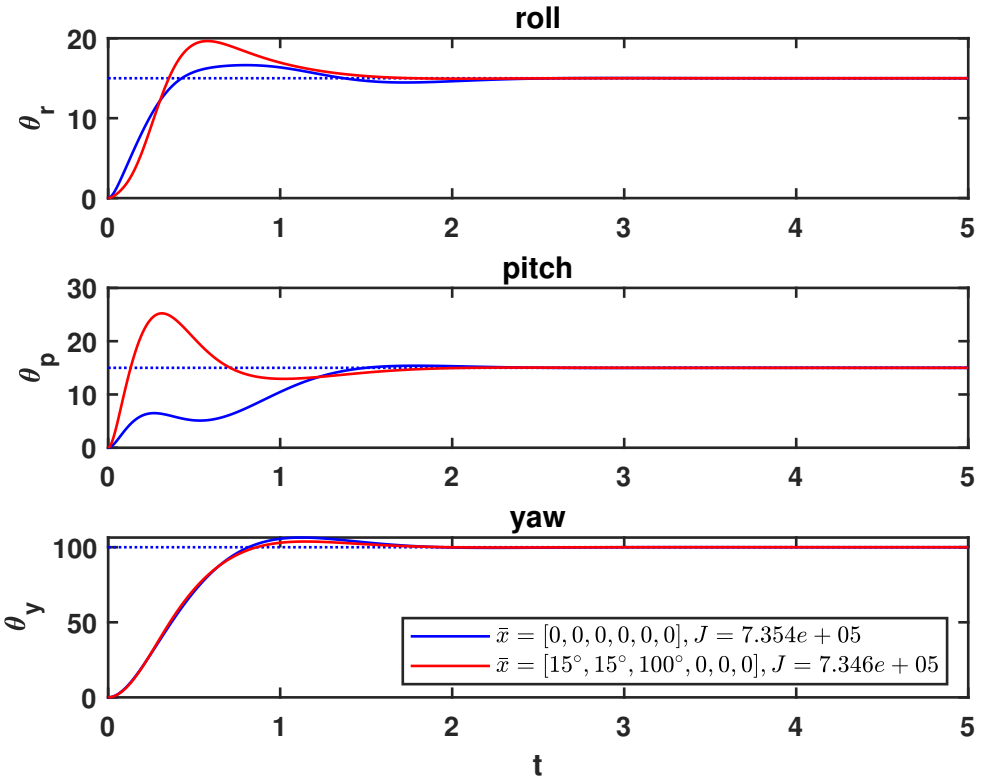


Fig 2.6: The LQR response for two differently linearized controllers with initial condition $x(0) = 0$ and reference $ref = [15°, 15°, 100°, 0, 0, 0]^T$.
J is the quadratic cost (3.10).

2.4.2 Integral states

In the search of a system that responds with a zero steady state error, a useful solution is to add a control integral action of the error signal. By having a control integral action, the system will have close to zero steady state errors if the closed loop system is stable.

To include integral action to the controller, new states e_r, e_p and e_y are added to the state vector (2.14):

- e_r - referring to error at the roll-axis
- e_p - referring to error at the pitch-axis
- e_y - referring to error at the yaw-axis

Which gives the new state vector $x_e(t)$ as in Equation (2.31):

$$x_e(t) = [\theta_r, \theta_p, \theta_y, p, q, r, e_r, e_p, e_y]^T \tag{2.31}$$

These new states scale the error occurring from the reference value at each axis. When derivated, the error states are set to equal the angular states [θ_r, θ_p, θ_y]. We can then calculate three equations for the new states giving our system integral action:

$$\begin{aligned} \dot{e_r} &= x_1 = \theta_r \\ \dot{e_p} &= x_2 = \theta_p \\ \dot{e_y} &= x_3 = \theta_y \end{aligned} \quad \rightarrow \quad \begin{aligned} e_r &= \int_0^t x_1(t)dt \\ e_p &= \int_0^t x_2(t)dt \\ e_y &= \int_0^t x_3(t)dt \end{aligned}$$

This also results in three new linear equations to the system, denoted as f_7, f_8 and f_9 (2.32) - (2.34) in addition to the six original nonlinear Equations (2.4) - (2.9).

$$\dot{e_r} = f_7 = x_1 = \theta_r \tag{2.32}$$
$$\dot{e_p} = f_8 = x_2 = \theta_p \tag{2.33}$$
$$\dot{e_y} = f_9 = x_3 = \theta_y \tag{2.34}$$

Simultaneously, the matrices $A(x), B(x), Q$, and K will expand by n additional number of rows and columns corresponding to the new states. R stays the same as the control vector (2.15) does not change.

The state matrix $A_{IA}(x)$ will now be a $9x9$ matrix:

$$A_{IA}(x) = \begin{bmatrix} 0 & 0 & 0 & 1 & sin\theta_r tan\theta_p & cos\theta_r tan\theta_p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & cos\theta_r & -sin\theta_r & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{sin\theta_r}{cos\theta_p} & \frac{cos\theta_r}{cos\theta_p} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{J_p - J_y}{J_r} r & \frac{J_p - J_y}{J_r} q & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{J_y - J_r}{J_p} r & 0 & \frac{J_y - J_r}{J_p} p & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{J_r - J_p}{J_y} q & \frac{J_r - J_p}{J_y} p & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

While the input matrix B_{IA} with integral action is given by:

$$B_{IA}(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{I_{rq}}{J_r}K_v & -\frac{I_{rq}}{J_r}K_v & \frac{LK_f}{J_r} + \frac{I_{rq}}{J_r}K_v & -\frac{LK_f}{J_r} + \frac{I_{rq}}{J_r}K_v \\ \frac{LK_f}{J_p} + \frac{I_{rp}}{J_p}K_v & -\frac{LK_f}{J_p} + \frac{I_{rp}}{J_p}K_v & -\frac{I_{rp}}{J_p}K_v & -\frac{I_{rp}}{J_p}K_v \\ -\frac{K_t}{J_y} & -\frac{K_t}{J_y} & \frac{K_t}{J_y} & \frac{K_t}{J_y} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Finally, the weighting matrix Q_{IA} decided with integral action:

$$Q_{IA} = diag(350,\ 350,\ 500,\ 20,\ 20,\ 0,\ 1,\ 1,\ 1)$$

An example of what a computed gain matrix K with integral action will look like:

$$K = \begin{bmatrix} 0 & 134.21 & -113.65 & 0 & 36.28 & -41.67 & 0 & 7.07 & -5.0 \\ 0 & -134.21 & -113.65 & 0 & -36.28 & -41.67 & 0 & -7.07 & -5.0 \\ 134.21 & 0 & 113.65 & 36.28 & 0 & 41.67 & -7.07 & 0 & 5.0 \\ -134.21 & 0 & 113.65 & -36.28 & 0 & 41.67 & -7.07 & 0 & 5.0 \end{bmatrix}$$

It is observed that for $Q_{IA} = diag(350,\ 350,\ 500,\ 20,\ 20,\ 0,\ i_r,\ i_p,\ i_y)$ with $i_x = 0$, the LQR was not able to build due to Q and A having unobservable modes on the imaginary axis. However, his aspect is not further investigated in this report.

Part II

Implementing Fuzzy Logic

Chapter 3

Takagi-Sugeno-Kang approach

3.1 Takagi-Sugeno-Kang fuzzy inference system

An approach to control is the Takagi-Sugeno-Kang (TSK) fuzzy inference system. While the Mamdani system [6] is well-suited to human input, by easily applying rules that correlate to the system behavior, the TSK system is to a greater extent compatible with mathematical analysis. Additionally, TSK systems offer greater computational efficiency. The Mamdani systems are more often associated with a considerable computational burden as the final process involves calculating the centroid of a two-dimensional area while TSK systems use a weighted average of a few data points [19].

The system consists of input variables, input membership functions (mf), rules and linear output membership functions, structured according to Figure 3.1.

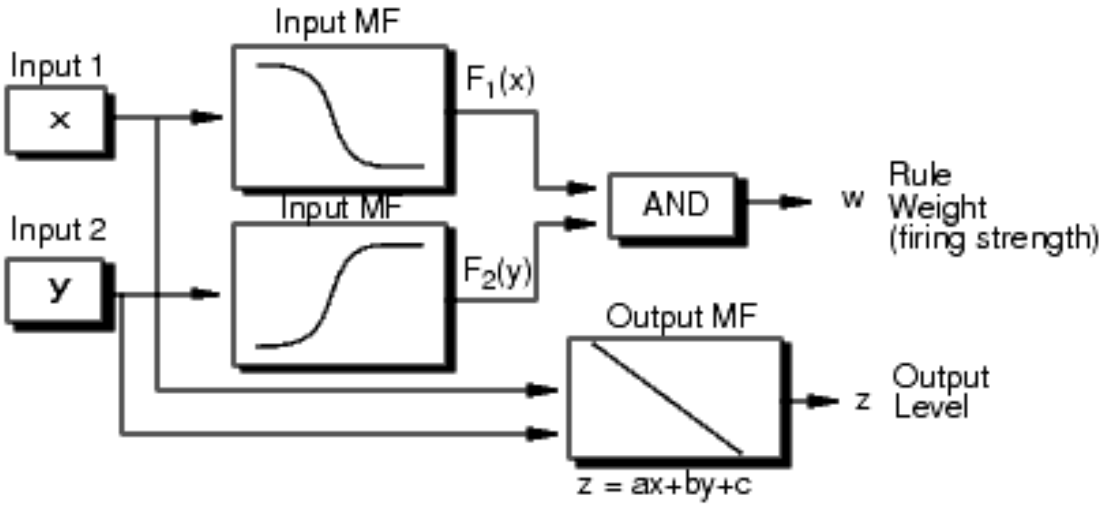


Fig 3.1: Diagram of a TSK system rule implementation with two inputs, x and y [1].

By each rule, the system is given a weight and an output in the form of either a linear equation or a constant. The final output is then calculated as a weighted average in Equation (3.1).

$$u = \frac{\sum_{i=1}^{n} w_i \cdot (-K_i) \cdot x}{\sum_{i=1}^{n} w_i} \tag{3.1}$$

3.1.1 Membership function for the input

The implementation of a fuzzy inference system will start off with the fuzzification of the input variables. The input values are assigned a degree of membership connected to each of the membership functions resulting in a vector known as a fuzzy set. The membership functions may look like in Figure 3.2.

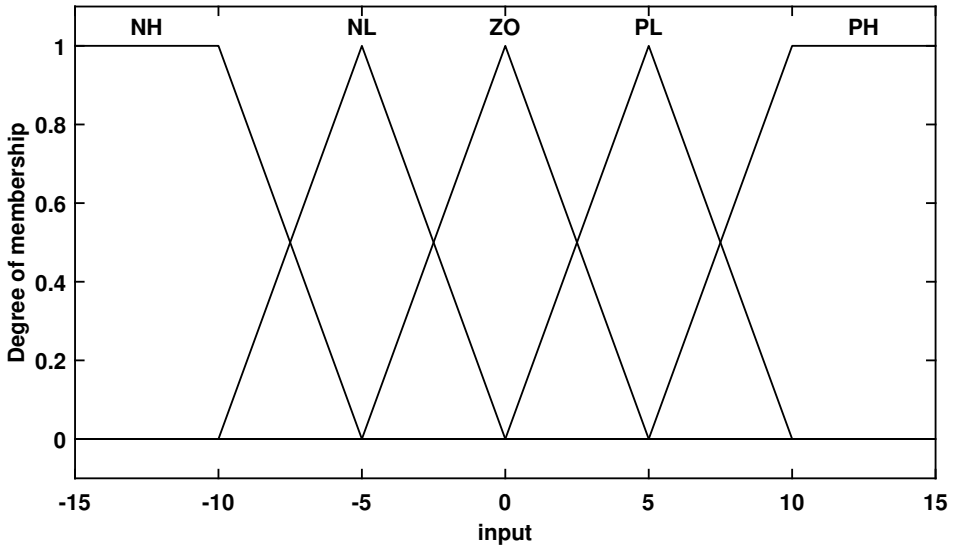


Fig 3.2: The input membership functions.

The membership functions are usually given names that allow for interpretation in a linguistic manner. For an input with 5 membership functions, a method of naming that would make for easy interpretation goes like this:

- NH - Negative High
- NL - Negative Low
- ZO - Zero
- PL - Positive Low
- PH - Positive High

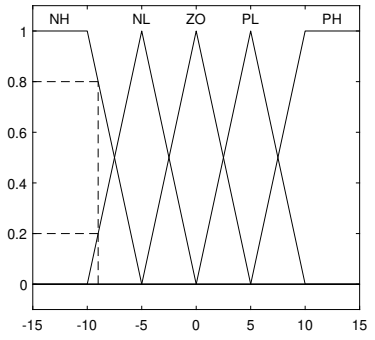


Fig 3.3: Fuzzification example.

To produce the fuzzy values, the input variable is simply ran through the membership functions, noting the degree of membership for each of them. The outcome is a vector with a value between 0 and 1 corresponding to each membership function.

Figure 3.3 shows how an input $x = -9$ results in the fuzzy set given in Equation (3.2), where each element are the membership functions with the input inserted. The membership functions will be denoted as F.

$$[F_{NH}(x), F_{NL}(x), F_{ZO}(x), F_{PL}(x), F_{PH}(x)] = [0.8, 0.2, 0, 0, 0] \tag{3.2}$$

3.1.2 Rule Structure and fuzzy model

This section will start by introducing the rules syntax, then explaining how the rules are interpreted and molded into the final output of the controller.

For the controller being presented in this chapter, each rule describes each input by a membership function. All possible fuzzy combinations of angles and velocities are present in the list of rules, giving a fuzzy model (3.3) - (3.6) [8].

rule 1:

$$\textbf{IF } x_1 \textbf{ IS } F_{11} \textbf{ AND } x_2 \textbf{ IS } F_{21} \textbf{ AND } \ldots \textbf{ AND } x_j \textbf{ IS } F_{j1} \tag{3.3}$$

$$\textbf{THEN} \begin{cases} \dot{x}(t) = A_1 x(t) + B_1 u(t) \\ y(t) = C_1 x(t) \end{cases} \tag{3.4}$$

$$\vdots$$

rule n:

$$\textbf{IF } x_1 \textbf{ IS } F_{1k} \textbf{ AND } x_2 \textbf{ IS } F_{2k} \textbf{ AND } \ldots \textbf{ AND } x_j \textbf{ IS } F_{jk} \tag{3.5}$$

$$\textbf{THEN} \begin{cases} \dot{x}(t) = A_n x(t) + B_n u(t) \\ y(t) = C_n x(t) \end{cases} \tag{3.6}$$

For $n = k^j$ rules, j inputs and k membership functions, where A_n, B_n and C_n are the state, input and output matrices describing the linearized system. Here, F_{jk} is the k'th membership function for input j.

The model (2.4) - (2.9) is linearized as in Subsection 2.3.2 about the operating points corresponding to the membership function the input is compared to[1]

Each rule correlates with a gain matrix computed for the linearized model with weighting matrices Q and R as in Section 2.4. This gives the input-output relation presented in Equation (3.7).

$$\text{rule 1:} \quad u_1(t) = -K_1 x(t) \qquad \dots \qquad \text{rule n:} \quad u_n(t) = -K_n x(t) \tag{3.7}$$

The rules are built using the Boolean operator AND(x,y), and while Boolean operators are used to compare variables that are strictly true or false, the variables of a fuzzy inference system are on a sliding scale. Thus a method of interpreting the Boolean operators for fuzzy variables is needed. The Zadeh operators stated in Table 3.1 provide just this functionality, therefore MIN(x,y) will be used in place of AND(x,y) when implementing the rules in code.

Boolean	Fuzzy
AND(x,y)	MIN(x,y)
OR(x,y)	MAX(x,y)
NOT(x)	1 - x

Table 3.1: Zadeh operators.

The rule weight (firing strength) for n rules is computed as in Equation (3.8)-(3.9).

$$w_1 = min(F_{11}(x_1), F_{21}(x_2), \ \dots \ , F_{j1}(x_j)) \tag{3.8}$$

$$\vdots$$

$$w_n = min(F_{1k}(x_1), F_{2k}(x_2), \ \dots \ , F_{jk}(x_j)) \tag{3.9}$$

Where $F_{jk}(x_j)$ is the k'th membership function evaluated for the state j. The rule weights w_n and gain matrices K_n are then used to calculate the output (3.1).

[1] The operating point connected with a membership function is the input which would give a 100% degree of membership.

3.2 Testing the TSK fuzzy logic controller

Comparing the TSK Fuzzy Logic Controller (FLC) to the linear controller is essential in the search of making fuzzy control of a system as good as possible.

One way of quantifying the performance of a controller is with the cost function (3.10) mentioned in Section 2.4. A sum (for discrete time systems) or integral (for continuous time systems) of the variables of interest in our system. When referring to linear systems, the LQR-design guarantees the lowest cost function available. By designing a custom controller, it is possible to get close to the cost function of the LQR controller, but it never results in a cheaper cost function.

The linear controller is only expected to work better close to the equilibrium points. The TSK fuzzy controller on the other hand, when tuned, is expected to work significantly better when far from equilibrium points. The fuzzy logic helps the controller be more precise with its membership functions than a linear controller which is only an approximation of the nonlinear system around the equilibrium point.

3.2.1 Obtaining the gain matrices

Unique gain matrices are obtained by calculating the matrices A and B corresponding to different operating points and computing K for the same Q, R and B matrices.

The operating points are linearly spaced within the limits which are mentioned in Section 2.1. The angles are bounded by $\pm 37.5^\circ \approx \pm 0.6545$ rad for the roll and pitch, while the yaw can spin freely both clockwise and counter-clockwise. The yaw can spin freely due to a slip ring mechanism, which transmit the encoder and motor signals. The resulting intervals are as follows:

$$
\begin{aligned}
-37.5^\circ < \theta_r < +37.5^\circ \qquad & -1\ \frac{rad}{s} < p < +1\ \frac{rad}{s} \\
-37.5^\circ < \theta_p < +37.5^\circ \qquad & -1\ \frac{rad}{s} < q < +1\ \frac{rad}{s} \\
-180^\circ < \theta_y < +180^\circ \qquad & -1\ \frac{rad}{s} < r < +1\ \frac{rad}{s}
\end{aligned}
$$

The operating points can then be handled as a Dirac delta function located at the center of the membership functions as illustrated in Figure 3.4.

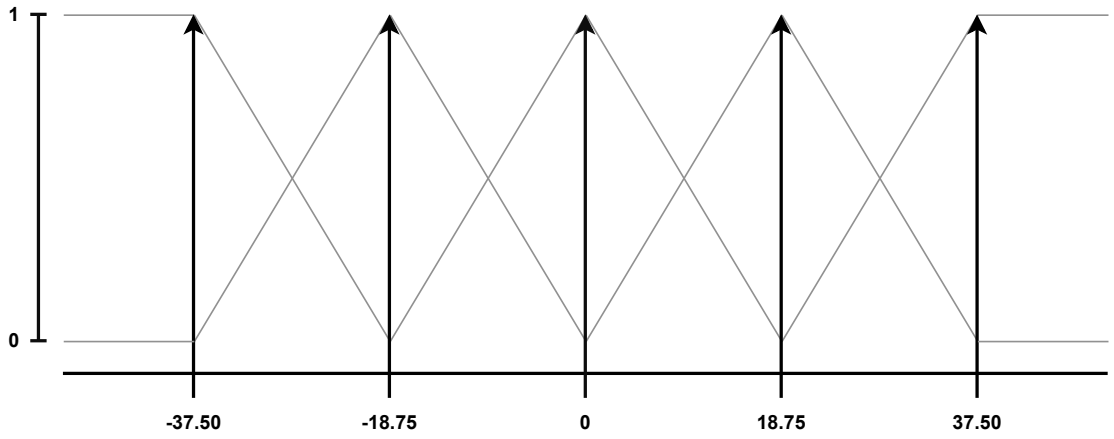


Fig 3.4: Operating points described by Dirac delta functions.

Section 3.1.2 states that the list of rules is the result of the combinatoric permutation with repetition, meaning every combination of membership functions is present. Consequently, as the membership functions also reflect corresponding operating points, such a list can be created with $n = k^j = 5^5 = 3125$ elements (k membership functions and j inputs):

$$\begin{bmatrix} -37.50 & -37.50 & -1 & -1 & -1 \\ -18.75 & -37.50 & -1 & -1 & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 18.75 & 37.50 & 1 & 1 & 1 \\ 37.50 & 37.50 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \text{Operating points, Rule 1} \\ \text{Operating points, Rule 2} \\ \vdots \\ \text{Operating points, Rule 3124} \\ \text{Operating points, Rule 3125} \end{bmatrix}$$

The function `RuleMaker.m` creates the list of operating points and rules in MATLAB. It should be noted that as θ_y is not a variable in either the state matrix $A(x)$ or input matrix $B(x)$, no fuzzy inference will be made for that state.

The state matrix $A(x)$ (2.19) is evaluated at the different set of operating points:

$$A_1 = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{16} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & a_{56} \\ a_{61} & \dots & a_{65} & a_{66} \end{bmatrix}_{\substack{\theta_r = -37.50 \\ \theta_p = -37.50 \\ p\text{=-1} \\ q\text{=-1} \\ r\text{=-1}}} = \begin{bmatrix} 0 & 0 & 0 & 1.000 & 0.467 & -0.609 \\ 0 & 0 & 0 & 0 & 0.793 & 0.609 \\ 0 & 0 & 0 & 0 & -0.767 & 1.000 \\ 0 & 0 & 0 & 0 & 0.993 & 0.993 \\ 0 & 0 & 0 & -0.993 & 0 & -0.993 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{16} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & a_{56} \\ a_{61} & \dots & a_{65} & a_{66} \end{bmatrix}_{\substack{\theta_r = -18.75 \\ \theta_p = -37.50 \\ p\text{=-1} \\ q\text{=-1} \\ r\text{=-1}}} = \begin{bmatrix} 0 & 0 & 0 & 1.000 & 0.247 & -0.727 \\ 0 & 0 & 0 & 0 & 0.947 & 0.321 \\ 0 & 0 & 0 & 0 & -0.405 & 1.194 \\ 0 & 0 & 0 & 0 & 0.993 & 0.993 \\ 0 & 0 & 0 & -0.993 & 0 & -0.993 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\vdots$$

$$A_{3124} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{16} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & a_{56} \\ a_{61} & \dots & a_{65} & a_{66} \end{bmatrix}_{\substack{\theta_r = 18.75 \\ \theta_p = 37.50 \\ p\text{=1} \\ q\text{=1} \\ r\text{=1}}} = \begin{bmatrix} 0 & 0 & 0 & 1.000 & 0.247 & 0.727 \\ 0 & 0 & 0 & 0 & 0.947 & -0.321 \\ 0 & 0 & 0 & 0 & 0.405 & 1.194 \\ 0 & 0 & 0 & 0 & -0.993 & -0.993 \\ 0 & 0 & 0 & 0.993 & 0 & 0.993 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_{3125} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{16} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & a_{56} \\ a_{61} & \dots & a_{65} & a_{66} \end{bmatrix}_{\substack{\theta_r = 37.50 \\ \theta_p = 37.50 \\ p\text{=1} \\ q\text{=1} \\ r\text{=1}}} = \begin{bmatrix} 0 & 0 & 0 & 1.000 & 0.467 & 0.609 \\ 0 & 0 & 0 & 0 & 0.793 & -0.609 \\ 0 & 0 & 0 & 0 & 0.767 & 1.000 \\ 0 & 0 & 0 & 0 & -0.993 & -0.993 \\ 0 & 0 & 0 & 0.993 & 0 & 0.993 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Then the input matrix $B(x)$ (2.22) is evaluated at the different set of operating points:

$$B_1 = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \vdots & \vdots & \vdots & \vdots \\ b_{61} & b_{62} & b_{63} & b_{64} \end{bmatrix}_{\substack{\theta_r = -37.50 \\ \theta_p = -37.50 \\ p\text{=-1} \\ q\text{=-1} \\ r\text{=-1}}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.060 & 0.060 & 0.364 & -0.484 \\ 0.364 & -0.484 & 0.060 & 0.060 \\ -0.033 & -0.033 & 0.033 & 0.033 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \vdots & \vdots & \vdots & \vdots \\ b_{61} & b_{62} & b_{63} & b_{64} \end{bmatrix}_{\substack{\theta_r = -18.75 \\ \theta_p = -37.50 \\ p\text{=-1} \\ q\text{=-1} \\ r\text{=-1}}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.060 & 0.060 & 0.364 & -0.484 \\ 0.364 & -0.484 & 0.060 & 0.060 \\ -0.033 & -0.033 & 0.033 & 0.033 \end{bmatrix}$$

$$\vdots$$

$$B_{3124} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \vdots & \vdots & \vdots & \vdots \\ b_{61} & b_{62} & b_{63} & b_{64} \end{bmatrix}_{\substack{\theta_r = 18.75 \\ \theta_p = 37.50 \\ p=1 \\ q=1 \\ r=1}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.060 & -0.060 & 0.484 & -0.364 \\ 0.484 & -0.364 & -0.060 & -0.060 \\ -0.033 & -0.033 & 0.033 & 0.033 \end{bmatrix}$$

$$B_{3125} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \vdots & \vdots & \vdots & \vdots \\ b_{61} & b_{62} & b_{63} & b_{64} \end{bmatrix}_{\substack{\theta_r = 37.50 \\ \theta_p = 37.50 \\ p=1 \\ q=1 \\ r=1}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.060 & -0.060 & 0.484 & -0.364 \\ 0.484 & -0.364 & -0.060 & -0.060 \\ -0.033 & -0.033 & 0.033 & 0.033 \end{bmatrix}$$

The input matrix $B(x)$ is independent of the states θ_r, θ_p and θ_y, but are dependent on p, q and r. Which is why B_1 and B_2 stay the same despite changing the roll axis operating point. B_{3125} looks different as the value of the velocities are at a maximum compared to a minimum in the first input matrices.

For the state matrix A_i and input matrix B_i, with the same R and Q matrix as stated in Section 2.4, the LQR gain matrices (2.28) are:

$$K_1 = \begin{bmatrix} 57.030 & 1.6149 & -181.25 & 5.4866 & 33.585 & -62.098 \\ -11.001 & -153.07 & 62.074 & 2.9447 & -40.674 & -23.057 \\ 100.61 & 82.644 & 115.25 & 31.160 & 4.8153 & 50.201 \\ -146.64 & 68.815 & 3.9264 & -39.592 & 2.2734 & 34.955 \end{bmatrix}$$

$$K_2 = \begin{bmatrix} 51.134 & 59.9112 & -169.23 & 5.2946 & 32.693 & -64.142 \\ 4.4314 & -161.98 & -6.0933 & 3.3518 & -39.834 & -27.846 \\ 96.358 & 52.943 & 142.238 & 31.063 & 4.5419 & 52.790 \\ -151.92 & 49.023 & 33.080 & -39.710 & 2.5990 & 39.198 \end{bmatrix}$$

$$\vdots$$

$$K_{3124} = \begin{bmatrix} -12.975 & 161.47 & -3.2855 & -3.3688 & 39.728 & -27.902 \\ -43.429 & -60.130 & -172.16 & -5.4091 & -32.612 & -64.000 \\ 153.41 & -41.320 & 37.938 & 39.859 & -2.5380 & 39.942 \\ -97.008 & -60.025 & 137.51 & -31.081 & -4.5783 & 51.959 \end{bmatrix}$$

$$K_{3125} = \begin{bmatrix} 2.5328 & 152.68 & 64.708 & -2.9160 & 40.514 & -23.099 \\ -49.544 & -1.7931 & -184.36 & -5.6592 & -33.580 & -61.987 \\ 148.87 & -63.197 & 11.527 & 39.788 & -2.1458 & 35.809 \\ -101.86 & -87.695 & 108.13 & -31.213 & -4.8890 & 49.278 \end{bmatrix}$$

The gain matrices have been calculated through the function `GainMatrix.m` in MATLAB, which uses the built-in function `lqr` to find the gain matrix that minimizes the cost function (2.26). Similarly, when an integral action is used, the different gain matrices are calculated through the function `GainMatrixInt.m`.

3.2.2 Implementation using Fuzzy Logic Toolbox in MATLAB

A way of implementing the fuzzy model (3.3) - (3.6) defined in the previous section, is with the Fuzzy Logic Toolbox function in MATLAB. The toolbox allows for the implementation of TSK and Mamdani type systems both at the command line and with an interactive user interface. However, this report will only benefit from the command line (script) implementation of TSK systems.

The input variables are based on the state vector (2.14) and added with the function `addInput`. The yaw angle θ_y is omitted due to its absence in the state matrix (2.19). The remaining states will act as an input on the inference system, with limitations for each being set according to the physical observations made in Section 2.1. The function `addMF` is used to add input membership functions with a structure as in Figure 3.2.

The output variables in this case will be the voltages to the different motors, which is the control vector (2.15). These are added with the function `addOutput`, where each voltage will be set as an output. `addMF` is then used again, in this case to add linear functions of the inputs that will generate an output. These linear functions are extracted from the gain matrix in the following manner,

$$u(x(t)) = \begin{cases} u_{V_f} = a_1\theta_r + a_2\theta_p + \ldots + a_6 r \\ u_{V_b} = b_1\theta_r + b_2\theta_p + \ldots + b_6 r \\ u_{V_r} = c_1\theta_r + c_2\theta_p + \ldots + c_6 r \\ u_{V_l} = d_1\theta_r + d_2\theta_p + \ldots + d_6 r \end{cases}, \quad \text{with } K_n = \begin{bmatrix} a_1 & a_2 & \ldots & a_6 \\ b_1 & b_2 & \ldots & b_6 \\ c_1 & c_2 & \ldots & c_6 \\ d_1 & d_2 & \ldots & d_6 \end{bmatrix}$$

where K_n is the gain matrix corresponding to the n'th rule.

Lastly, the rules of the TSK system are added with the function `addRule`.
Code 3.1 demonstrates the syntax,

```
ruleList = [[1 1 0 1 1 1 1 1 1 1 1 1];...
            [5 1 0 1 1 4 2 2 2 2 1 1];...
            [3 3 0 3 3 3 3 3 3 3 1 1];...

FIS = addRule(FIS,ruleList)
```

Code 3.1: Example of numeric rule description.

which would correspond to the following rules:

if θ_r is NH and ... and r is NH then Vf is F_1 and ... and Vl is F_1
if θ_r is PH and ... and r is PL then Vf is F_2 and ... and Vl is F_2
if θ_r is ZO and ... and r is ZO then Vf is F_3 and ... and Vl is F_3

A total of twelve digit values are set for each rule. The first six describe the antecedent (input), the next four describe the consequent (output). The penultimate column digits are for setting the rule weight which must be a number from 0 to 1. The last digit chooses the fuzzy operator: 1 for AND(x,y), 2 for OR(x,y).

Despite the fact that TSK systems are quite simple to implement, as the only operations involved are sums and products, the runtime with the Fuzzy Logic Toolbox presents an issue. The toolbox was slow when conducting experiments with more than 3 membership functions, which is necessary for this report.

For this reason the class `tsk.m` was written. This class constructs the membership functions, list of rules and gain matrices based on desired properties such as number of membership functions and how they are arranged. Later on in Chapter 4, the class will be built upon to support weighted linearization.

In conjunction with the aforementioned class, a function `EvalFuzzy.m` was written. This function handles the fuzzification, rule weighting and output calculation both in simulation and in real-time testing.

Table 3.2 shows how the user-defined class and function provide vastly improved runtimes, whereas the code was (at times) not able to complete in the toolbox's environment at all within a feasible timeframe.

No. of: Membership Functions	No. of: Rules	Runtime: User-Defined Class	Runtime: Fuzzy Logic Toolbox	Percentage improvement
3	243	00:00:02	00:00:18	91 %
5	3125	00:00:15	00:43:48	99 %
7	16807	00:01:13	unable to complete	100 %
9	59049	00:04:19	unable to complete	100 %
11	161051	00:12:37	unable to complete	100 %
13	371293	00:30:22	unable to complete	100 %

Table 3.2: Comparing runtime of User-Defined classes and functions with the Fuzzy Logic Toolbox in MATLAB.

The Fuzzy Logic Toolbox implementation is done via the script `SugenoFIS.m`.

3.2.3 Simulation

Before implementing and testing the Fuzzy Logic Controller (FLC) on the 3DOF Hover, simulations are done in MATLAB. The performance of the controller is quantified by the discrete-time cost function as the simulated signal under consideration is with a fixed-step size. The function is presented in Equation (3.10):

$$J = x_N^T Q x_N + \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) \tag{3.10}$$

The function will give a single value that should be comparably lower for the fuzzy controller than the linear controller. The cost is calculated for the trajectory followed by the Hover as it goes from an initial condition $x(0) \neq 0$ to $x = 0$. An example of the fuzzy logic and linear controller response side by side is shown in Figure 3.5, plotted from the code script `SimulationFIS.m` and `SimulationLQR.m`.

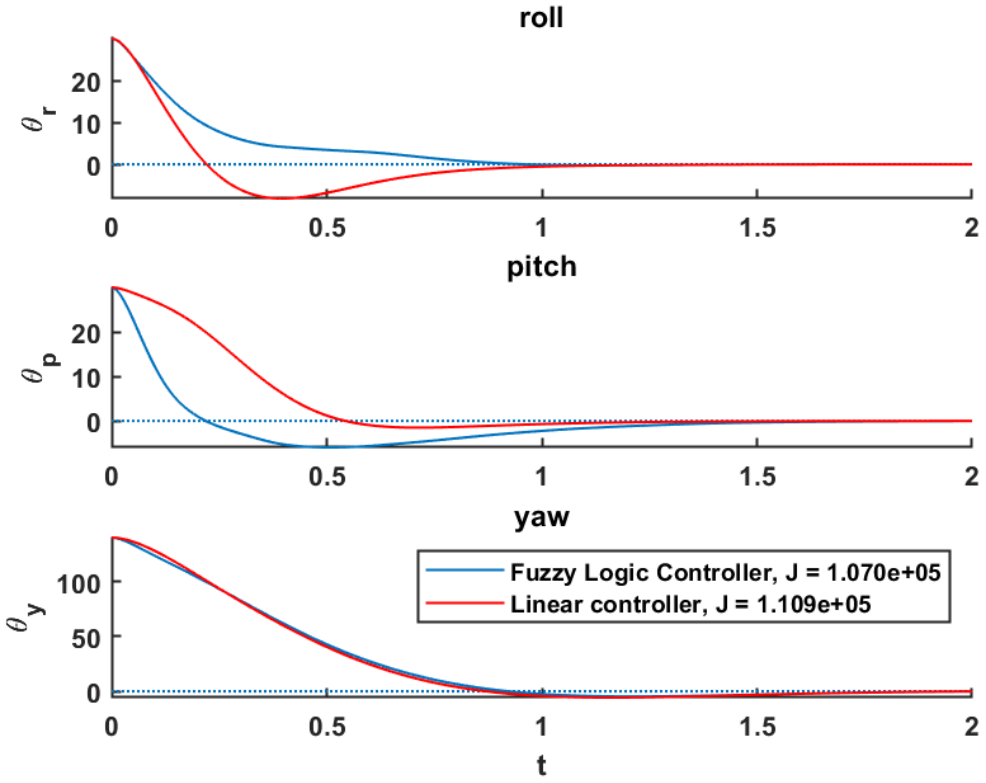


Fig 3.5: Natural response of the system with $x(0) = [30°, 30°, 140°, 0, 0, 0]$ for both the FLC and linear controller.

In Figure 3.5 the states $x = [\theta_r, \theta_p, \theta_y]^T$ had an initial condition of $x(0) = [30°, 30°, 140°]$. The simulation was set to have a small step size (.001) and to run long enough for the angles to stabilize around the desired degree-reference which was set to equal zero for both controllers.

Comparing the simulation of the FLC to the simulation of the linear controller, the following quadratic cost (3.10) was obtained:

$$J_{FUZZY} = 1.07 \cdot 10^5$$

$$J_{LINEAR} = 1.109 \cdot 10^5$$

Figure 3.6 shows that for repeated simulations with increasing initial condition of $x = [\theta_r, \theta_p, \theta_y]^T$, the percentage improvement of the cost will rise when comparing the FLC to the linear controller proposed in Section 2.4.

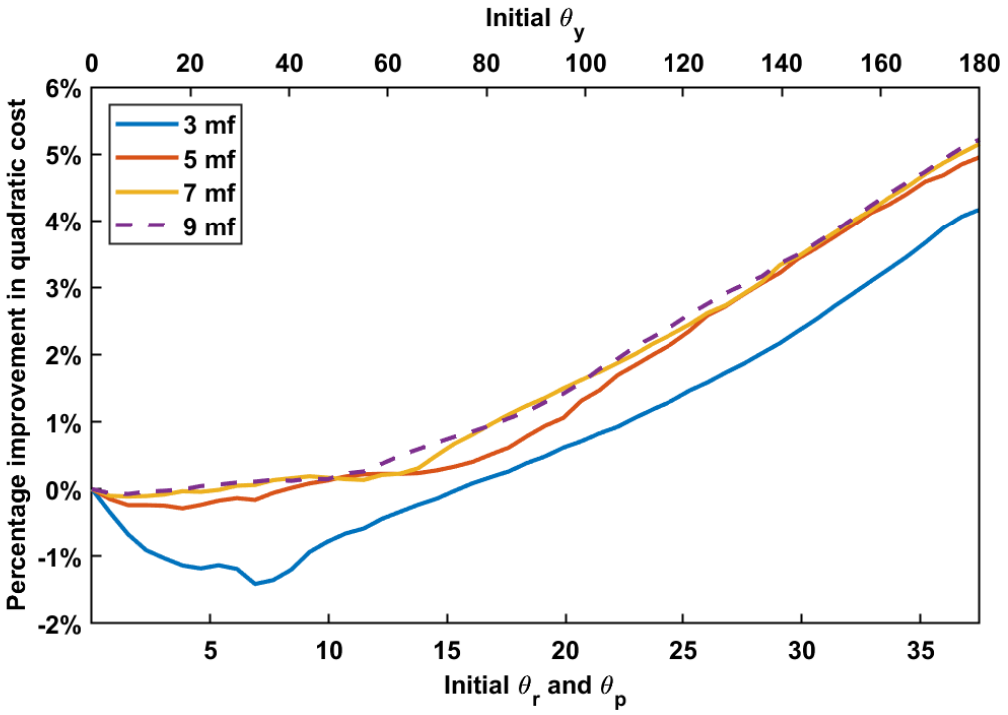


Fig 3.6: Percentage improvement in quadratic cost for the FLC over the linear controller in simulation. The initial condition $x(0)$ is increased for each simulation.

The earlier part of the plot displaying experiments with lower initial conditions $x(0)$ prefers the linear controller over the FLC. This aligns with the theory that a FLC should only improve performance when far away from the equilibrum point $\bar{x} = 0$. There is also a diminishing return in performance as the controllers get more complex, meaning the controller with 5 membership functions might be preferred over one with 7 or 9.

3.2.4 Real-time testing

The FLC and linear controller are compared against each other as was done in simulation, giving a representation of the quality of the produced fuzzy inference system by a percentage improvement in quadratic cost. A first step into making a good FLC would be to make it give a lower cost function than the linear controller. The finite-horizon, continuous-time cost function is shown in Equation (3.11).

$$J = \int_{t_0}^{t_1} (x^T(t)Qx(t) + u^T(t)Ru(t))\ dt \tag{3.11}$$

Which in turn gives Equation (3.12) when the state vector (2.14) and input vector (2.15) are inserted.

$$= \int_{t_0}^{t_1} \begin{bmatrix} \theta_r & \theta_p & \theta_y & p & q & r \end{bmatrix} \begin{bmatrix} 500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 350 & 0 & 0 & 0 & 0 \\ 0 & 0 & 350 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix} \begin{bmatrix} \theta_r \\ \theta_p \\ \theta_y \\ p \\ q \\ r \end{bmatrix}$$

$$+ \begin{bmatrix} V_f & V_b & V_r & V_l \end{bmatrix} \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix} \begin{bmatrix} V_f \\ V_b \\ V_r \\ V_l \end{bmatrix} \tag{3.12}$$

To begin comparing the controllers, the Simulink model provided by Quanser is adjusted to allow for fuzzy control and calculation of the cost function (3.11). The Simulink schemes can be seen in Appendix D, where Figure D.1 in particular illustrates the cost calculation.

Figure 3.7 presents the different stages of an experiment. Firstly the initial condition is reached using a linear controller, thereafter either the fuzzy or linear controller drives all the states to zero. Each experiment is run twice, once for both the linear and fuzzy controller. Lastly, the quadratic cost is calculated in the timeframe 10-20 seconds and compared for both controllers.

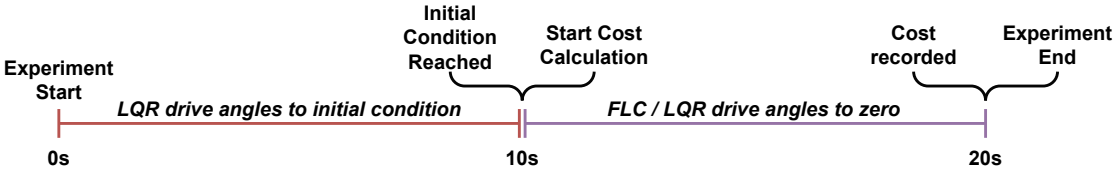


Fig 3.7: The different stages of an experiment testing the FLC.

To reduce variance in the initial conditions and results between experiments, the following countermeasures are implemented:

Ramp reference - To reduce oscillations and unwanted behavior when approaching the initial condition. Using a step reference results in a behavior that is increasingly unstable as the reference is set higher, meaning an increased load on the rig and deviation in initial conditions between experiments.

Integral states - To reduce the steady state error when approaching the initial condition. The addition of integral states is discussed in Section 2.4.2.

Estimation over multiple experiments - To reduce the variance in results. While it's possible that either controller may be better than the other for a single experiment, it would be more interesting to see how the controllers compare when anomalies are disregarded. The median is used as it accounts for outliers in the dataset which may skew the results significantly.

An experiment is recorded in Figure 3.8 with the aforementioned description.

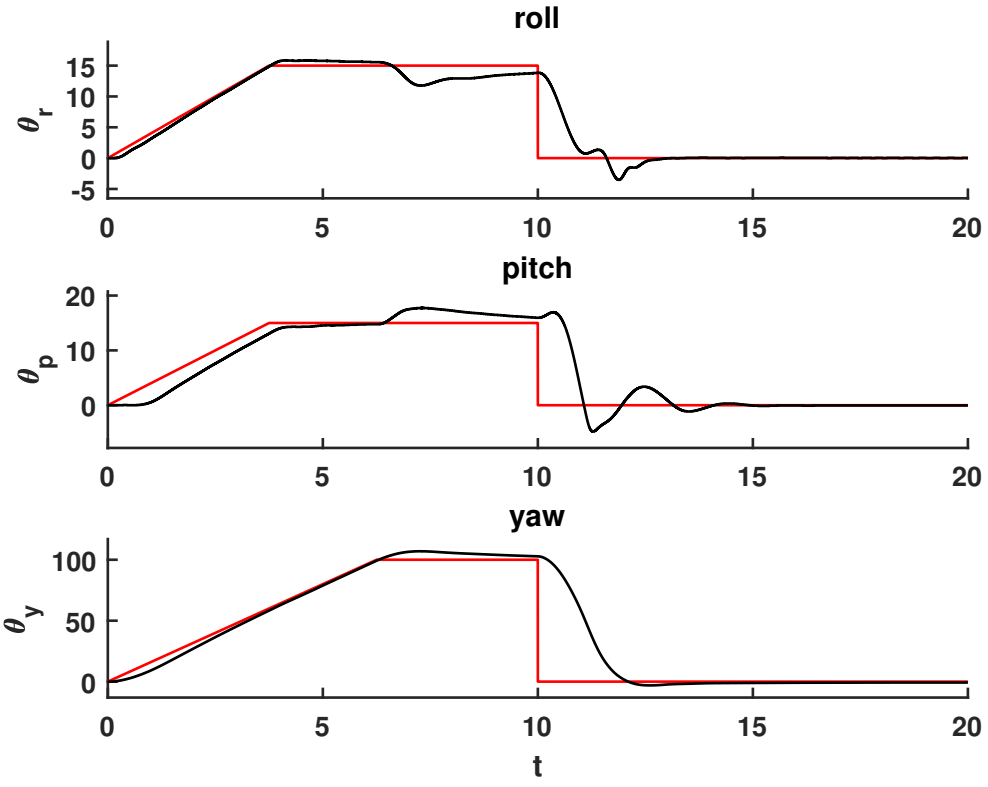


Fig 3.8: Real-time response of the FLC with an initial condition $x(10) \approx [15°, 15°, 100°]$.

A series of experiments are recorded, each time incrementing the initial condition away from the equilibrium point $\bar{x} = 0$. The results are plotted as the percentage improvement in quadratic cost, as was done in simulation, giving the Figure 3.9.

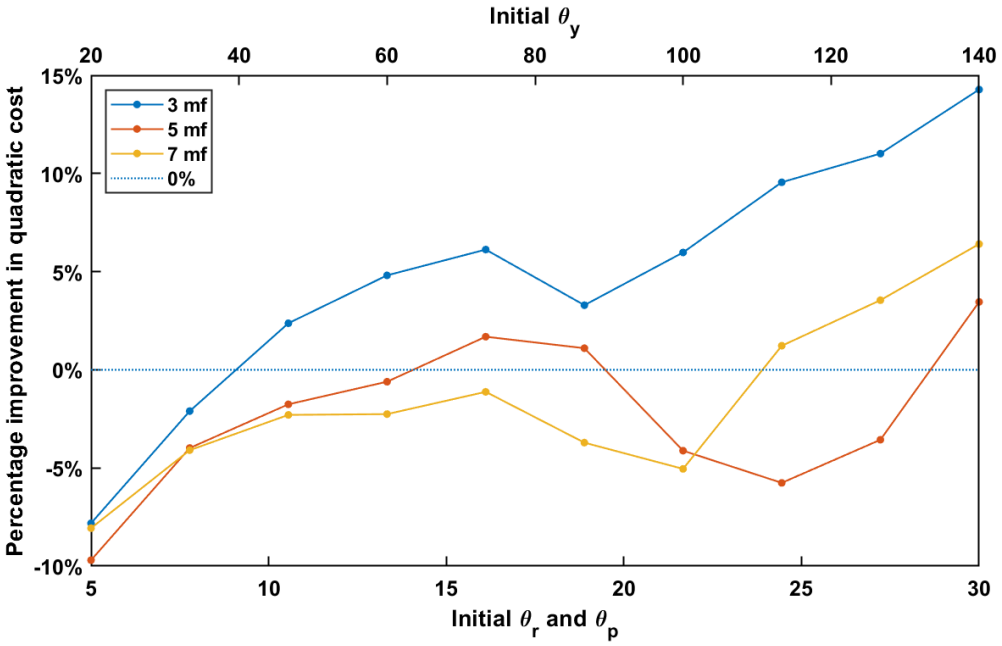


Fig 3.9: Percentage improvement in cost for the FLC over the linear controller in real-time. Each data point is an estimation from 10 identical experiments.

The outcome of Figure 3.9 aligns with expectations in some ways, and fails in others. While a FLC with 3 membership functions clearly shows improved performance, more complex controllers decline.

It will be mentioned that in early testing, the results did not agree with the theory that an addition of fuzzy inference would increase performance. If the mathematical model describing a system is known, then applying a fuzzy logic controller design should improve the overall performance of the system. The poor results raised the question about the accuracy of the nonlinear model, which ended up being the culprit. Appendix A presents experimental results along with comments and corrections, narrating the evolvement of the mathematical model.

In it's current condition, the 5 mf controller has no correlation with the 3 mf controller, as visualized in Figure 3.10.

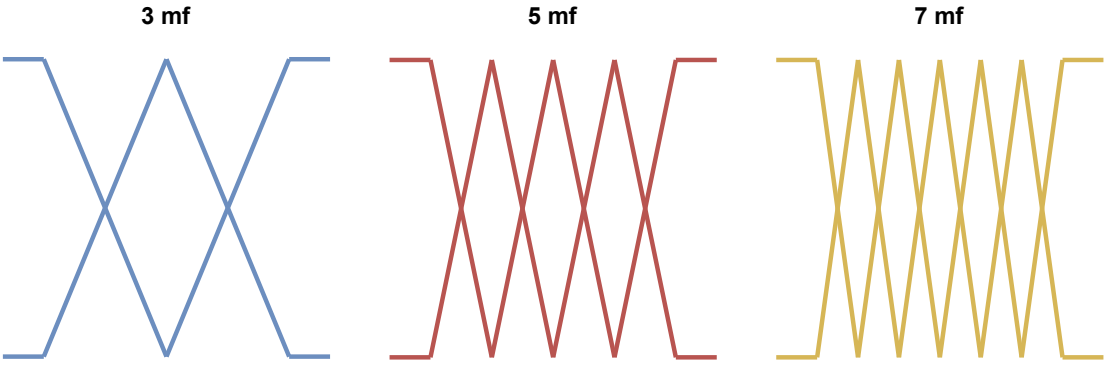


Fig 3.10: Linearly spaced membership functions with a common width.

To include the robustness of the 3 mf controller in subsequent controllers, instead of linearly spacing the membership functions with the same widths, each new scheme uses the previous one as a basis. Figure 3.11 presents the new method.

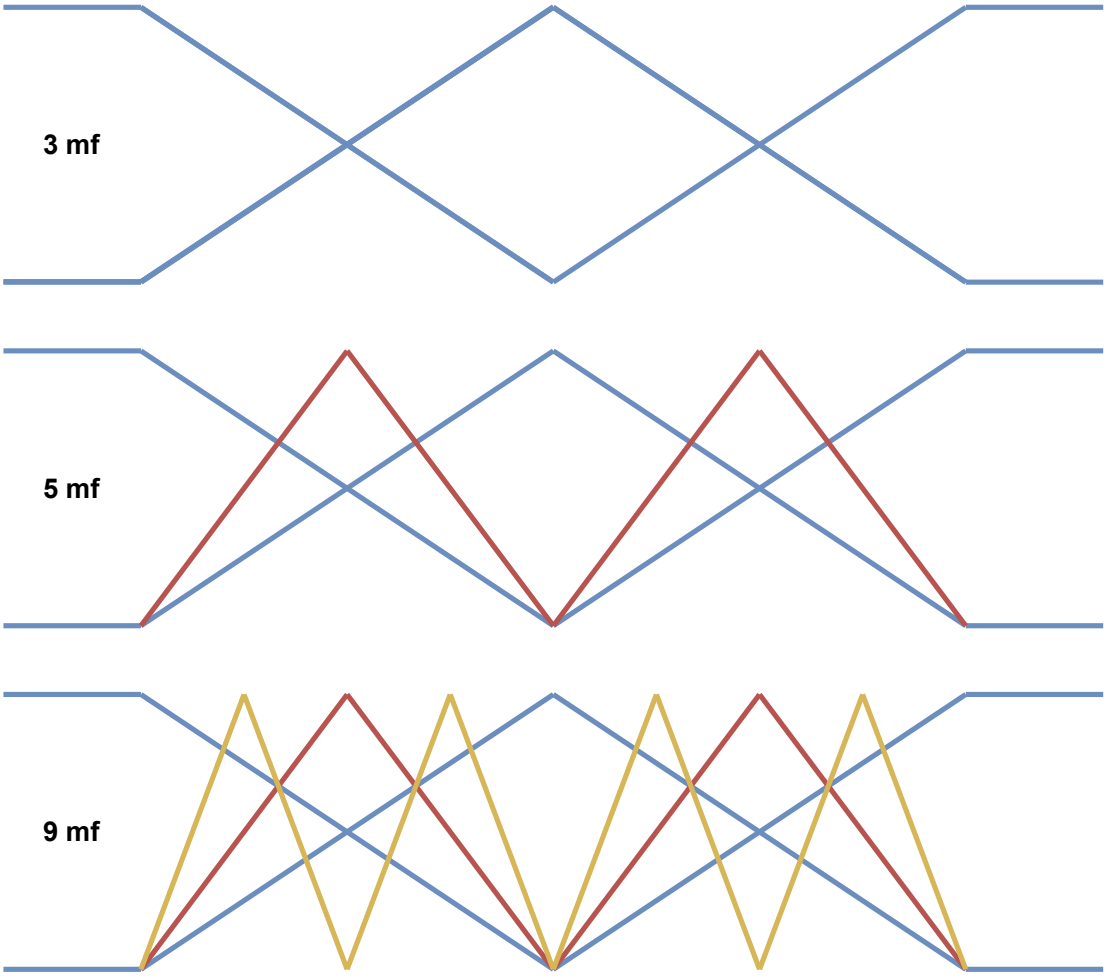


Fig 3.11: Adding membership functions with the previous arrangement as a basis.

The new scheme does significantly improve the controller, as seen in Figure 3.12.

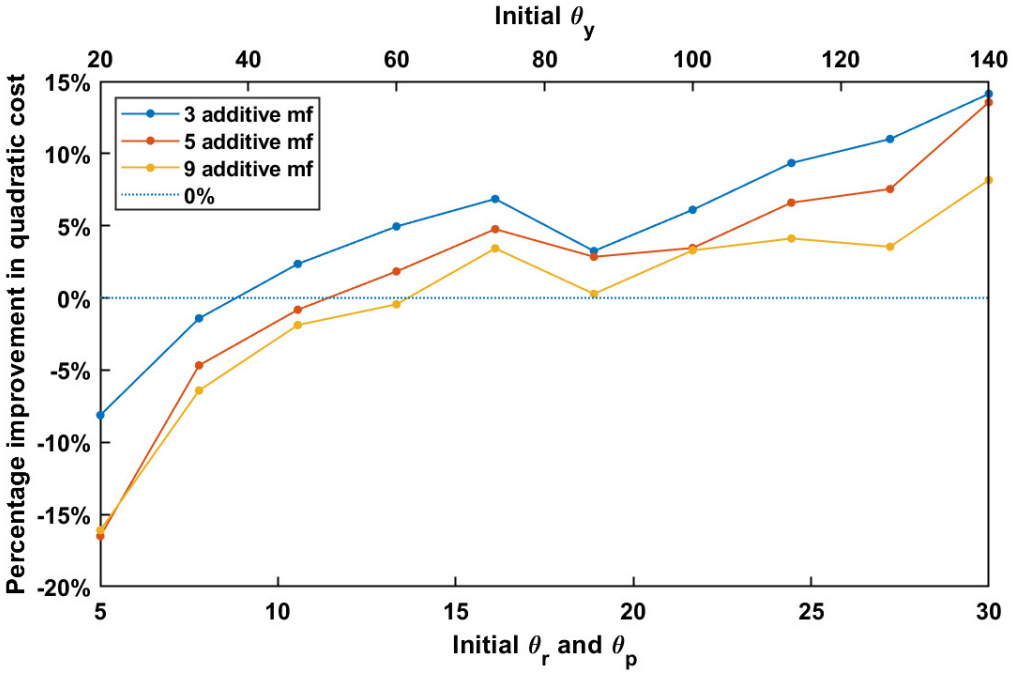


Fig 3.12: Percentage improvement in cost for the FLC over the linear controller in real-time. Each data point is an estimation from 10 identical experiments.

Ideally, the 5 mf controller would add information to the 3 mf controller, by the addition of 2 membership functions, thus performing even better. However, this is not the case. There continues to be a decline in performance when adding membership functions when testing in real-time. Nonetheless, the new membership function arrangement does improve overall performance when compared to Figure 3.9.

Manipulating the shape and position of the membership functions surely affects performance. Therefore an experiment was conducted, with a wider membership function range for the velocities as demonstrated by Figure 3.13, and an additive arrangement as in Figure 3.11.

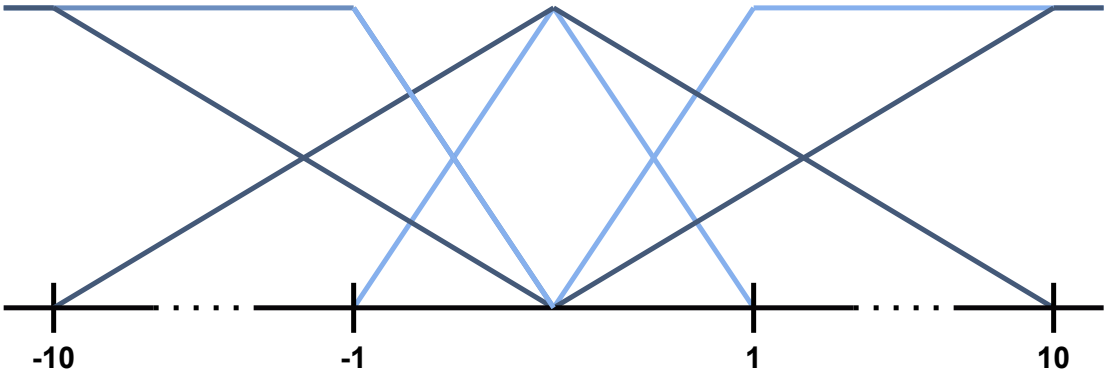


Fig 3.13: Widening the membership function range for the velocities.

The range [−10 10] was chosen experimentally, as it would increase performance compared to a more narrow or wide range. The results are displayed in Figure 3.14.

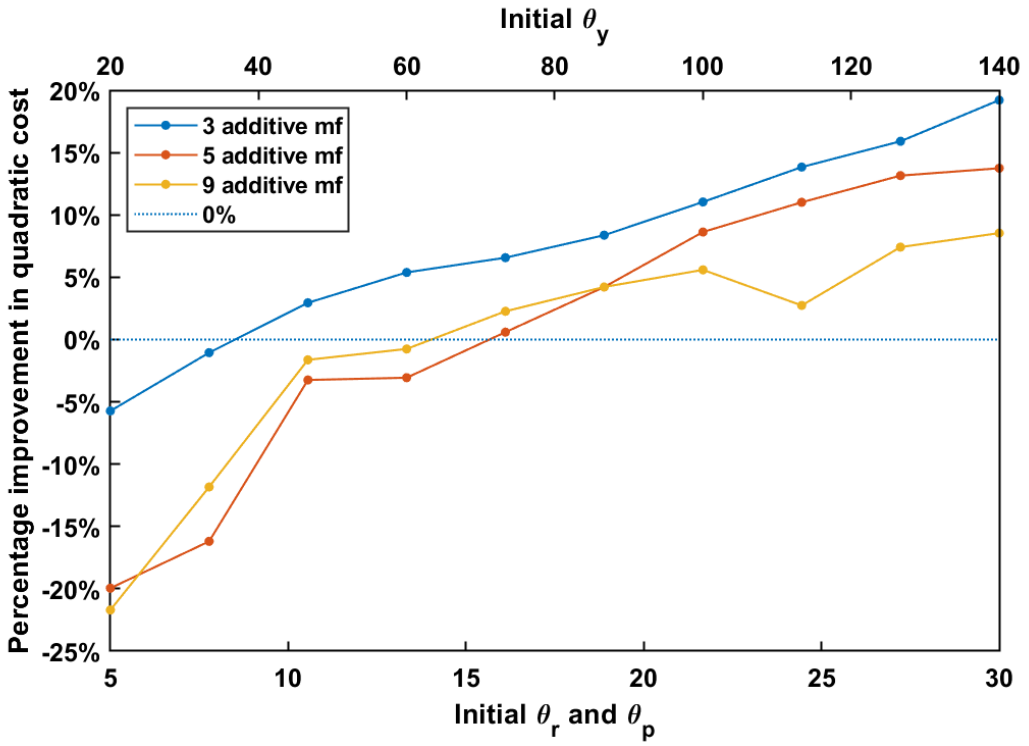


Fig 3.14: Percentage improvement in cost for the FLC over the linear controller in real-time. Each data point is an estimation from 4 identical experiments.

Increasing the membership function range, and consequently changing the operating points at which the gain matrices are built around has a big impact on performance, mostly for the 3 mf controller. More work should therefore be contributed to deciding the arrangement of the velocity input membership functions.

Chapter 4

Weighted Linearization Approach

> *... a generalization of the linearization technique in which the computation of the Jacobian matrices at the state trajectory of interest is replaced by the multiple integral over the state and input spaces of the Jacobian matrix functions multiplied by a weighting function. ...* [20]

4.1 Weighted linearization of the 3DOF hover system

The second approach worth looking into for the 3DOF hover system is weighted linearization of the nonlinear system. Taking into account the bigger picture of the system performance, by linearizing multiple scenarios with help from a weighting function. In this approach, the weighting function will be triangular, to match the existing membership functions in Chapter 3.1.1. An example of how it could be done with a Gaussian weighting function is shown in Appendix B.

Figure 4.1 depicts how weighted linearization includes neighbouring operating points by having multiple Dirac delta functions that are differently weighted.

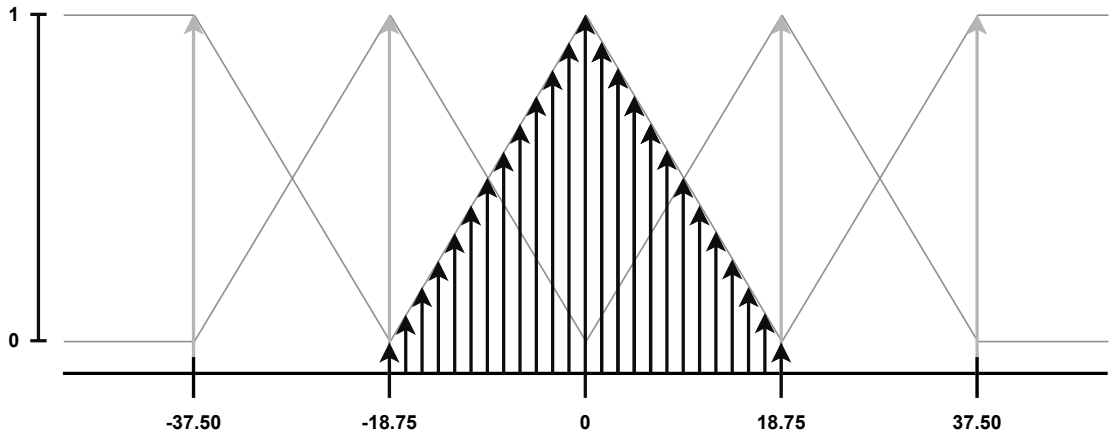


Fig 4.1: Operating points described by Multiple Dirac delta functions.

Followingly, the matrices $A(t)$ and $B(t)$ are produced from the Equations (4.1) and (4.2).

$$A(t) = \int \cdots \int_{\mathbb{R}^{n+m}} \rho(x,u,t) \frac{Df}{Dx}(x,u,t) dx du \tag{4.1}$$

$$B(t) = \int \cdots \int_{\mathbb{R}^{n+m}} \rho(x,u,t) \frac{Df}{Du}(x,u,t) dx du \tag{4.2}$$

Where $\rho(x,u,t)$ is the weighting function. In essence, the factors $\frac{Df}{Dx}(x,u,t)$ and $\frac{Df}{Du}(x,u,t)$ are the different elements of the state matrix $A(x)$ (2.19) and input matrix $B(x)$ (2.22) derived from taylor series expansion.

In general, the weighting function is any function that fulfills the requirements of (4.3).

$$\int \cdots \int_{\mathbb{R}^{n+m}} \rho(x,u,t) dx du = 1 \qquad \forall t \in \mathbb{R} \tag{4.3}$$

As the weighting functions need to accord with Equation (4.3), the membership functions can not be used directly. Instead, the height of the membership functions are altered as illustrated in Figure 4.2, giving it an integral of 1.

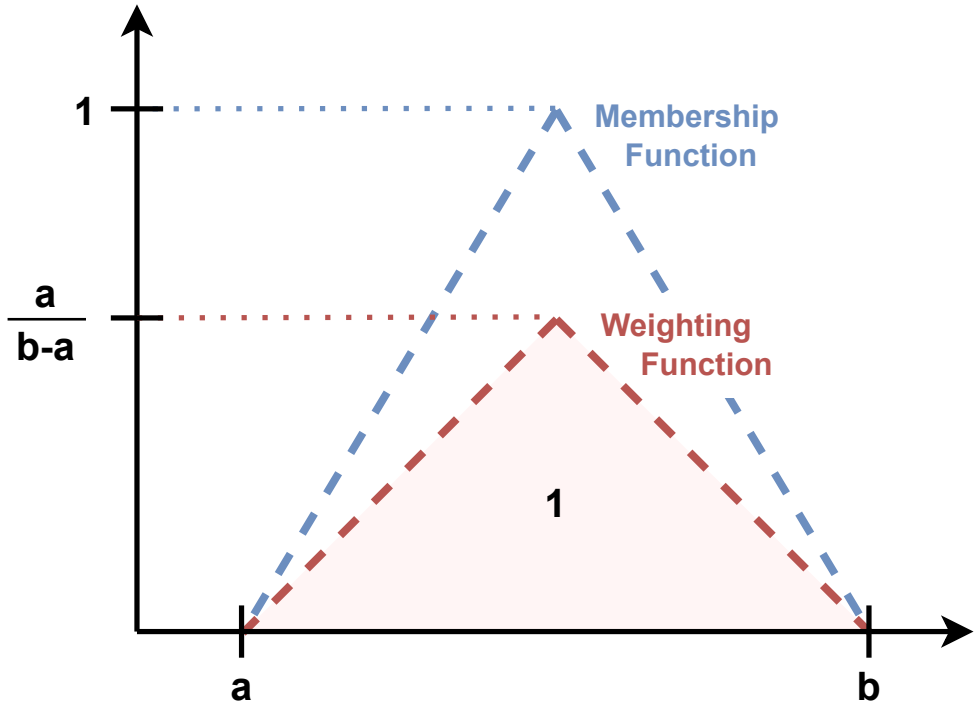


Fig 4.2: Relationship between a membership function and a weighting function.

As Figure 4.3 illustrates. By breaking up the operating range of state variables (2.14), the weighting functions can be translated into linear functions of the form $\rho(x,t) = ax+b$ for the blue areas. The remaining red areas correspond to a weighting function $\rho = 0$.

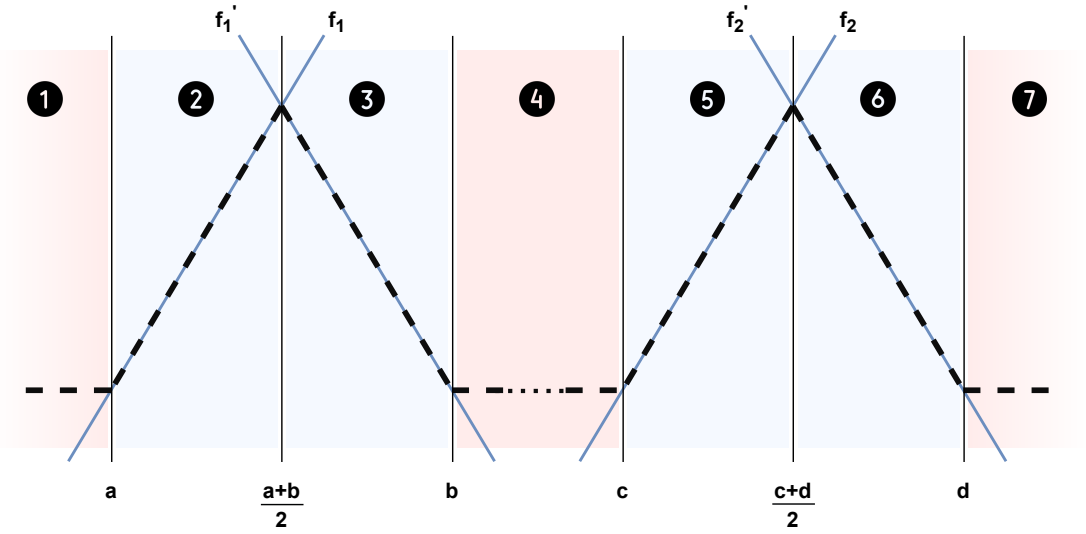


Fig 4.3: The weighting functions divided into several linear equations.

The linear functions $f(x,t)$ and $f'(x,t)$ can be found by the Equations (4.4) and (4.5), where x is the state.

$$f(x,t) = \frac{4}{(b-a)^2}(x-a) \tag{4.4}$$

$$f'(x,t) = \frac{-4}{(b-a)^2}(x-b) \tag{4.5}$$

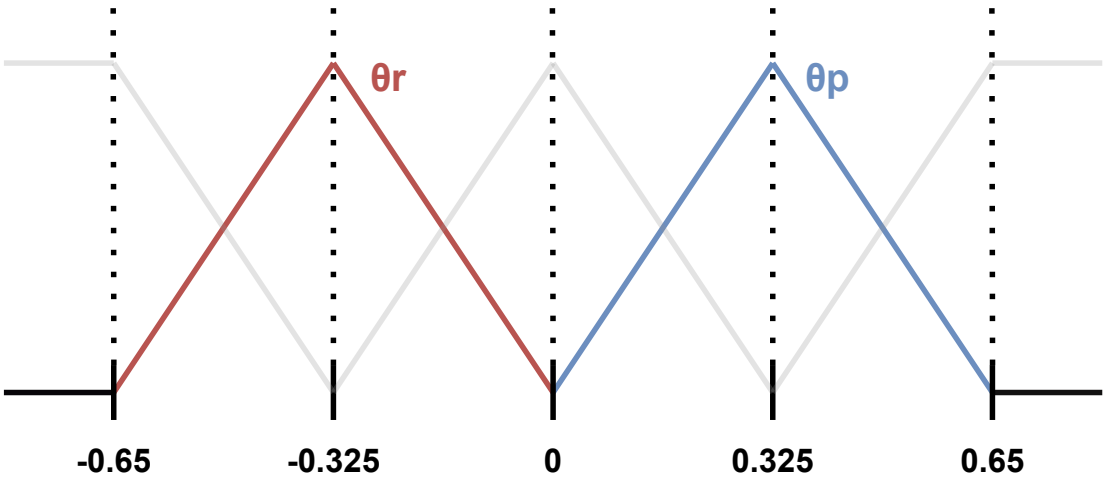
4.1.1 Example

The following example demonstrates how the weighted linearization is performed for a single element in the state matrix a_{15}. Starting with the Equation (4.1).

$$A(t) = \int \cdots \int_{\mathbb{R}^{n+m}} \rho(x,u,t)\frac{Df_1}{Dr}(x,u,t)dxdu$$
$$a_{15} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \rho_{\theta_r}(x,u,t)\rho_{\theta_p}(x,u,t)(cos\theta_r tan\theta_p)(x,u,t)d\theta_r d\theta_p$$

Where the weighting function $\rho(x,u,t)$ is in the general form $ax+b$, and $\frac{Df_1}{Dr} = cos\theta_r tan\theta_p$ is the same element as can be found in the corresponding element of matrix (2.19).

By breaking up the membership functions of the input as such:



The double integrals can be easily computed using a math software, with the limits defined by the triangular function shape. Equation (4.6) displays the procedure.

$$
\begin{aligned}
a_{15} &= \int_{-\infty}^{\infty} \rho_{\theta_p}(x,u,t) \underbrace{\int_{-0.65}^{-0.325} (9.47\theta_r + 6.15)(cos\theta_r tan\theta_p)(x,u,t)d\theta_r}_{F_1} d\theta_p \\
&\quad + \int_{-\infty}^{\infty} \rho_{\theta_p}(x,u,t) \underbrace{\int_{-0.325}^{0} (-9.47\theta_r)(cos\theta_r tan\theta_p)(x,u,t)d\theta_r}_{F_2} d\theta_p \\
&= \int_{0}^{0.325} (9.47\theta_p)(\underbrace{0.451 tan\theta_p}_{F_1} + \underbrace{(-3.024 tan\theta_p)}_{F_2})d\theta_p \\
&\quad + \int_{0.325}^{0.65} (-9.47\theta_p + 6.15)(\underbrace{0.451 tan\theta_p}_{F_1} + \underbrace{(-3.024 tan\theta_p)}_{F_2})d\theta_p \\
&= -0.285 + (-0.597) \\
&\underline{\underline{= -0.882}}
\end{aligned} \tag{4.6}
$$

The same calculation is done for every element of the state matrix A_{WL}.

4.1.2 Simulation

The weighted linearization is implemented in `tsk.m` code line 204 - 432, bringing about the results shown in Figure 4.4.

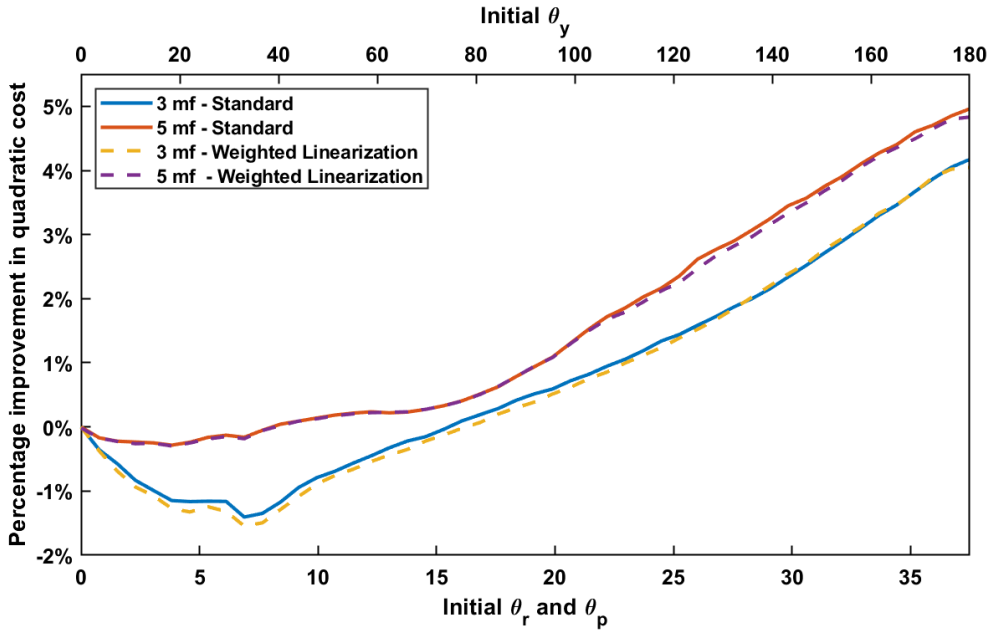


Fig 4.4: Percentage improvement in quadratic cost for the WL FLC over the linear controller in simulation. The initial condition $x(0)$ is increased for each simulation.

The weighted linearization with a membership function arrangement according to Figure 3.11 does not improve the results in simulation in any obvious way. The result indicates minor improvements in certain areas however, as can be seen in Figure 4.5. With some manipulation of the membership functions, better results may be possible.

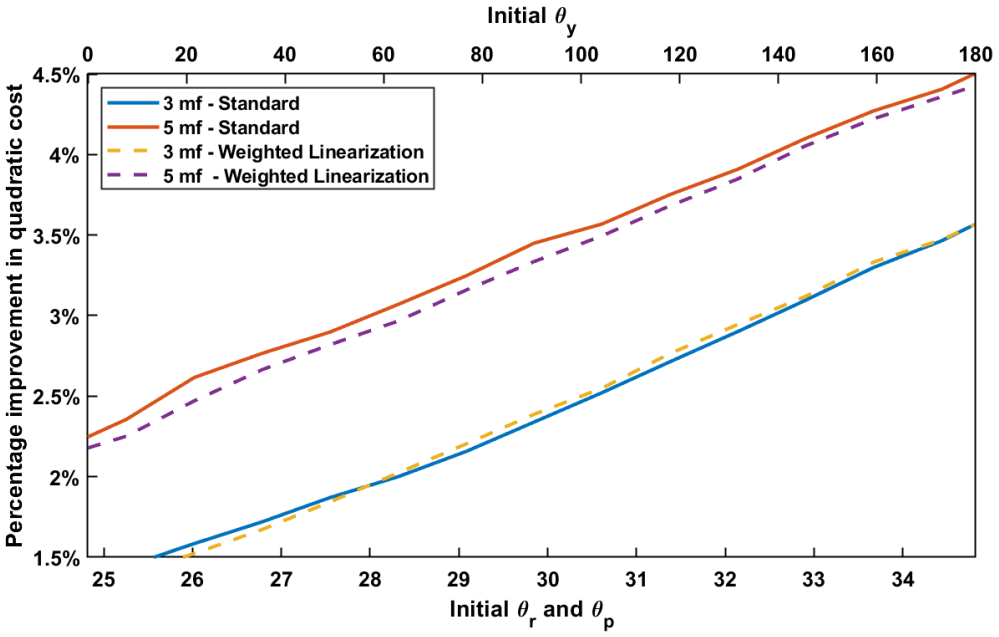


Fig 4.5: Percentage improvement in quadratic cost for the WL FLC over the linear controller in simulation. The initial condition $x(0)$ is increased for each simulation (zoom).

4.1.3 Real-time testing

The weighted linearization method is tested at the 3-DOF Hover bed in real-time. Compared to what we observed in the simulation section, where no notable $\pm$ difference could be observed, the weighted linearization acted similarly to the FLC also in real-time. The system was tested with 3, 5 and 9 membership functions according to Figure 3.11. The results are shown in Figure 4.6 and further analyzed.

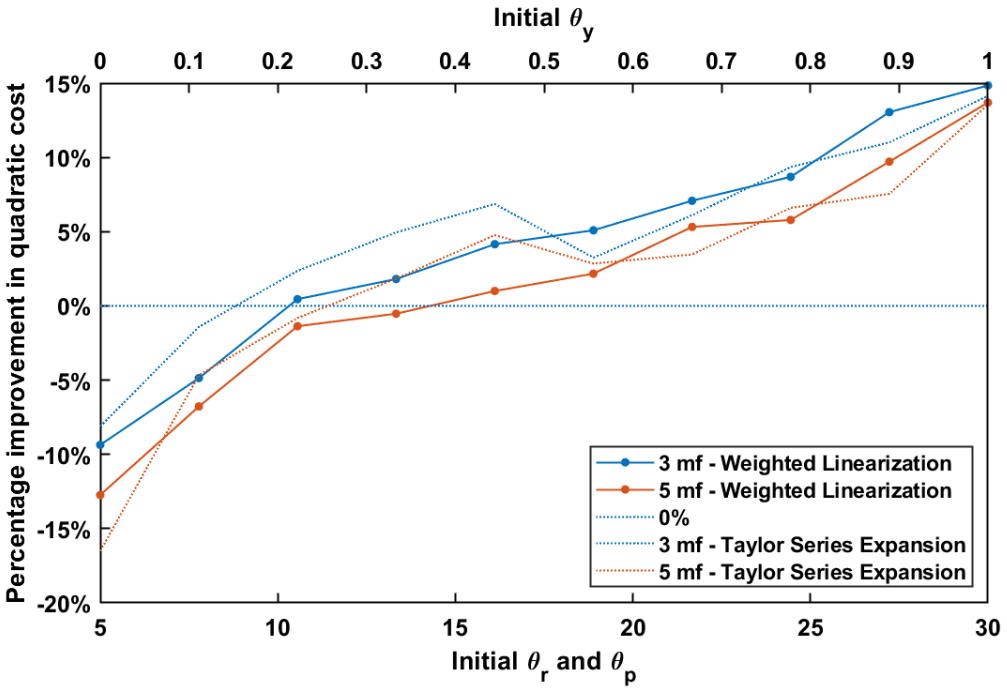


Fig 4.6: Percentage improvement in quadratic cost for the WL FLC over the linear controller in real-time. Each data point is an estimation from 4 identical experiments.

On average, the performance of the controller is subpar to a FLC using the taylor series expansion method of linearization. However, the FLC using a weighted linearization method has greater robustness, providing a steadier incline in performance. In combination with membership functions that better suit the nonlinear model, weighted linearization may prove a worthwhile addition to the controller scheme due to its ability to provide steadily improving results.

Chapter 5

Conclusions and Future work

5.1 Conclusions

5.1.1 Fuzzy Logic Controller

In the simulation comparison, the sun always shines because one does not have influencing factors compared to real-time. In this part, the fuzzy logic controller works better than the linear controller for all initial conditions angles bigger than zero, as shown in Figure 3.6. And as observed from the same figure, the percentage improvement over the linear controller increases as initial conditions are raised.

On the other hand, the real-time comparison turned out to reward the FLC less compared to the simulation. Especially for the more complex controllers with 5 or more membership functions. The FLC would still score well, especially when operating far from the equilibrium point. There are several reasons why the performance in real-time does not align with simulations however. The fact that it is a physical system for one, gives several influencing factors which are difficult to find, and therefore not included in the non-linear calculations.

A similar conclusion can be found in other papers, like in this paper written by students from Spain and Romania published as an Inproceeding[1] in the 2011 50th IEEE Conference on Decision and Control and European Control Conference [21].

[1] A paper that was published in the proceedings of a conference, and must have been accepted by the conference.

Their conclusion is mostly the same regarding simulation, however, they seemed to get a better experimental real-time result than we managed:

> *An LMI-based Takagi-Sugeno nonlinear observer has been designed for attitude and rotational speed estimation in a quadrotor. The experimental results presented show that a better estimation is obtained with the TS observer when the operating range is far away from the point of linearization of a similarly designed linear observer. In this way, the theoretical advantages of the TS framework are confirmed in a real experiment.*

As observed in Chapter 3.2.4 involving real-time testing, some questions about the membership functions occur. Both about how the operating points of the velocities act, and how different membership functions should be applied to further improve performance. This is mentioned as an aspect to improve in the future work Chapter 5.2, the system will surely improve more by researching and experimenting with the setup of the membership functions. Anyhow, a quite basic fuzzy logic controller with 3 membership functions does provide quite reliable improvements.

5.1.2 Weighted Linearization approach

The weighted linearization was quite an interesting approach. The baseline for this part of the report was Damiano Rotondo's *"Weighted linearization of nonlinear systems"* [20]. This really helped the understanding of what weighted linearization is generally about, and the benefits of it. Good examples of other papers using weighted linearization on a non-linear system were rare and hard to find. The combination of weighted linearization and triangular membership/weighting functions could not be found in any published paper as well. Surely more published papers using these combinations would help the understanding of the approach, and a better FLC based on weighted linearization could be evolved.

The simulation and real-time results both show varying degrees of improvement in quadratic cost. However, they seem to follow the same kind of pattern, lying close to the regular FLC cost-improvement compared to the linear controller. Results indicate that the approach could work better with weighted linearization. Figure 4.6 suggests a more robust rate of improvement when moving away from the equilibrium point $\bar{x} = 0$. However, the weighting-function relationship should be improved to reach a conclusion.

5.2 Future work

This project could be further improved by looking into the following aspects:

- Yet unknown physical factors acting on the 3DOF hover system could have been further investigated, to obtain an even more precise model. A continuation of the work done in Appendix A might be beneficial to further work with the rig.

  In particular, it was found to be a nonlinear relationship between the applied voltages and the propeller speed. More experiments involving open-loop testing could prove to find a more accurate description of the propeller angular frequency. For example as stated in [22], where the propeller angular frequency was interpreted with a first-order transfer function.

- The operating points of the velocities should be further researched and resolved why they act as they do. It was found that manipulating the velocity input membership functions would impact the performance more than the angular input membership functions.

- Evolve even more appropriate membership functions in the FLC, concerning the behavior of the 3DOF Hover. An arbitrary example is shown in Figure 5.1, where the membership functions are not necessarily symmetrical.

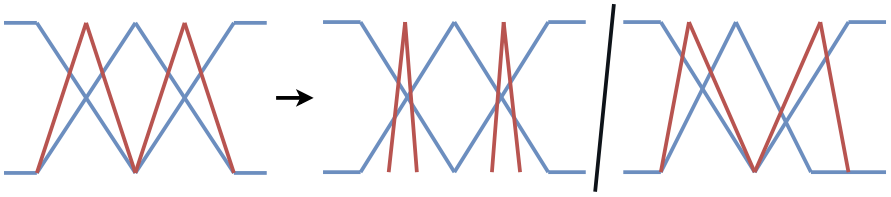


Fig 5.1: Example of how the membership function could be evolved.

  In particular, it would be interesting to find the operating points that would be most appropriately chosen as the center of the membership functions, especially with regards to the velocities as mentioned in the previous comment. Some methods might be found to choose the most optimal membership functions.

- A brief observation at the end of Section 2.4.2 mentions unobservable modes in the imaginary axis of Q and A. The connection between integral states and these unobservable modes could have been a topic of investigation.

Bibliography

[1] MATLAB. Mamdani and sugeno fuzzy inference systems. Link here.

[2] Quanser. *Quanser 3 DOF Hover User Manual*. Quanser, Markham, Ontario, Canada, 2013.

[3] Quanser. 3 dof hover product site. Link here.

[4] Y. Wang, J. Pan, B. Jiang, and N-Y. Lu. Hybrid model based fault tolerant control for quadrotor helicopter with structural damage. In *2016 Chinese Control and Decision Conference (CCDC)*, pages 5933–5938, 2016. doi: 10.1109/CCDC.2016.7532059.

[5] Wikipedia contributors. Łukasiewicz logic — Wikipedia, the free encyclopedia, 2021. Link here.

[6] E.H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. *Proceedings of the Institution of Electrical Engineers*, 121:1585–1588(3), December 1974. ISSN 0020-3270. Link here.

[7] L. P. Holmblad and J. J. Ostergaard. Control of a cement kiln by fuzzy logic techniques. *IFAC Proceedings Volumes*, 14:809–814, 1981. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)63582-1. Link here.

[8] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):116–132, 1985. doi: 10.1109/TSMC.1985.6313399.

[9] M. Sugeno and G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28(1):15–33, 1988. Cited By :1991.

[10] L. A. Zadeh. *Fuzzy Logic, Neural Networks, and Soft Computing*, pages 775–782. doi: 10.1142/9789814261302_0040. Link here.

[11] K. Tanaka and M. Sugeno. Stability analysis of fuzzy systems using lyapunov's direct method. *Proc.NAFIPS'90*, pages 133–136, 1990.

[12] *Pittman 923x series brush commuted DC motors*. Pittman, 343 Godshall Drive, Harleysville, PA 19438, 2013.

[13] F. Haugen. *Praktisk Reguleringsteknikk*. Fagbokforlaget, 2009. ISBN 978-82-519-1887-9.

[14] F. Haugen. *Dynamiske systemer. Modellering, analyse og simulering*. Fagbokforlaget, 2012. ISBN 978-82-519-2260-9.

[15] J.L. Pitarch and A. Sala. Multicriteria fuzzy-polynomial observer design for a 3dof nonlinear electromechanical platform. *Engineering Applications of Artificial Intelligence*, 30:96–106, 2014. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2013.11.006. Link here.

[16] A. Prach, E. Kayacan, and D. S. Bernstein. *An experimental evaluation of the forward propagating Riccati equation to nonlinear control of the Quanser 3 DOF Hover testbed*. 2016 American Control Conference (ACC). IEEE, 2016. ISBN 1-4673-8680-4.

[17] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[18] Ph.D. J. Apkarian and M.A.SC. M. Lévis. *Quanser 3 DOF Hover Laboratory Guide*. Quanser, Markham, Ontario, Canada, 2013.

[19] M. Sugeno. *Industrial Applications of Fuzzy Control*. Elsevier Science Inc., USA, 1985. ISBN 0444878297.

[20] D. Rotondo. Weighted linearization of nonlinear systems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 1–1, 2021. doi: 10.1109/TCSII.2021.3137611. Link here.

[21] Z. Lendek, A. Berna, J. Guzmán-Giménez, A. Sala, and P. García. Application of takagi-sugeno observers for state estimation in a quadrotor. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 7530–7535, 2011. doi: 10.1109/CDC.2011.6160439.

[22] M. Dhaybi and N. Daher. Real-time estimation of the inertia tensor elements of a quadcopter hover platform. pages 1347–1352, 07 2019. doi: 10.1109/AIM.2019.8868641.

Appendix A

Experimental results and model adjustment

This appendix includes, in chronological order, the experimental results and observations that led to the adjustment of the mathematical model describing the system. The various models will be brought up in succession with their additions highlighted in red.

Figure A.1 provides a plot of the percentage improvement over repeated experiments where the initial condition was increased incrementally. The plot showed abnormalities concerning the higher initial conditions, as well as a general variance in the results.

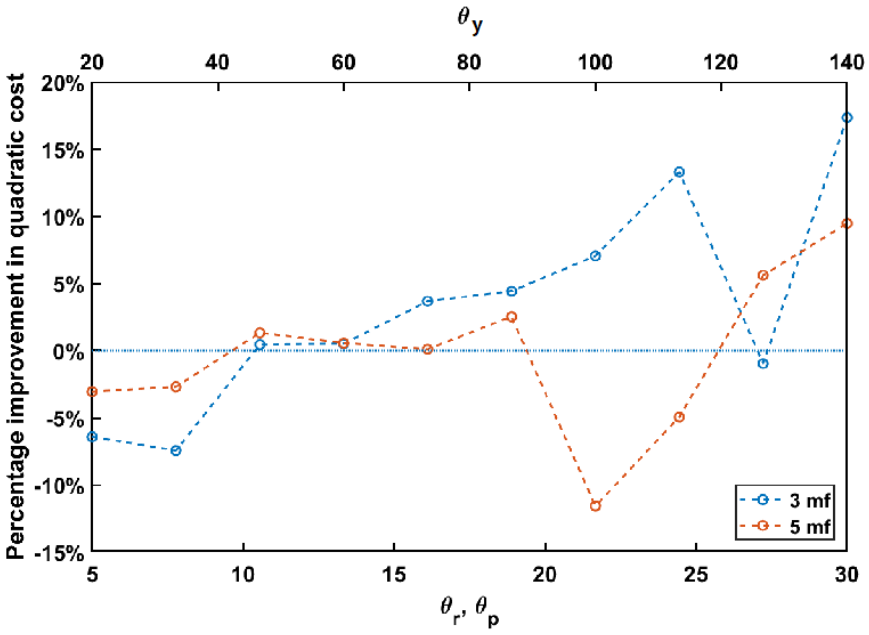


Fig A.1: Percentage improvement in cost for the FLC over the linear controller.

The variance was most obvious when comparing datasets from a range of experiments as can be seen in Figure A.2.

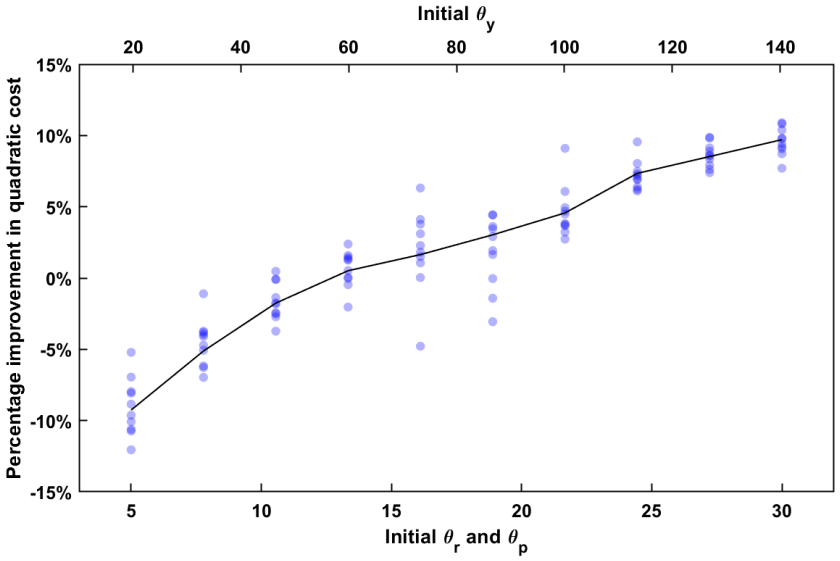


Fig A.2: Percentage improvement in cost for the FLC over the linear controller for 100 experiments.

As the Figures A.1 and A.2 show, there was a need to implement some procedures to reduce variance and abnormalities. Therefore the ramp reference and integral states was implemented, which in combination with an estimation over multiple experiments gave the resulting plot of Figure A.3.

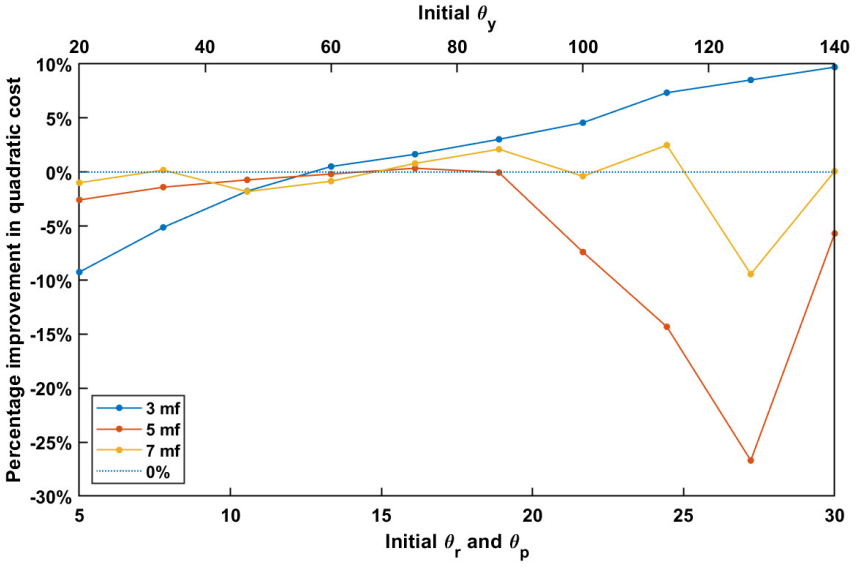


Fig A.3: Percentage improvement in cost for the FLC over the linear controller. Each data point is an estimation from 10 identical experiments.

Figure A.3 revealed a trend that was not obvious in Figure A.1. While it was evident that having 3 membership functions resulted in an improved controller, having 5 or 7 seemed to aggravate the performance significantly. The poor results raised the question about the accuracy of the nonlinear model (A.1) - (A.6).

MODEL VERSION 1

$$\dot{\theta_r} = f_1 = \theta_r + sin\theta_r tan\theta_p \theta_p + cos\theta_r tan\theta_p \theta_y \tag{A.1}$$

$$\dot{\theta_p} = f_2 = cos\theta_r \theta_p - sin\theta_r \theta_y \tag{A.2}$$

$$\dot{\theta_y} = f_3 = \frac{sin\theta_r}{cos\theta_p}\theta_p + \frac{cos\theta_r}{cos\theta_p}\theta_y \tag{A.3}$$

$$\ddot{\theta_r} = f_4 = \frac{J_p - J_y}{J_r}\dot{\theta_p}\dot{\theta_y} + \frac{1}{J_r}\tau_r \tag{A.4}$$

$$\ddot{\theta_p} = f_5 = \frac{J_y - J_r}{J_p}\dot{\theta_r}\dot{\theta_y} + \frac{1}{J_p}\tau_p \tag{A.5}$$

$$\ddot{\theta_y} = f_6 = \frac{J_r - J_p}{J_y}\dot{\theta_r}\dot{\theta_p} + \frac{1}{J_y}\tau_y \tag{A.6}$$

Figure A.4 shows discrepancies between the closed-loop response of a simulated model and the physical rig with the model (A.1) - (A.6).

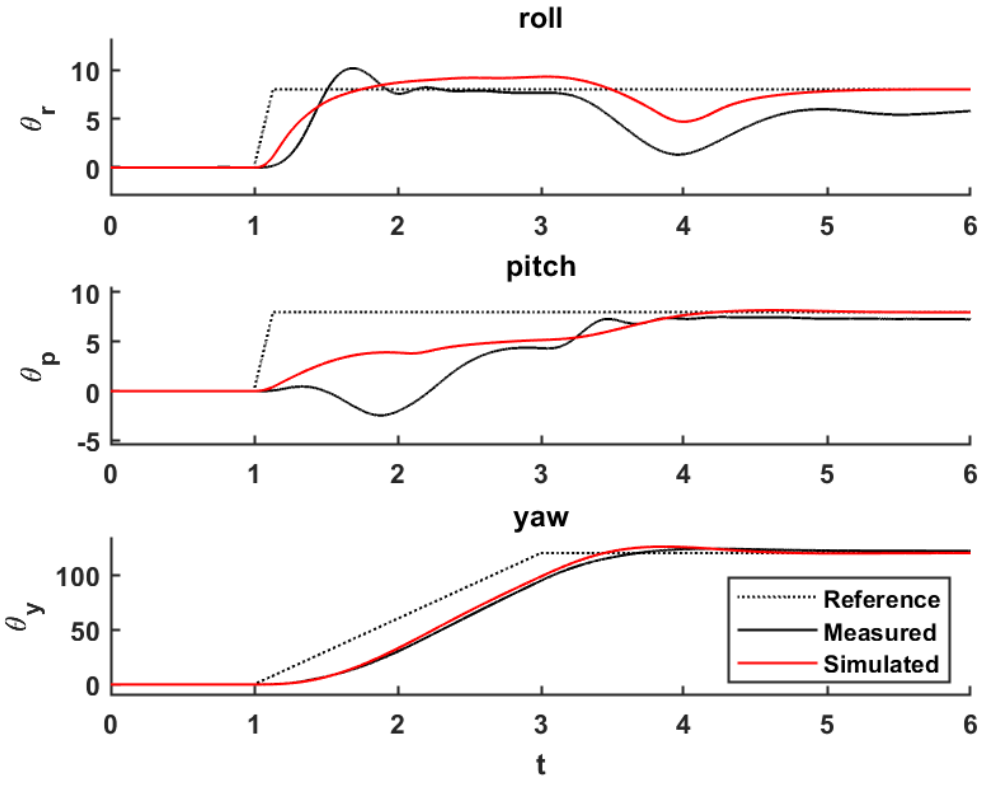


Fig A.4: Simulated vs. real closed-loop response to a reference change without a modelled gyroscopic effect.

By comparing simulations and real time experiments, factors that were not included in (A.1) - (A.6) were investigated. Table A.1 shows dynamics both already included, and ones that were not.

Physical effects	Source for the effect	Outcome variable	Included
Aerodynamics	Propeller rotation	Roll torque	✓
Aerodynamics	Blades flapping	Pitch torque	✓
Torque from inertia	Change in propeller rotation speed	Yaw torque	✓
Gyroscopic	Change in the orientation of the rigid body	Body gyroscopic torque	✓
Gyroscopic	Change in the orientation of the propeller plane	Rotor gyroscopic torque	X
Gravity	Center of mass position	Gravity	X
Near surface	The aircraft near the surface	Air velocity	X

Table A.1: Physical effects in the 3DOF Hover system [4].

By taking into account the omitted factors, firstly the rotor gyroscopic effects, the nonlinear Equations (A.1) - (A.6) evolved into the Equations (A.7) - (A.12) [4].

MODEL VERSION 2: Added gyroscopic effect

$$\dot{\theta_r} = f_1 = \theta_r + sin\theta_r tan\theta_p \theta_p + cos\theta_r tan\theta_p \theta_y \tag{A.7}$$

$$\dot{\theta_p} = f_2 = cos\theta_r \theta_p - sin\theta_r \theta_y \tag{A.8}$$

$$\dot{\theta_y} = f_3 = \frac{sin\theta_r}{cos\theta_p}\theta_p + \frac{cos\theta_r}{cos\theta_p}\theta_y \tag{A.9}$$

$$\ddot{\theta_r} = f_4 = \frac{J_p - J_y}{J_r}\dot{\theta_p}\dot{\theta_y} + \frac{1}{J_r}\tau_r + \frac{I_x\dot{\theta_p}}{J_r}u_g \tag{A.10}$$

$$\ddot{\theta_p} = f_5 = \frac{J_y - J_r}{J_p}\dot{\theta_r}\dot{\theta_y} + \frac{1}{J_p}\tau_p - \frac{I_x\dot{\theta_r}}{J_p}u_g \tag{A.11}$$

$$\ddot{\theta_y} = f_6 = \frac{J_r - J_p}{J_y}\dot{\theta_r}\dot{\theta_p} + \frac{1}{J_y}\tau_y \tag{A.12}$$

As Figure A.5 demonstrates, including the gyroscopic effect in the model (A.7) - (A.12) resulted in a greater resemblance between the simulated and real response.

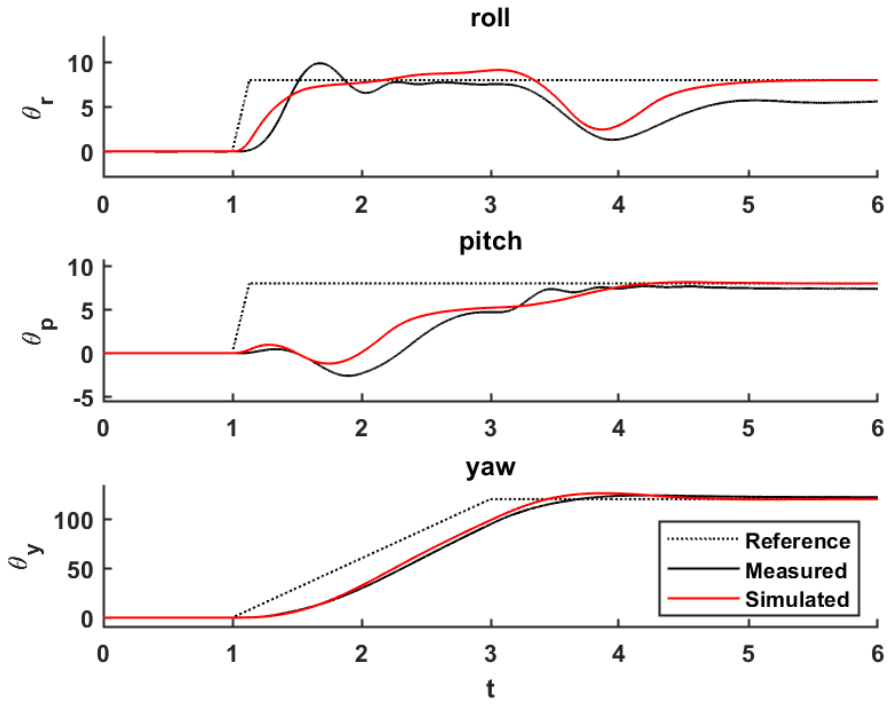


Fig A.5: Simulated vs. real closed-loop response to a reference change with a modelled gyroscopic effect.

There was an improvement in performance as presented in Figure A.6, however, the controllers with more than 3 membership functions continued to perform poorly when compared to the linear controller.

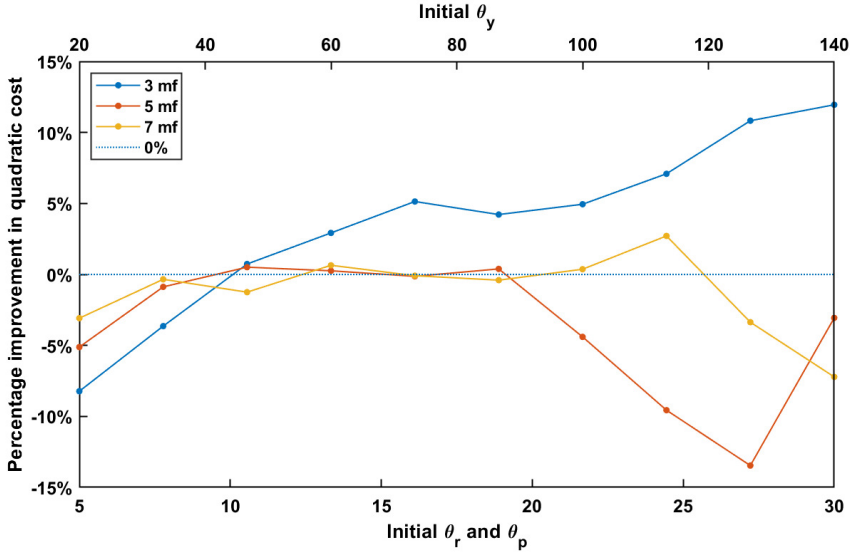


Fig A.6: Percentage improvement in cost for the FLC over the linear controller. Each data point is an estimation from 4 identical experiments.

Figure A.7 demonstrates how the standard deviation in results when repeating an experiment went down when the gyroscopic effect was modeled, making it a good addition.

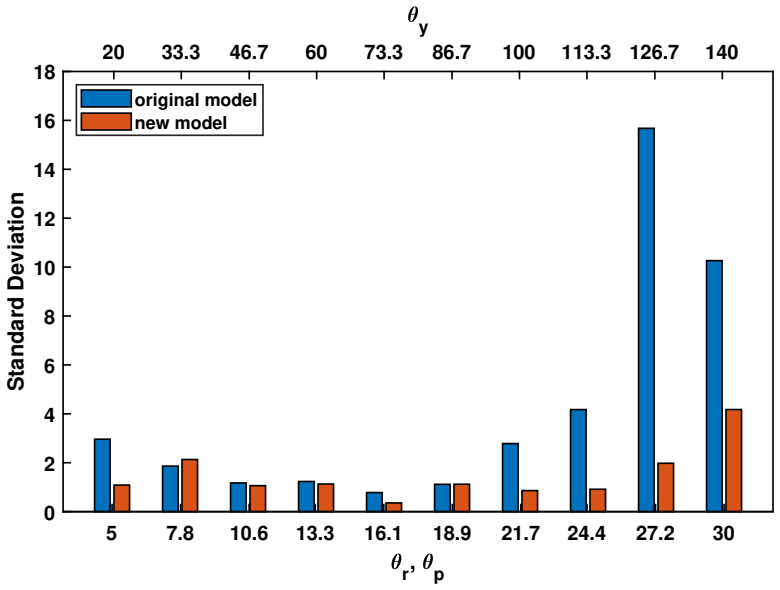


Fig A.7: Standard deviation in results for the 5 mf controller from Figure A.3 and A.6. X-axis displays initial condition.

Nonetheless, the performance of the fuzzy logic controller did still degrade quite rapidly with higher angles. The assumption that the angular velocities would be the derivative of the Euler angles was questioned, and it explained why higher initial conditions would be detrimental to the performance, as the difference between the Euler angular rates and angular velocities of the hover differ more as the angles increase. Figure A.8 shows discrepancies between Euler angular rate and angular velocity for a simple simulation.

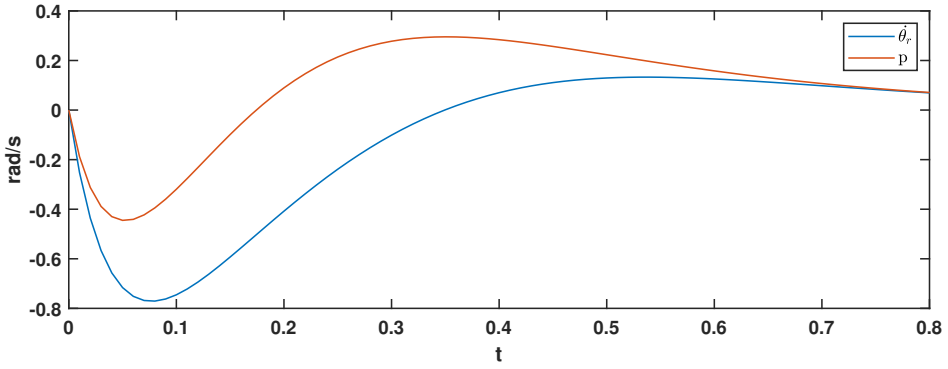


Fig A.8: Euler angular rate $\dot{\theta}_r$ and angular velocity p.

Thus the states $[\dot{\theta}_r, \dot{\theta}_p, \dot{\theta}_y]$ were replaced with the angular velocities of the hover $[p, q, r]$. Resulting in Equations (A.13) - (A.18) which are described in great detail in Section 2.3.1.

MODEL VERSION 3: Eular angular rates translated to angular velocities

$$\dot{\theta}_r = f_1 = p + sin\theta_r tan\theta_p q + cos\theta_r tan\theta_p r \tag{A.13}$$

$$\dot{\theta}_p = f_2 = cos\theta_r q - sin\theta_r r \tag{A.14}$$

$$\dot{\theta}_y = f_3 = \frac{sin\theta_r}{cos\theta_p} q + \frac{cos\theta_r}{cos\theta_p} r \tag{A.15}$$

$$\dot{p} = f_4 = \frac{J_p - J_y}{J_r} qr + \frac{1}{J_r}\tau_r + \frac{I_x q}{J_r} u_g \tag{A.16}$$

$$\dot{q} = f_5 = \frac{J_y - J_r}{J_p} pr + \frac{1}{J_p}\tau_p - \frac{I_x p}{J_p} u_g \tag{A.17}$$

$$\dot{r} = f_6 = \frac{J_r - J_p}{J_y} pq + \frac{1}{J_y}\tau_y \tag{A.18}$$

The remaining effects listed in Table A.1 which are not modeled in the 3DOF hover system will not be featured.

The gravity effect is not something to consider, assuming the hover is resting at its center of mass by the rig's pivot joint. The effect is however taken to account physically, as a weight clip shown in Figure A.9 is attached to the frame. This makes the hover balanced and in parallel with the floor level when at rest.

The air velocity the system would gain eventually being near the surface is also a non-considering factor here. The hover works at the same height, only adjusting the axes' angles.

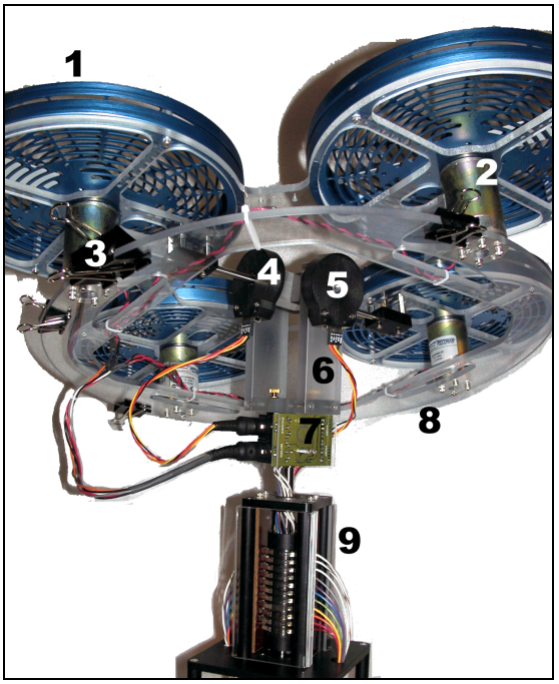


Fig A.9: Weighting clips attached to the 3-DOF hover, shown as component #3 [2].

Appendix B

Weighted Linearization based on a Gaussian weighting function

Multiple linearized state matrices A_σ are obtained by choosing different values for σ, a value defining the width of the Gaussian weighting function as seen in Figure B.1.

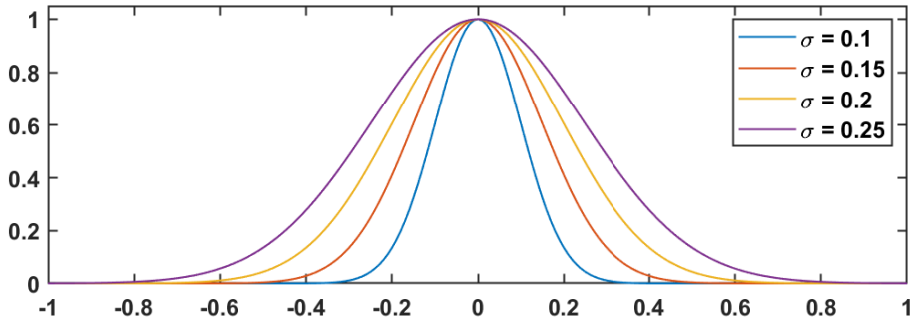


Fig B.1: Gaussian weighting functions with different deviation σ.

Looking at the state-space nonlinear equations (2.4) - (2.9), one would need to calculate the different values for A_σ by an equation which includes an integral for each of the applicable states from each element in the matrix $A(x)$ (2.19) obtained from taylor series expansion. This stems from the factor $\frac{Df}{Dx}$ of Equation (4.1).

Five of the states x (2.14) appear in the state matrix $A(x)$, which means the multidimensional Gaussian weighting function will look like in Equation (B.1).

$$\rho(\theta_r, \theta_p, p, q, r) = \frac{10^5}{2\pi^{5/2}} e^{-\left(\frac{(\theta_r - \bar{\theta_r})^2}{2\sigma^2} + \frac{(\theta_p - \bar{\theta_p})^2}{2\sigma^2} + \cdots + \cdots + \frac{(r-\bar{r})^2}{2\sigma^2}\right)} \tag{B.1}$$

This is calculated by the Equation (B.2), described in [20] as follows:

A class of weighting functions that showed promising results are multidimensional Gaussian functions centered on $(\bar{x}(t), \bar{u}(t))$ with unit hypervolume, which are described by the following expression:

$$\rho(x,u,t) = \sqrt{\frac{detH}{\pi^{n+m}}} \exp\left(- \begin{bmatrix} x-\bar{x}(t) \\ u-\bar{u}(t) \end{bmatrix}^T H \begin{bmatrix} x-\bar{x}(t) \\ u-\bar{u}(t) \end{bmatrix} \right) \tag{B.2}$$

where H is a positive definite (n+m) x (n+m) matrix.

When the system responds to a chosen deviation value of $\sigma = 0.1$, and $\bar{x} = 0$, the weighting function appears as in Equation (B.3).

$$\rho(\theta_r, \theta_p, p, q, r) = \frac{10^5}{(2\pi)^{5/2}} e^{-50(\theta_r^2+\theta_p^2+p^2+q^2+r^2)} \tag{B.3}$$

$A_{0.1}$ can then be calculated through Equation (B.4).

$$\frac{10^Q}{2\pi^{Q/2}} = \int \cdots \int_{R^Q} e^{-50(\theta_r^2+\theta_p^2+p^2+q^2+r^2)} (A_{xx})\ d\theta_r\ d\theta_p\ dp\ dq\ dr \tag{B.4}$$

Where Q corresponds to the number of states included in each element of $A(x)$. This fact means that we will never look at Q > 2, as the expressions from the original state matrix $A(x)$ (2.19) does not include more than 2 states. For example, the value A_{36} will look like:

$$\frac{10^2}{2\pi} \int \cdots \int_{R^2} e^{-50(\theta_r^2+\theta_p^2+p^2+q^2+r^2)} (A_{36})\ d\theta_r\ d\theta_p\ dp\ dq\ dr$$

$$\frac{10^2}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-50(\theta_r^2+\theta_p^2)} \left(\frac{\cos\theta_r}{\cos\theta_p}\right)\ d\theta_r\ d\theta_p = 1.000051676$$

Calculating for all the points in $A(x)$ and a weighting function with $\sigma = 0.1$, gives the following A matrix :

$$A_{0.1} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt[200]{e}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.000051676 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Table B.1 includes values with different σ, and the states present for the function set with an operating point $\bar{x} = 0$. The function here is $\frac{Df_3}{Dr}$, or element A_{36} of the state matrix linearized with taylor series expansion. It will be noted that the equation would not integrate with values bigger than $\sigma = 0.25$.

$\bar{\theta_r}$	$\bar{\theta_p}$	σ	$\sqrt{\frac{det(H)}{\pi^{n+m}}}$	f(x)	$e^{\frac{-(x-\bar{x})^2}{2\sigma^2}}$	Result
0	0	0.1	$\frac{10^2}{2\pi}$	$\frac{\cos\theta_r}{\cos\theta_p}$	$e^{-50(\theta_r^2+\theta_p^2)}$	1.000051676
0	0	0.15	$\frac{\frac{20}{3}^2}{2\pi}$	$\frac{\cos\theta_r}{\cos\theta_p}$	$e^{-(\frac{200}{9})(\theta_r^2+\theta_p^2)}$	1.00026406
0	0	0.2	$\frac{5^2}{2\pi}$	$\frac{\cos\theta_r}{\cos\theta_p}$	$e^{-12.5(\theta_r^2+\theta_p^2)}$	1.000873712
0	0	0.25	$\frac{4^2}{2\pi}$	$\frac{\cos\theta_r}{\cos\theta_p}$	$e^{-8(\theta_r^2+\theta_p^2)}$	1.002261065

Table B.1: The weighted linearization of f_3 with regards to the angular velocity r using a Gaussian weighting function with different values of σ.

For the function A_{25} where only the state θ_r appears, results can be seen in Table B.2.

$\bar{\theta_r}$	σ	$\sqrt{\frac{det(H)}{\pi^{n+m}}}$	f(x)	$e^{\frac{-(x-\bar{x})^2}{2\sigma^2}}$	Result
0	0.1	$\frac{10}{2\pi}$	$\cos\theta_r$	$e^{-50(\theta_r^2)}$	0.995012
0	0.15	$\frac{\frac{20}{3}}{2\pi}$	$\cos\theta_r$	$e^{-(\frac{200}{9})(\theta_r^2)}$	0.988813
0	0.2	$\frac{5}{2\pi}$	$\cos\theta_r$	$e^{-12.5(\theta_r^2)}$	0.980199
0	0.25	$\frac{4}{2\pi}$	$\cos\theta_r$	$e^{-8(\theta_r^2)}$	0.969233

Table B.2: The weighted linearization of f_2 with regards to the angular velocity q using a Gaussian weighting function with different values of σ.

Appendix C

MATLAB code

C.1 HoverSim.m

```
function xdot = HoverSim(x,jp,jr,jy,l,kf,kt,u)

rtau = l*kf*(u(3)-u(4));
ptau = l*kf*(u(1)-u(2));
ytau = kt*(u(3)+u(4))-kt*(u(1)+u(2));

xdot = zeros(6,1);
xdot(1,1) = x(4) + sin(x(1))*tan(x(2))*x(5)+cos(x(1))*tan(x(2))*x(6);
xdot(2,1) = cos(x(1))*x(5) - sin(x(1))*x(6);
xdot(3,1) = (cos(x(1))/cos(x(2)))*x(6) + (sin(x(1))/cos(x(2)))*x(5);
xdot(4,1) = (((jp-jy)/jr)*x(5)*x(6) + rtau/jr);
xdot(5,1) = ((jy-jr)/jp)*x(4)*x(6) + ptau/jp;
xdot(6,1) = (((jr-jp)/jy)*x(4)*x(5) + ytau/jy);
```

C.2 HoverSimIntegralStates.m

```
function xdot = HoverSimIntegralStates(x,ref,jp,jr,jy,l,kf,kt,u)

rtau = l*kf*(u(3)-u(4));
ptau = l*kf*(u(1)-u(2));
ytau = kt*(u(3)+u(4))-kt*(u(1)+u(2));

xdot = zeros(6,1);
xdot(1,1) = x(4) + sin(x(1))*tan(x(2))*x(5)+cos(x(1))*tan(x(2))*x(6);
xdot(2,1) = cos(x(1))*x(5) - sin(x(1))*x(6);
xdot(3,1) = (cos(x(1))/cos(x(2)))*x(6) + (sin(x(1))/cos(x(2)))*x(5);
xdot(4,1) = ((jp-jy)/jr)*x(5)*x(6) + rtau/jr;
```

```
xdot(5,1) = ((jy-jr)/jp)*x(4)*x(6) + ptau/jp + 1/jp;
xdot(6,1) = ((jr-jp)/jy)*x(4)*x(5) + ytau/jy;
xdot(7,1) = x(1)-ref(1);
xdot(8,1) = x(2)-ref(2);
xdot(9,1) = x(3)-ref(3);
```

C.3 HoverConstants.m

```
function [Jp,Jy,Jr,L,Kf,Kt,Ir,Kv] = HoverConstants()

Jp = 0.0552;
Jy = 0.110;
Jr = 0.0552;
L = 0.197;
Kf = 0.1188;
Kt = 0.0036;
Ir = 6e-5;
Kv = 54.945;
```

C.4 SimulationLQR.m

```
%% Defining the parameters

[Jp,Jy,Jr,L,Kf,Kt,Ir,Kv] = HoverConstants();

v_bias = [2 2 2 2];

Q = diag([350 350 500 20 20 0]);
R = diag([0.01 0.01 0.01 0.01]);
K = GainMatrix(0,0,0,0,0,0);

%% Simulate closed-loop system

x0(1,1) = deg2rad(30);
x0(2,1) = deg2rad(30);
x0(3,1) = deg2rad(140);
x0(4,1) = 0;
x0(5,1) = 0;
x0(6,1) = 0;

ref(1,1) = deg2rad(0);
ref(2,1) = deg2rad(0);
ref(3,1) = deg2rad(0);
```

```
ref(4,1) = 0;
ref(5,1) = 0;
ref(6,1) = 0;

tspan = 0:0.01:2;

u=@(x)(-K*(x - ref))' + v_bias;

[t,x] = ode45(@(t,x)HoverSim(x,Jp,Jr,Jy,L,Kf,Kt,u(x)),tspan,x0);


%% Finding the quadratic cost

quad = zeros(length(x)-1,1);

for k = 1:(length(x)-1)
    quad(k) = x(k,:)*Q*x(k,:)' + u(x(k,:)')*R*u(x(k,:)')';
end

J_int = x(end,:)*Q*x(end,:)' + sum(quad);

fprintf('Linearized System gives quadratic cost: %s\n',J_int)
disp(' ')

%% plot

x = arrayfun(@(x)rad2deg(x),x(:,1:3)); % \\ in degrees

% figure(2)

subplot(3,1,1);hold on
plot(t,x(:,1),'Color','red')
% line([0 tspan(end)],[rad2deg(ref(1,1)) rad2deg(ref(1,1))],'LineStyle
    ',':','Color','red')
ylabel('\theta_{r}','Interpreter','tex')
title('roll','FontWeight','normal')


subplot(3,1,2);hold on
plot(t,x(:,2),'Color','red')
% line([0 tspan(end)],[rad2deg(ref(2,1)) rad2deg(ref(2,1))],'LineStyle
    ',':','Color','red')
ylabel('\theta_{p}','Interpreter','tex')
title('pitch','FontWeight','normal')
hold on

subplot(3,1,3);hold on
plot(t,x(:,3),'Color','red')
```

```
% line([0 tspan(end)],[rad2deg(ref(3,1)) rad2deg(ref(3,1))],'LineStyle
    ',':','Color','red')
ylabel('\theta_{y}','Interpreter','tex')
xlabel('t')
title('yaw','FontWeight','normal')
hold on
```

C.5 SimulationFIS.m

```
%% Defining the parameters

[Jp,Jy,Jr,L,Kf,Kt,Ir,Kv] = HoverConstants();

v_bias = [2 2 2 2];

Q = diag([350 350 500 20 20 0]);
R = diag([0.01 0.01 0.01 0.01]);

FIS = tsk(5);

%% Simulate closed-loop system

x0(1,1) = deg2rad(30);
x0(2,1) = deg2rad(30);
x0(3,1) = deg2rad(140);
x0(4,1) = 0;
x0(5,1) = 0;
x0(6,1) = 0;

ref(1,1) = deg2rad(0);
ref(2,1) = deg2rad(0);
ref(3,1) = deg2rad(0);
ref(4,1) = 0;
ref(5,1) = 0;
ref(6,1) = 0;

tspan = 0:0.01:2;

u = @(x) -EvalFuzzy(FIS,x-ref)' + v_bias;

[t,x] = ode45(@(t,x)HoverSim(x,Jp,Jr,Jy,L,Kf,Kt,u(x)),tspan,x0);


%% Finding the quadratic cost

quad = zeros(length(x)-1,1);
```

```
38
39 for k = 1:(length(x)-1)
40     quad(k) = x(k,:)*Q*x(k,:)' + u(x(k,:)')*R*u(x(k,:)')';
41 end
42
43 J_fis = x(end,:)*Q*x(end,:)' + sum(quad);
44
45 fprintf('Fuzzy Inference System gives quadratic cost: %s\n',J_fis)
46 disp(' ')
47
48 %% plot
49
50 x = arrayfun(@(x)rad2deg(x),x(:,1:3));
51
52 figure('Name','FIS simulation','NumberTitle','off');
53
54 subplot(3,1,1)
55 plot(t,x(:,1))
56 line([0 tspan(end)],[rad2deg(ref(1,1)) rad2deg(ref(1,1))],'LineStyle',':'
    )
57 ylabel('\theta_{r}')
58 title('roll')
59 set(gca,'box','off');
60
61 subplot(3,1,2)
62 plot(t,x(:,2))
63 line([0 tspan(end)],[rad2deg(ref(2,1)) rad2deg(ref(2,1))],'LineStyle',':'
    )
64 ylabel('\theta_{p}')
65 title('pitch')
66 set(gca,'box','off');
67
68 subplot(3,1,3)
69 plot(t,x(:,3))
70 line([0 tspan(end)],[rad2deg(ref(3,1)) rad2deg(ref(3,1))],'LineStyle',':'
    )
71 ylabel('\theta_{y}')
72 xlabel('t')
73 title('yaw')
74 set(gca,'box','off');
```

C.6 GainMatrix.m

```
1 function K = GainMatrix(roll, pitch, ~, p, q, r)
2
3 [Jp,Jy,Jr,L,Kf,Kt,Ir,Kv] = HoverConstants();
```

```
4
5  A = [...
6      0 0 0 1 sin(roll)*tan(pitch) cos(roll)*tan(pitch);...
7      0 0 0 0 cos(roll) -sin(roll);...
8      0 0 0 0 sin(roll)/cos(pitch) cos(roll)/cos(pitch);...
9      0 0 0 0 (Jp-Jy)*r/Jr (Jp-Jy)*q/Jr;...
10     0 0 0 (Jy-Jr)*r/Jp 0 (Jy-Jr)*p/Jp;...
11     0 0 0 (Jr-Jp)*q/Jy (Jr-Jp)*p/Jy 0];
12
13 B = [...
14     0 0 0 0;...
15     0 0 0 0;...
16     0 0 0 0;...
17     -Ir*q*Kv/Jr -Ir*q*Kv/Jr L*Kf/Jr+Ir*q*Kv/Jr -L*Kf/Jr+Ir*q*Kv/Jr;...
18     L*Kf/Jp+Ir*p*Kv/Jr -L*Kf/Jp+Ir*p*Kv/Jr -Ir*p*Kv/Jr -Ir*p*Kv/Jr;...
19     -Kt/Jy -Kt/Jy Kt/Jy Kt/Jy];
20
21 Q = diag([350 350 500 20 20 0]);
22 R = diag([0.01 0.01 0.01 0.01]);
23
24 K = lqr(A, B, Q, R);
```

C.7 GainMatrixInt.m

```
1  function K = GainMatrixInt(roll, pitch, ~, p, q, r, qrint, qpint, qyint)
2
3  [Jp,Jy,Jr,L,Kf,Kt,Ir,Kv] = HoverConstants();
4
5  A = [...
6      0 0 0 1 sin(roll)*tan(pitch) cos(roll)*tan(pitch);...
7      0 0 0 0 cos(roll) -sin(roll);...
8      0 0 0 0 sin(roll)/cos(pitch) cos(roll)/cos(pitch);...
9      0 0 0 0 (Jp-Jy)*r/Jr (Jp-Jy)*q/Jr;...
10     0 0 0 (Jy-Jr)*r/Jp 0 (Jy-Jr)*p/Jp;...
11     0 0 0 (Jr-Jp)*q/Jy (Jr-Jp)*p/Jy 0];
12
13 B = [...
14     0 0 0 0;...
15     0 0 0 0;...
16     0 0 0 0;...
17     -Ir*q*Kv/Jr -Ir*q*Kv/Jr L*Kf/Jr+Ir*q*Kv/Jr -L*Kf/Jr+Ir*q*Kv/Jr;...
18     L*Kf/Jp+Ir*p*Kv/Jr -L*Kf/Jp+Ir*p*Kv/Jr -Ir*p*Kv/Jr -Ir*p*Kv/Jr;...
19     -Kt/Jy -Kt/Jy Kt/Jy Kt/Jy];
20
21 Q = diag([350 350 500 20 20 0]);
22 R = diag([0.01 0.01 0.01 0.01]);
```

```
23
24 if qrint ~= 0
25     A = [A,zeros(length(A),1);1,zeros(1,length(A))];
26     B = [B;0,0,0,0];
27     Q = [Q,zeros(length(Q),1);zeros(1,length(Q)),qrint];
28 end
29 if qpint ~= 0
30     A = [A,zeros(length(A),1);0,1,zeros(1,length(A)-1)];
31     B = [B;0,0,0,0];
32     Q = [Q,zeros(length(Q),1);zeros(1,length(Q)),qpint];
33 end
34 if qyint ~= 0
35     A = [A,zeros(length(A),1);0,0,1,zeros(1,length(A)-2)];
36     B = [B;0,0,0,0];
37     Q = [Q,zeros(length(Q),1);zeros(1,length(Q)),qyint];
38 end
39
40 K = lqr(A, B, Q, R);
```

C.8 PermRep.m

```
1  function y = PermRep(n,k)
2
3  %PERMREP calculates all the combinatoric permutations with repetition.
4  %   A vector of length n^k is created consisting of all combinations of n
5  %   given k positions. n may be a vector of arbitrary numbers.
6  %
7  %   Example:
8  %       PermRep(2,3)
9  %
10 %          1     1     1
11 %          2     1     1
12 %          1     2     1
13 %          2     2     1
14 %          1     1     2
15 %          2     1     2
16 %          1     2     2
17 %          2     2     2
18
19 x = n;
20 K = k;
21
22 %// create all possible permutations (with repetition).
23 %// Code from www.stackoverflow.com
24
25 C = cell(K, 1);             %// Preallocate a cell array
```

```
[C{:}] = ndgrid(x);          %// Create K grids of values
y = cellfun(@(x){x(:)}, C); %// Convert grids to column vectors
y = [y{:}];                  %// Obtain all permutations
```

C.9 RuleMaker.m

```
function [no_points, eq_list, rule_list] = RuleMaker(num_mf, limit)

%% declaring variables needed for the logic deciding the equilibrium
    point

pos = linspace(-limit,limit,num_mf); % a vector where each fuzzy set
    represents a position in the range of angles

vel = linspace(-1,1,num_mf);          % a vector where each fuzzy set
    represents a velocity in the range

%% filling the equilibrium and rule list

eq_list = zeros(num_mf^5,5);
rule_list = zeros(num_mf^5,12);

m = 1;

for h = 1:num_mf
for i = 1:num_mf
for j = 1:num_mf
for k = 1:num_mf
for l = 1:num_mf
    eq_list(m,:) = [pos(h),pos(i),vel(j),vel(k),vel(l)];
    rule_list(m,:) = [h,i,0,j,k,l,[1,1,1,1]*m,1,1];
    m = m+1;
end
end
end
end
end

no_points = length(eq_list);
```

C.10 SugenoFIS.m

```
%% declaring constants

roll_range = deg2rad(37.5);
pitch_range = deg2rad(37.5);
yaw_range = deg2rad(180); %spins freely 360 degrees
velocity_range = 1;

%% creating the sugeno fuzzy inference system object

FIS = sugfis('Name','hover');

%% producing the gain matrices

% num_mf = 3;
limit = roll_range;

[no_points,eq_list,rule_list] = RuleMaker(num_mf,limit);

K = cell(no_points,1);

for i = 1:no_points
    K{i} = GainMatrix(eq_list(i,1),eq_list(i,2),0,eq_list(i,3),eq_list(i
    ,4),eq_list(i,5));
end

%% adding inputs

FIS = addInput(FIS,[-deg2rad(37.5) deg2rad(37.5)],'Name',"roll");
steps = (-roll_range:roll_range*2/(num_mf-1):roll_range);
for i = 1:num_mf
    if i == 1
        FIS = addMF(FIS,"roll","trapmf",[-deg2rad(37.5) -deg2rad(37.5)
    steps(1) steps(2)]);
    elseif i == num_mf
        FIS = addMF(FIS,"roll","trapmf",[steps(end-1) steps(end) deg2rad
    (37.5) deg2rad(37.5)]);
    else
        FIS = addMF(FIS,"roll","trimf",[steps(i-1) steps(i) steps(i+1)]);
    end
end

FIS = addInput(FIS,[-deg2rad(37.5) deg2rad(37.5)],'Name',"pitch");
steps = (-pitch_range:pitch_range*2/(num_mf-1):pitch_range);
for i = 1:num_mf
    if i == 1
        FIS = addMF(FIS,"pitch","trapmf",[-deg2rad(37.5) -deg2rad(37.5)
    steps(1) steps(2)]);
```

```
    elseif i == num_mf
        FIS = addMF(FIS,"pitch","trapmf",[steps(end-1) steps(end) deg2rad(37.5) deg2rad(37.5)]);
    else
        FIS = addMF(FIS,"pitch","trimf",[steps(i-1) steps(i) steps(i+1)]);
    end
end

FIS = addInput(FIS,[-deg2rad(180) deg2rad(180)],'Name',"yaw");
steps = (-yaw_range:yaw_range*2/(num_mf-1):yaw_range);
for i = 1:num_mf
    if i == 1
        FIS = addMF(FIS,"yaw","trapmf",[-deg2rad(180) -deg2rad(180) steps(1) steps(2)]);
    elseif i == num_mf
        FIS = addMF(FIS,"yaw","trapmf",[steps(end-1) steps(end) deg2rad(180) deg2rad(180)]);
    else
        FIS = addMF(FIS,"yaw","trimf",[steps(i-1) steps(i) steps(i+1)]);
    end
end

FIS = addInput(FIS,[-2 2],'Name',"roll_velocity");
steps = (-velocity_range:velocity_range*2/(num_mf-1):velocity_range);
for i = 1:num_mf
    if i == 1
        FIS = addMF(FIS,"roll_velocity","trapmf",[-2 -2 steps(1) steps(2)]);
    elseif i == num_mf
        FIS = addMF(FIS,"roll_velocity","trapmf",[steps(end-1) steps(end) 2 2]);
    else
        FIS = addMF(FIS,"roll_velocity","trimf",[steps(i-1) steps(i) steps(i+1)]);
    end
end

FIS = addInput(FIS,[-2 2],'Name',"pitch_velocity");
steps = (-velocity_range:velocity_range*2/(num_mf-1):velocity_range);
for i = 1:num_mf
    if i == 1
        FIS = addMF(FIS,"pitch_velocity","trapmf",[-2 -2 steps(1) steps(2)]);
    elseif i == num_mf
        FIS = addMF(FIS,"pitch_velocity","trapmf",[steps(end-1) steps(end) 2 2]);
    else
```

```
        FIS = addMF(FIS,"pitch_velocity","trimf",[steps(i-1) steps(i) steps(i+1)]);
    end
end

FIS = addInput(FIS,[-2 2],'Name',"yaw_velocity");
steps = (-velocity_range:velocity_range*2/(num_mf-1):velocity_range);
for i = 1:num_mf
    if i == 1
        FIS = addMF(FIS,"yaw_velocity","trapmf",[-2 -2 steps(1) steps(2)]);
    elseif i == num_mf
        FIS = addMF(FIS,"yaw_velocity","trapmf",[steps(end-1) steps(end) 2 2]);
    else
        FIS = addMF(FIS,"yaw_velocity","trimf",[steps(i-1) steps(i) steps(i+1)]);
    end
end

%% adding outputs

FIS = addOutput(FIS,'Name','vf');
for i = 1:no_points
    FIS = addMF(FIS,'vf','linear',[K{i}(1,:),0]);
end

FIS = addOutput(FIS,'Name','vb');
for i = 1:no_points
    FIS = addMF(FIS,'vb','linear',[K{i}(2,:),0]);
end

FIS = addOutput(FIS,'Name','vr');
for i = 1:no_points
    FIS = addMF(FIS,'vr','linear',[K{i}(3,:),0]);
end

FIS = addOutput(FIS,'Name','vl');
for i = 1:no_points
    FIS = addMF(FIS,'vl','linear',[K{i}(4,:),0]);
end

%% adding rules

FIS = addRule(FIS, rule_list);
```

C.11 tsk.m

```
classdef tsk

    %TSK Creates a Takagi-Sugeno-Kang Fuzzy Inference system for the 'Quanser 3DOF Hover'.
    %   FIS = (NumMF,NAME1,VALUE1,...) Creates a tsk fuzzy inference
    %   system with specified number of membership functions (default = 3).
    %   Additionally, using Name/Value pairs the following properties can be
    %   specified:
    %
    %       AngOuter        - Input variable range for the angles.
    %
    %       AngInner        - Membership function variable range for the
    %                         angles.
    %
    %       VelOuter        - Input variable range for the velocities.
    %
    %       Method          - Linearization method. 'TSE' (standard) for
    %                         taylor series expansion, 'WL' for weighted linearization.
    %
    %       Arrangement     - Membership function arrangement. 'Normal' (standard)
    %                         for linearly spaced and common width. 'Additive' for
    %                         membership functions added onto previous iteration.


    properties  (Hidden)
        AngleIndex              % Indexing vector for angles.
        VelocityIndex           % Indexing vector for velocities.
        AngOuter                % Input variable range for the angles.
        AngInner                % Membership function variable range for the angles.
        VelOuter                % Input variable range for the velocities.
        VelInner                % Membership function variable range for the velocities.
    end

    properties  (Hidden, Constant)
        NumInputs     = 5       % Number of inputs.
```

```
35          StepIncrement = 1/90    % Step increment size for the membership
    functions
36                                  % which defines accuracy for EvalFuzzy.m
37      end
38
39      properties
40          NumMF                   % Number of membership functions.
41          MemFunctions            % Cell containing membership functions.
42          RuleList                % Array containing all rules.
43          MatrixList              % Array containing all gain matrices.
44      end
45
46      methods
47
48          function obj = tsk(NumMF, Options)
49          % Creates the tsk object
50
51              arguments
52                  NumMF                       double = 3
53                  Options.AngOuter    (1,1)   double = deg2rad(37.5)
54                  Options.AngInner    (1,1)   double = deg2rad(37.5)
55                  Options.VelOuter    (1,1)   double = 2
56                  Options.VelInner    (1,1)   double = 1
57                  Options.Method              char   = 'TSE'
58                  Options.Arrangement         char   = 'Normal'
59              end
60
61              if strcmp(Options.Arrangement,'Additive') && ~ismember(NumMF
    ,[3,5,9])
62                  error('For additive arrangement, choose 3, 5 or 9
    membership functions')
63              elseif ~(strcmp(Options.Arrangement,'Normal') || strcmp(
    Options.Arrangement,'Additive'))
64                  error('Void arrangement (case sensitive)')
65              elseif ~(strcmp(Options.Method,'TSE') || strcmp(Options.
    Method,'WL'))
66                  error('Void linearization method (case sensitive)')
67              end
68
69              obj.AngOuter        = Options.AngOuter;
70              obj.AngInner        = Options.AngInner;
71              obj.VelOuter        = Options.VelOuter;
72              obj.VelInner        = Options.VelInner;
73              obj.NumMF           = NumMF;
74
75              obj.MemFunctions    = cell(obj.NumInputs,1);
76
```

```
77          obj.AngleIndex      = (-Options.AngOuter:Options.AngOuter*obj
    .StepIncrement:Options.AngOuter)';
78          obj.VelocityIndex   = (-Options.VelOuter:Options.VelOuter*obj
    .StepIncrement:Options.VelOuter)';
79
80          obj.MemFunctions{1} = createMF(obj,'angle',Options.
    Arrangement);
81          obj.MemFunctions{2} = createMF(obj,'angle',Options.
    Arrangement);
82          obj.MemFunctions{3} = createMF(obj,'velocity',Options.
    Arrangement);
83          obj.MemFunctions{4} = createMF(obj,'velocity',Options.
    Arrangement);
84          obj.MemFunctions{5} = createMF(obj,'velocity',Options.
    Arrangement);
85
86          [obj.MatrixList,obj.RuleList] = createRules(obj,Options.
    Method);
87
88      end
89
90      function [MFcell,WFcell] = createMF(obj, type, arr)
91      % Creates a membership function depending on type (angle or
    velocity)
92      % and arrangement type (normal or additive).
93
94          if strcmp(type,'velocity')
95              lim = obj.VelOuter;
96              ran = obj.VelInner;
97              x = obj.VelocityIndex;
98          elseif strcmp(type,'angle')
99              lim = obj.AngOuter;
100             ran = obj.AngInner;
101             x = obj.AngleIndex;
102         end
103
104         MFcell = cell(obj.NumMF,1);
105         WFcell = cell(obj.NumMF,1);
106
107         if strcmp(arr,'Normal')
108             steps = linspace(-ran,ran,obj.NumMF);
109             for i = 1:obj.NumMF
110                 if i == 1
111                     MFcell{i} = trapmf(x, [-lim -lim steps(1) steps
    (2)]);
112                 elseif i == obj.NumMF
113                     MFcell{i} = trapmf(x, [steps(end-1) steps(end)
    lim lim]);
```

```
114                     else
115                         MFcell{i} = trimf(x, [steps(i-1) steps(i) steps(i
    +1)]);
116                     end
117                 end
118             end
119 
120             if strcmp(arr,'Additive')
121                 for h = 1:(obj.NumMF-1)/2
122                     if h == 1
123                         steps = linspace(-ran,ran,3);
124                         for i = 1:3
125                             if i == 1
126                                 MFcell{1} = trapmf(x, ...
127                                     [-lim -lim steps(1) steps(2)]);
128                             elseif i == 3
129                                 MFcell{obj.NumMF} = trapmf(x, ...
130                                     [steps(end-1) steps(end) lim lim]);
131                             else
132                                 MFcell{(obj.NumMF+1)/2} = trimf(x, ...
133                                     [steps(i-1) steps(i) steps(i+1)]);
134                             end
135                         end
136                     elseif h == 2
137                         steps = linspace(-ran,ran,4*h-3);
138                         if obj.NumMF == 5
139                             MFcell{2} = trimf(x, [steps(1) steps(2) steps
    (3)]);
140                             MFcell{4} = trimf(x, [steps(3) steps(4) steps
    (5)]);
141                         elseif obj.NumMF == 9
142                             MFcell{3} = trimf(x, [steps(1) steps(2) steps
    (3)]);
143                             MFcell{7} = trimf(x, [steps(3) steps(4) steps
    (5)]);
144                         end
145                     elseif h == 3
146                         steps = linspace(-ran,ran,4*h-3);
147                         MFcell{2} = trimf(x, [steps(1) steps(2) steps(3)
    ]);
148                         MFcell{4} = trimf(x, [steps(3) steps(4) steps(5)
    ]);
149                         MFcell{6} = trimf(x, [steps(5) steps(6) steps(7)
    ]);
150                         MFcell{8} = trimf(x, [steps(7) steps(8) steps(9)
    ]);
151                     end
152                 end
```

```
153             end
154         end
155
156         function [matrices, rule_list] = createRules(obj,Method)
157         % Creates the rule list and cell of gain matrices
158
159             if strcmp(Method,'TSE')
160                 pos = linspace(-obj.AngInner,obj.AngInner,obj.NumMF);
161                 vel = linspace(-obj.VelInner,obj.VelInner,obj.NumMF);
162
163                 rule_list = zeros(obj.NumMF^obj.NumInputs,obj.NumInputs);
164                 matrices = cell(obj.NumMF^obj.NumInputs,1);
165
166                 m = 1;
167
168                 for h = 1:obj.NumMF
169                 for i = 1:obj.NumMF
170                 for j = 1:obj.NumMF
171                 for k = 1:obj.NumMF
172                 for l = 1:obj.NumMF
173                     matrices{m} = GainMatrix(pos(h),pos(i),0,vel(j),vel(k
    ),vel(l));
174                     rule_list(m,:) = [h,i,j,k,l];
175                     m = m+1;
176                 end
177                 end
178                 end
179                 end
180                 end
181
182             elseif strcmp(Method,'WL')
183
184                 pos = linspace(-obj.AngInner*(1+2/(obj.NumMF-1)), ...
185                     obj.AngInner*(1+2/(obj.NumMF-1)), obj.NumMF+2);
186
187                 vel = linspace(-obj.VelInner*(1+2/(obj.NumMF-1)), ...
188                     obj.VelInner*(1+2/(obj.NumMF-1)), obj.NumMF+2);
189
190                 rule_list = zeros(obj.NumMF^obj.NumInputs,obj.NumInputs);
191                 matrices = cell(obj.NumMF^obj.NumInputs,1);
192                 A = cell(6,6);
193                 B = cell(6,4);
194
195                 m = 1;
196
197                 [Jp,Jy,Jr,L,Kf,Kt,Ir,Kv] = HoverConstants();
198
199                 Q = diag([350 350 500 20 20 0]);
```

```
200                 R = diag([0.01 0.01 0.01 0.01]);
201
202                 syms roll pitch yaw p q r
203
204             for h = 1:obj.NumMF
205
206                 roll_a = round(obj.AngleIndex(find(obj.MemFunctions{1}{h
    }, 1 )),2);
207                 roll_b = round(obj.AngleIndex(find(obj.MemFunctions{1}{h
    }, 1, 'last' )),2);
208
209                 if h ~= 1 || h ~= obj.NumMF
210
211                     roll_a = pos(h);
212                     roll_b = pos(h+2);
213
214                 end
215
216                 roll_func1 = 4/(roll_b-roll_a)^2*(roll-roll_a); % roll wf
     1
217                 roll_func2 = -4/(roll_b-roll_a)^2*(roll-roll_b); % roll
    wf 2
218
219             for i = 1:obj.NumMF
220
221                 pitch_a = round(obj.AngleIndex(find(obj.MemFunctions{2}{i
    }, 1 )),2);
222                 pitch_b = round(obj.AngleIndex(find(obj.MemFunctions{2}{i
    }, 1, 'last' )),2);
223
224                 if i ~= 1 || i ~= obj.NumMF
225
226                     pitch_a = pos(i);
227                     pitch_b = pos(i+2);
228
229                 end
230
231                 pitch_func1 = 4/(pitch_b-pitch_a)^2*(pitch-pitch_a); %
    pitch wf 1
232                 pitch_func2 = -4/(pitch_b-pitch_a)^2*(pitch-pitch_b); %
    pitch wf 2
233
234             for j = 1:obj.NumMF
235
236                 p_a = round(obj.VelocityIndex(find(obj.MemFunctions{3}{j
    }, 1 )),1);
237                 p_b = round(obj.VelocityIndex(find(obj.MemFunctions{3}{j
    }, 1, 'last' )),1);
```

```
238
239             if j ~= 1 || j ~= obj.NumMF
240
241                 p_a = vel(j);
242                 p_b = vel(j+2);
243
244             end
245
246             p_func1 = 4/(p_b-p_a)^2*(p-p_a); % roll velocity wf 1
247             p_func2 = -4/(p_b-p_a)^2*(p-p_b); % roll velocity wf 2
248
249         for k = 1:obj.NumMF
250
251             q_a = round(obj.VelocityIndex(find(obj.MemFunctions{4}{k
    }, 1 )),1);
252             q_b = round(obj.VelocityIndex(find(obj.MemFunctions{4}{k
    }, 1, 'last' )),1);
253
254             if k ~= 1 || k ~= obj.NumMF
255
256                 q_a = vel(k);
257                 q_b = vel(k+2);
258
259             end
260
261             q_func1 = 4/(q_b-q_a)^2*(q-q_a); % pitch velocity wf 1
262             q_func2 = -4/(q_b-q_a)^2*(q-q_b); % pitch velocity wf 2
263
264         for l = 1:obj.NumMF
265
266             r_a = round(obj.VelocityIndex(find(obj.MemFunctions{5}{l
    }, 1 )),1);
267             r_b = round(obj.VelocityIndex(find(obj.MemFunctions{5}{l
    }, 1, 'last' )),1);
268
269             if l == 1 || l == obj.NumMF
270
271                 r_a = vel(l);
272                 r_b = vel(l+2);
273
274             end
275
276             r_func1 = 4/(r_b-r_a)^2*(r-r_a); % yaw velocity wf 1
277             r_func2 = -4/(r_b-r_a)^2*(r-r_b); % yaw velocity wf 2
278
279             A{1,4} = 1;
280             A{1,5} = sin(roll)*tan(pitch);
281             A{1,6} = cos(roll)*tan(pitch);
```

```
282                 A{2,5} = cos(roll);
283                 A{2,6} = -sin(roll);
284                 A{3,5} = sin(roll)/cos(pitch);
285                 A{3,6} = cos(roll)/cos(pitch);
286                 A{4,5} = (Jp-Jy)*r/Jr;
287                 A{4,6} = (Jp-Jy)*q/Jr;
288                 A{5,4} = (Jy-Jr)*r/Jp;
289                 A{5,6} = (Jy-Jr)*p/Jp;
290                 A{6,4} = (Jr-Jp)*q/Jy;
291                 A{6,5} = (Jr-Jp)*p/Jy;
292
293                 B{4,1} = -Ir*q*Kv/Jr;
294                 B{4,2} = -Ir*q*Kv/Jr;
295                 B{4,3} = L*Kf/Jr+Ir*q*Kv/Jr;
296                 B{4,4} = -L*Kf/Jr+Ir*q*Kv/Jr;
297                 B{5,1} = L*Kf/Jp+Ir*p*Kv/Jp;
298                 B{5,2} = -L*Kf/Jp+Ir*p*Kv/Jp;
299                 B{5,3} = -Ir*p*Kv/Jp;
300                 B{5,4} = -Ir*p*Kv/Jp;
301
302                 A_wl = zeros(6,6);
303                 B_wl = zeros(6,4);
304
305                 %%% A(1,4)
306
307                 A_wl(1,4) = 1;
308
309                 %%% A(1,5)
310
311                 F1 = int(roll_func1*A{1,5}, [roll_a (roll_a+roll_b)/2]);
312                 F2 = int(roll_func2*A{1,5}, [(roll_a+roll_b)/2 roll_b]);
313
314                 A_wl(1,5) = int(pitch_func1*(F1+F2), [pitch_a (pitch_a+
    pitch_b)/2])...
315                     + int(pitch_func2*(F1+F2), [(pitch_a+pitch_b)/2
    pitch_b]);
316
317                 %%% A(1,6)
318
319                 F1 = int(roll_func1*A{1,6}, [roll_a (roll_a+roll_b)/2]);
320                 F2 = int(roll_func2*A{1,6}, [(roll_a+roll_b)/2 roll_b]);
321
322                 A_wl(1,6) = int(pitch_func1*(F1+F2), [pitch_a (pitch_a+
    pitch_b)/2])...
323                     + int(pitch_func2*(F1+F2), [(pitch_a+pitch_b)/2
    pitch_b]);
324
325                 %%% A(2,5)
```

```
326
327             A_wl(2,5) = int(roll_func1*A{2,5}, [roll_a (roll_a+roll_b
    )/2])...
328                 + int(roll_func2*A{2,5}, [(roll_a+roll_b)/2 roll_b]);
329
330             %%% A(2,6)
331
332             A_wl(2,6) = int(roll_func1*A{2,6}, [roll_a (roll_a+roll_b
    )/2])...
333                 + int(roll_func2*A{2,6}, [(roll_a+roll_b)/2 roll_b]);
334
335             %%% A(3,5)
336
337             F1 = int(roll_func1*A{3,5}, [roll_a (roll_a+roll_b)/2]);
338             F2 = int(roll_func2*A{3,5}, [(roll_a+roll_b)/2 roll_b]);
339
340             A_wl(3,5) = int(pitch_func1*(F1+F2), [pitch_a (pitch_a+
    pitch_b)/2])...
341                 + int(pitch_func2*(F1+F2), [(pitch_a+pitch_b)/2
    pitch_b]);
342
343             %%% A(3,6)
344
345             F1 = int(roll_func1*A{3,6}, [roll_a (roll_a+roll_b)/2]);
346             F2 = int(roll_func2*A{3,6}, [(roll_a+roll_b)/2 roll_b]);
347
348             A_wl(3,6) = int(pitch_func1*(F1+F2), [pitch_a (pitch_a+
    pitch_b)/2])...
349                 + int(pitch_func2*(F1+F2), [(pitch_a+pitch_b)/2
    pitch_b]);
350
351             %%% A(4,6)
352
353             A_wl(4,6) = int(q_func1*A{4,6}, [q_a (q_a+q_b)/2])...
354                 + int(q_func2*A{4,6}, [(q_a+q_b)/2 q_b]);
355
356             %%% A(6,4)
357
358             A_wl(6,4) = int(q_func1*A{6,4}, [q_a (q_a+q_b)/2])...
359                 + int(q_func2*A{6,4}, [(q_a+q_b)/2 q_b]);
360
361             %%% A(5,4)
362
363             A_wl(5,4) = int(r_func1*A{5,4}, [r_a (r_a+r_b)/2])...
364                 + int(r_func2*A{5,4}, [(r_a+r_b)/2 r_b]);
365
366             %%% A(4,5)
367
```

```
368                 A_wl(4,5) = int(r_func1*A{4,5}, [r_a (r_a+r_b)/2])...
369                     + int(r_func2*A{4,5}, [(r_a+r_b)/2 r_b]);
370
371                 %%% A(6,5)
372
373                 A_wl(6,5) = int(p_func1*A{6,5}, [p_a (p_a+p_b)/2])...
374                     + int(p_func2*A{6,5}, [(p_a+p_b)/2 p_b]);
375
376                 %%% A(5,6)
377
378                 A_wl(5,6) = int(p_func1*A{5,6}, [p_a (p_a+p_b)/2])...
379                     + int(p_func2*A{5,6}, [(p_a+p_b)/2 p_b]);
380
381                 %%% B(4,1)
382
383                 B_wl(4,1) = int(q_func1*B{4,1}, [q_a (q_a+q_b)/2])...
384                     + int(q_func2*B{4,1}, [(q_a+q_b)/2 q_b]);
385
386                 %%% B(4,2)
387
388                 B_wl(4,2) = int(q_func1*B{4,2}, [q_a (q_a+q_b)/2])...
389                     + int(q_func2*B{4,2}, [(q_a+q_b)/2 q_b]);
390
391                 %%% B(4,3)
392
393                 B_wl(4,3) = int(q_func1*B{4,3}, [q_a (q_a+q_b)/2])...
394                     + int(q_func2*B{4,3}, [(q_a+q_b)/2 q_b]);
395
396                 %%% B(4,4)
397
398                 B_wl(4,4) = int(q_func1*B{4,4}, [q_a (q_a+q_b)/2])...
399                     + int(q_func2*B{4,4}, [(q_a+q_b)/2 q_b]);
400
401                 %%% B(5,1)
402
403                 B_wl(5,1) = int(p_func1*B{5,1}, [p_a (p_a+p_b)/2])...
404                     + int(p_func2*B{5,1}, [(p_a+p_b)/2 p_b]);
405
406                 %%% B(5,2)
407
408                 B_wl(5,2) = int(p_func1*B{5,2}, [p_a (p_a+p_b)/2])...
409                     + int(p_func2*B{5,2}, [(p_a+p_b)/2 p_b]);
410
411                 %%% B(5,3)
412
413                 B_wl(5,3) = int(p_func1*B{5,3}, [p_a (p_a+p_b)/2])...
414                     + int(p_func2*B{5,3}, [(p_a+p_b)/2 p_b]);
415
```

```
                %%% B(5,4)

                B_wl(5,4) = int(p_func1*B{5,4}, [p_a (p_a+p_b)/2])...
                    + int(p_func2*B{5,4}, [(p_a+p_b)/2 p_b]);

                %%% B(6,1) - B(6,4)

                B_wl(6,1) = -Kt/Jy;
                B_wl(6,2) = -Kt/Jy;
                B_wl(6,3) = Kt/Jy;
                B_wl(6,4) = Kt/Jy;

                %%%

                matrices{m} = lqr(A_wl, B_wl, Q, R);
                rule_list(m,:) = [h,i,j,k,l];
                m = m+1;

            end
            end
            end
            end
            end
            end
        end

        function plotMem(obj,index)
        % Plots the membership functions for specified input

            figure('Name',sprintf('membership function %d',index), ...
                'NumberTitle','off');
            hold on

            if index == 1 || index == 2
                for i = 1:obj.NumMF
                    plot(obj.AngleIndex,obj.MemFunctions{index}{i})
                end
                xlim([obj.AngleIndex(1), obj.AngleIndex(end)])
            else
                for i = 1:obj.NumMF
                    plot(obj.VelocityIndex,obj.MemFunctions{index}{i})
                end
                xlim([obj.VelocityIndex(1) obj.VelocityIndex(end)])
            end
        end
    end
end
```

C.12 EvalFuzzy.m

```
function output = EvalFuzzy(obj,input)
    % Evaluates the fuzzy inference system at the given input

rulelist = obj.RuleList;
NumberOfRules = length(rulelist);

w = zeros(NumberOfRules,1);

fuzzyVar = zeros(5,obj.NumMF);
for i = 1:5
    if i == 1 || i == 2
        z = abs(obj.AngleIndex-input(i));
    else
        z = abs(obj.VelocityIndex-input(i+1));
    end
    index = min(z)==z;
    for j = 1:obj.NumMF
        fuzzyVar(i,j) = [obj.MemFunctions{i}{j}(index)];
    end
end


for i = 1:NumberOfRules
    w(i) = min([...
        fuzzyVar(1,rulelist(i,1)),...
        fuzzyVar(2,rulelist(i,2)),...
        fuzzyVar(3,rulelist(i,3)),...
        fuzzyVar(4,rulelist(i,4)),...
        fuzzyVar(5,rulelist(i,5))]);
end

w_nz = find(w);
u = zeros(4,nnz(w));

for i = 1:nnz(w)
    u(:,i) = w(w_nz(i)).*obj.MatrixList{w_nz(i)}*input;
end

output = sum(u,2)/sum(w(w_nz));
```

C.13 CalculateCost.m

```
function J = CalculateCost(x,u,Q,R)
```

```
2
3 quad = zeros(length(x)-1,1);
4
5 for k = 1:(length(x)-1)
6     quad(k) = x(k,:)*Q*x(k,:)' + u(x(k,:)')*R*u(x(k,:)')';
7 end
8
9 J = x(end,:)*Q*x(end,:)' + sum(quad);
```

C.14 CostPlot.m

```
1 clear;close;clc;tic
2 %%
3
4 NOruns = 80;
5 roll_final = deg2rad(37.5);
6 pitch_final = deg2rad(37.5);
7 yaw_final = deg2rad(180);
8 mf_final = 13;
9
10 roll = linspace(0,roll_final,NOruns);
11 pitch = linspace(0,pitch_final,NOruns);
12 yaw = linspace(0,yaw_final,NOruns);
13
14 K = GainMatrix(0,0,0,0,0,0);
15
16 Q = diag([350 350 500 20 20 0]);
17 R = diag([0.01 0.01 0.01 0.01]);
18
19 jp = 0.055;
20 jy = 0.110;
21 jr = 0.055;
22 l = 0.197;
23 kf = 0.119;
24 kt = 0.0036;
25
26 tspan = 0:0.01:1;
27
28 J_diff = zeros(1,NOruns);
29
30 ref(1,1) = 0;
31 ref(2,1) = 0;
32 ref(3,1) = 0;
33 ref(4,1) = 0;
34 ref(5,1) = 0;
35 ref(6,1) = 0;
```

```
36
37 figure('Name','Percentage improvement','NumberTitle','off');
38 hold on
39
40 for h = 3:2:mf_final
41
42     FIS = tsk(h);
43
44     u_fis = @(x)-EvalFuzzy(FIS,x - ref)';
45     u_lin = @(x)(-K*(x-ref))';
46
47     for i = 1:NOruns
48
49         x0(1,1)    = roll(i);
50         x0(2,1)    = pitch(i);
51         x0(3,1)    = yaw(i);
52         x0(4,1)    = 0;
53         x0(5,1)    = 0;
54         x0(6,1)    = 0;
55
56         [~,x_fis] = ode45(@(t,x)hover_sim(x,jp,jr,jy,l,kf,kt,u_fis(x)),
    tspan,x0);
57         [t,x_lin] = ode45(@(t,x)hover_sim(x,jp,jr,jy,l,kf,kt,u_lin(x)),
    tspan,x0);
58
59         J_fis      = CalculateCost(x_fis,u_fis,Q,R);
60         J_lin      = CalculateCost(x_lin,u_lin,Q,R);
61
62         J_diff(i) = (J_lin-J_fis)/J_lin*100;
63
64     end
65
66     % Plotting section
67
68     if h == mf_final
69         plot(rad2deg(roll),J_diff,'DisplayName',string(h)+' mf')
70         legend
71         xlim([0 rad2deg(roll(end))])
72         ylim([0 J_diff(end)])
73         ax(1)=gca;
74         set(ax(1),'Position',[0.12 0.12 0.80 0.70])
75         set(ax(1),'XColor','k','YColor','k');
76         ax(2)=axes('Position',get(ax(1),'Position'),...
77            'XAxisLocation','top',...
78            'YAxisLocation','right',...
79            'Color','none',...
80            'XColor','k','YColor','k');
81         set(gca, 'YTick', [])
```

```
82          set(ax,'box','off')
83          hl2=line(rad2deg(yaw),J_diff,'Color','k','Parent',ax(2));
84          hl2.LineStyle = 'none';
85          set(get(ax(1),'xlabel'),'string','$\theta_r,\theta_p$','
    Interpreter','latex')
86          set(get(ax(2),'xlabel'),'string','$\theta_y$','Interpreter','
    latex')
87          set(get(ax(1),'ylabel'),'string','Percentage improvement in
    quadratic cost')
88          ytickformat('percentage')
89          ax(1).YAxis.TickLabelFormat = '%g%%';
90      else
91          plot(rad2deg(roll),J_diff,'DisplayName',string(h)+' mf')
92      end
93      drawnow
94
95      % End of plotting section
96
97  end
```

Appendix D

Simulink Models

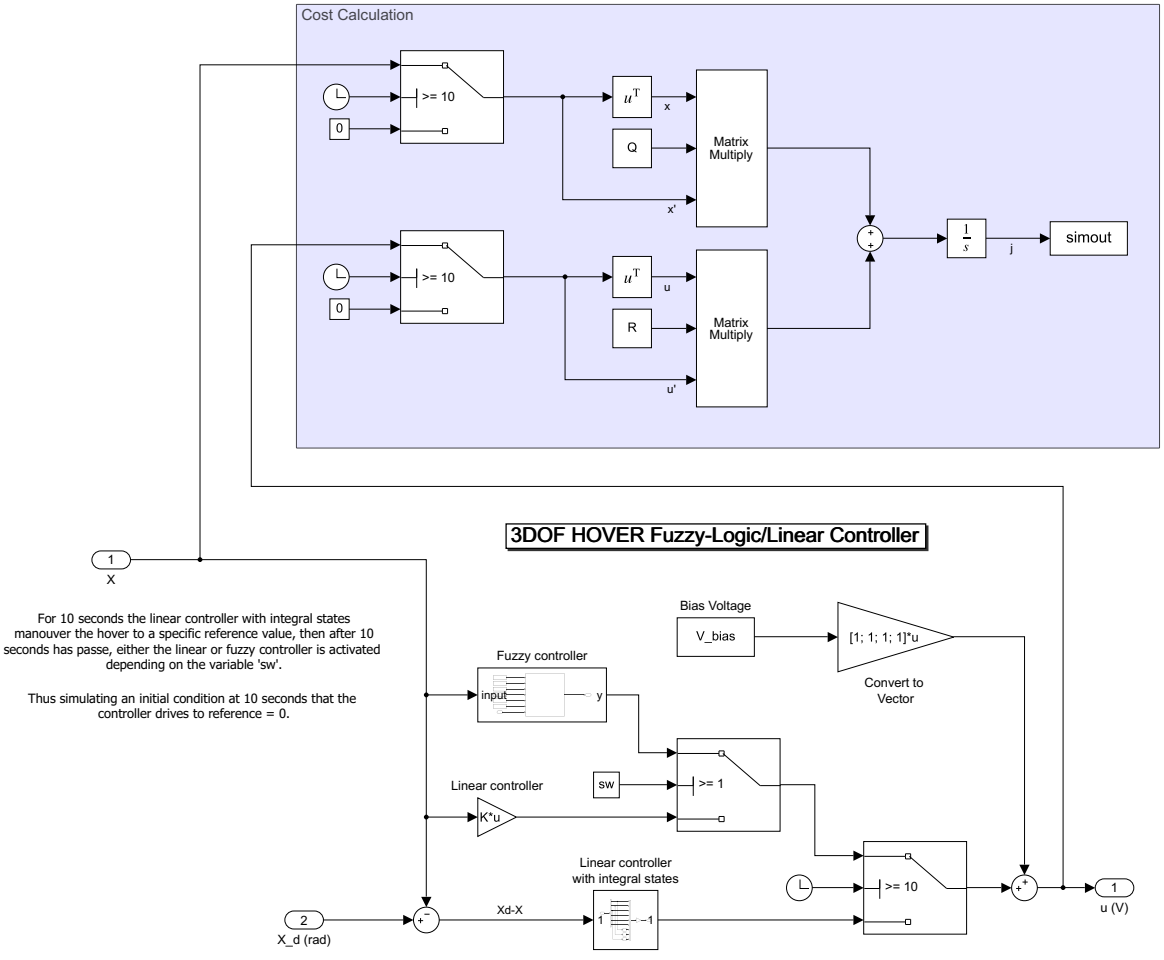
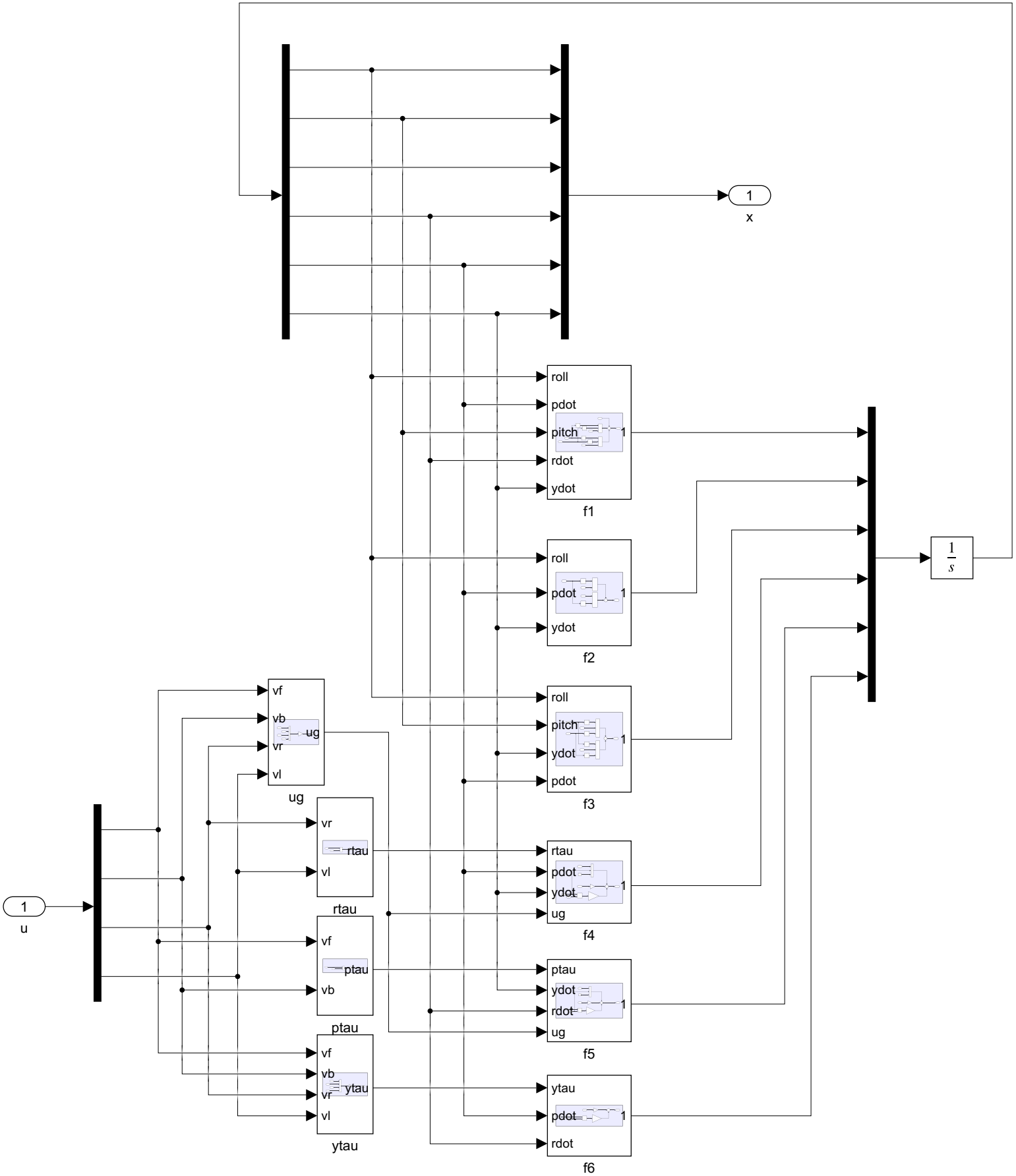
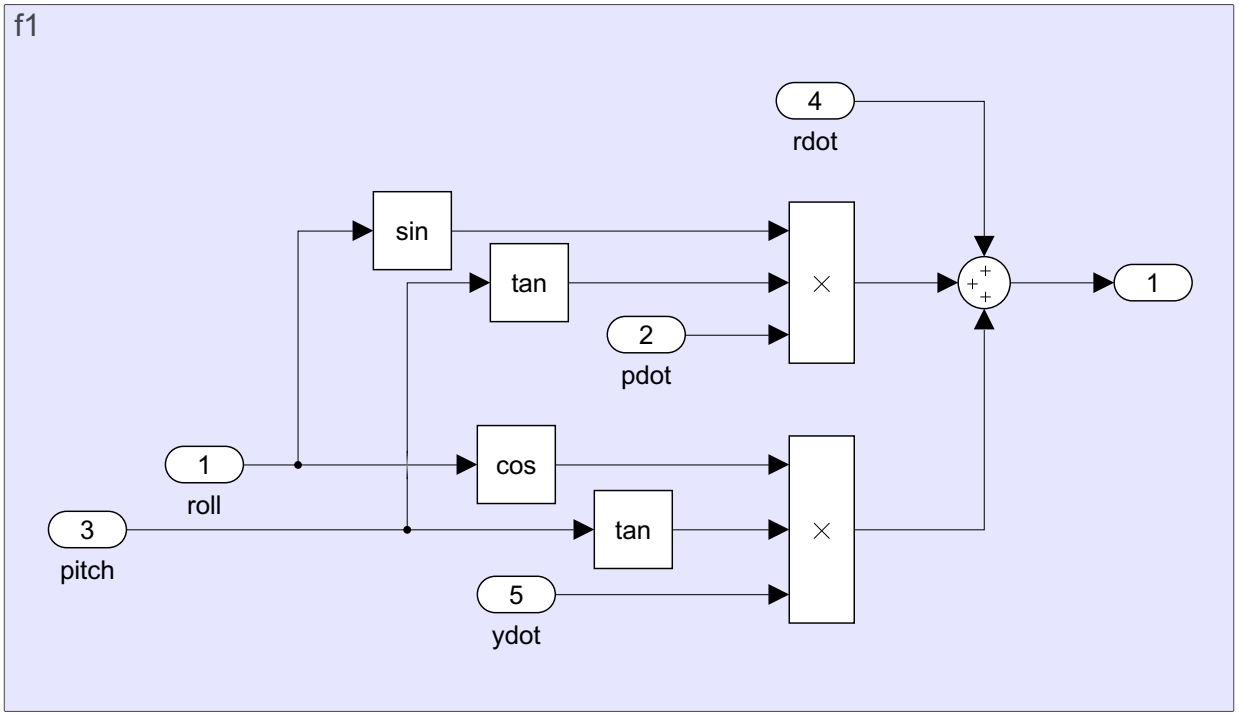
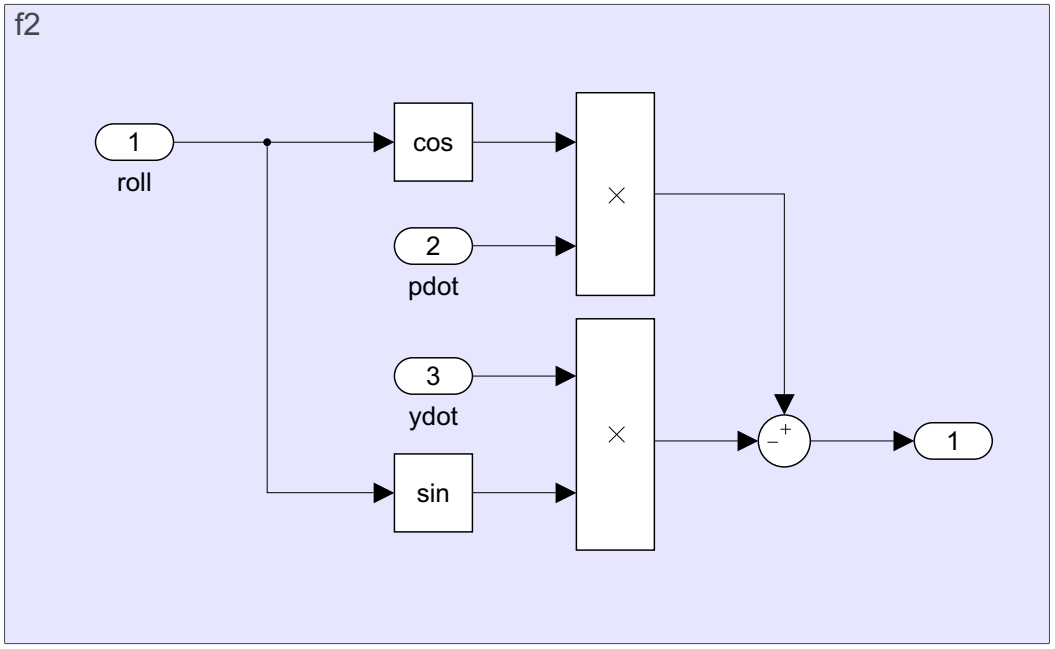
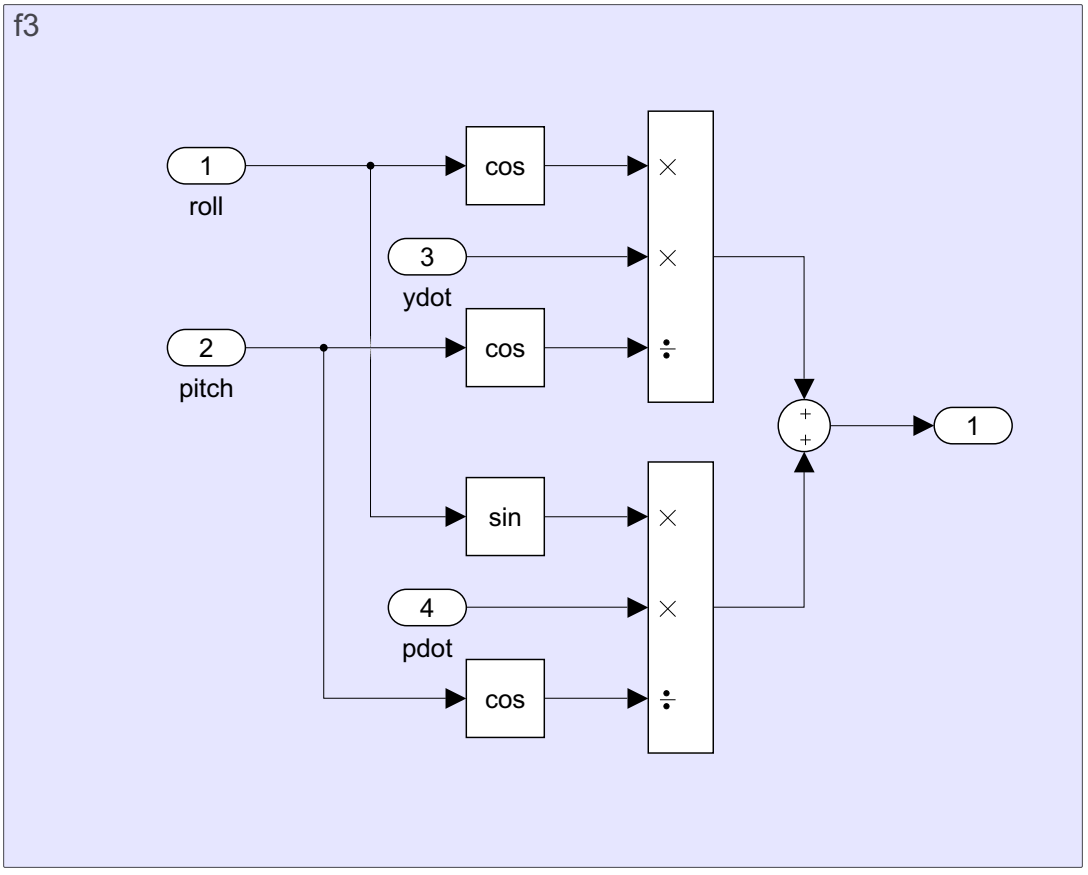


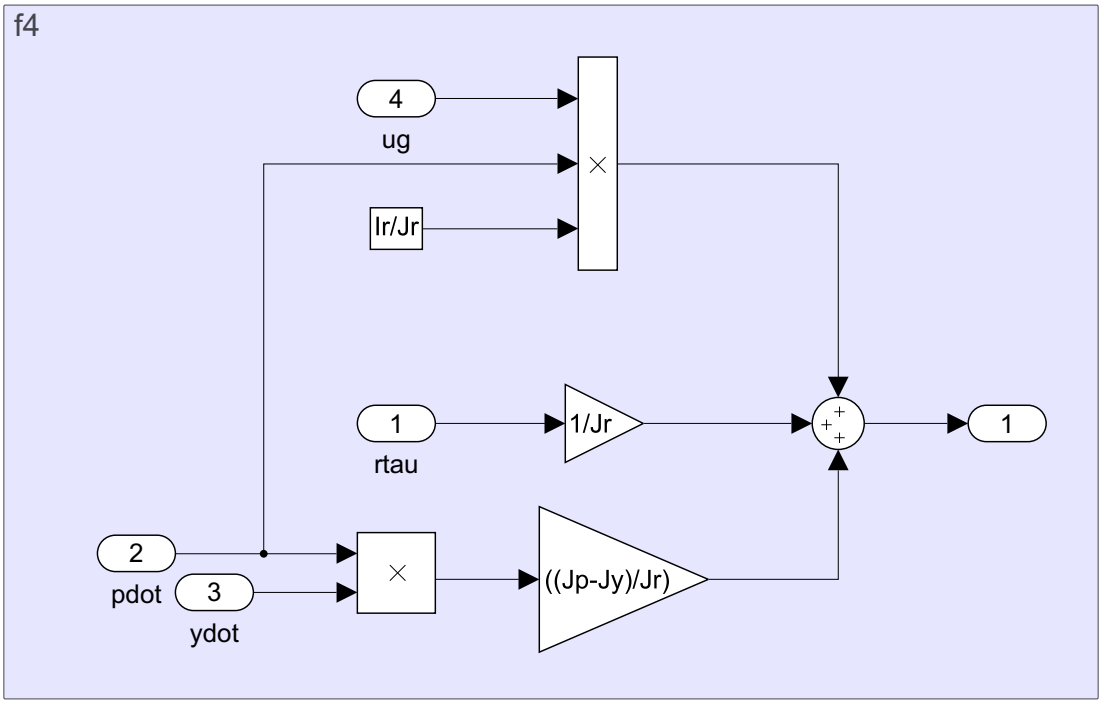
Fig D.1: The cost function implemented in Simulink.

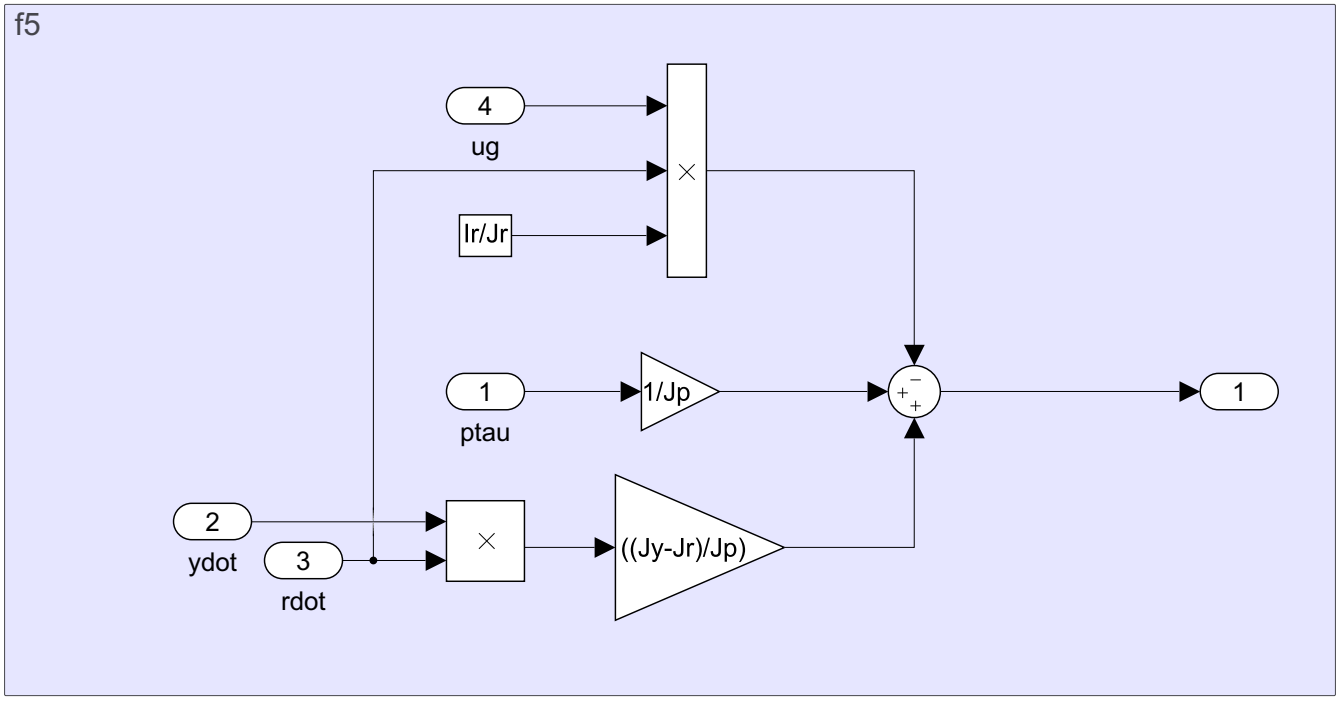

1
x
roll
pdot
pitch
rdot
ydot
1
f1
roll
pdot
ydot
1
f2
roll
pitch
ydot
pdot
1
f3
rtau
pdot
ydot
ug
1
f4
ptau
ydot
rdot
ug
1
f5
ytau
pdot
rdot
1
f6
1
s
1
u
vf
vb
vr
vl
ug
vr
rtau
vl
rtau
vf
ptau
vb
ptau
vf
vb
vr
ytau
vl
ytau

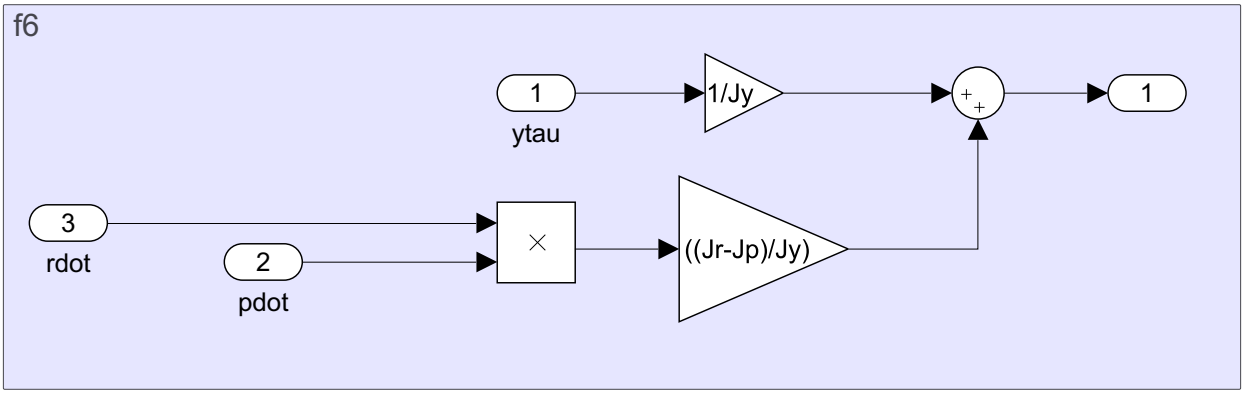

f1
4
rdot
sin
tan
2
pdot
1
roll
cos
3
pitch
tan
5
ydot
1

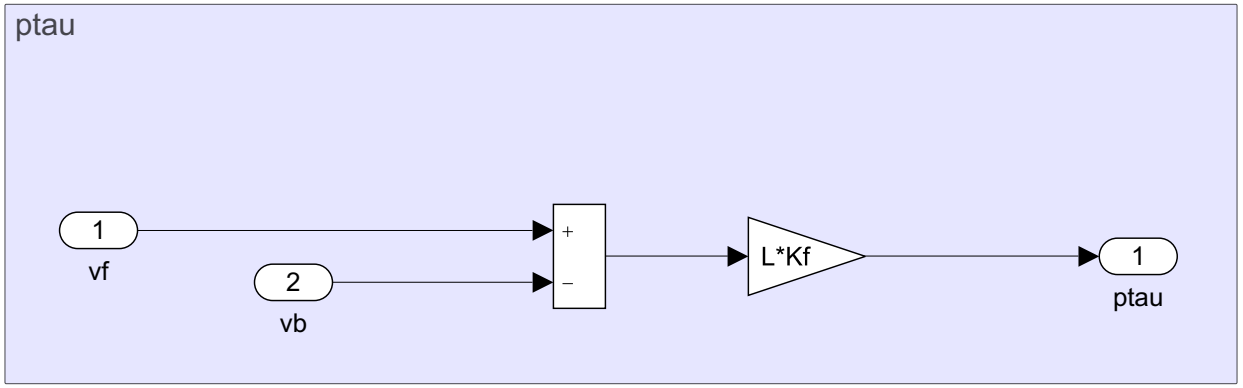

f2
1
roll
cos
2
pdot
3
ydot
sin
1

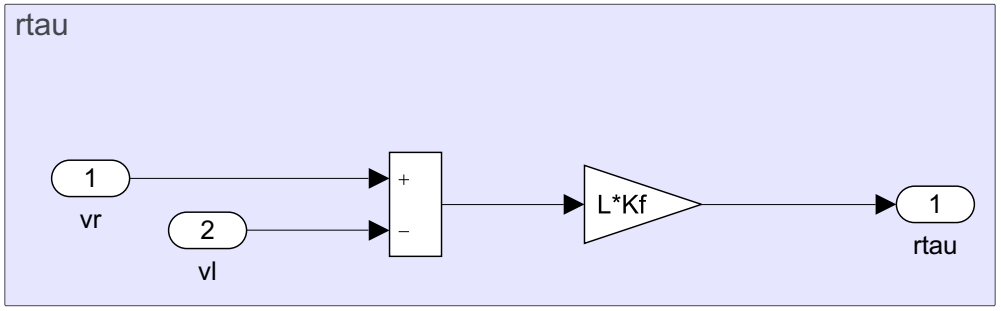

f3
1
roll
2
pitch
cos
cos
sin
cos
3
ydot
4
pdot
1

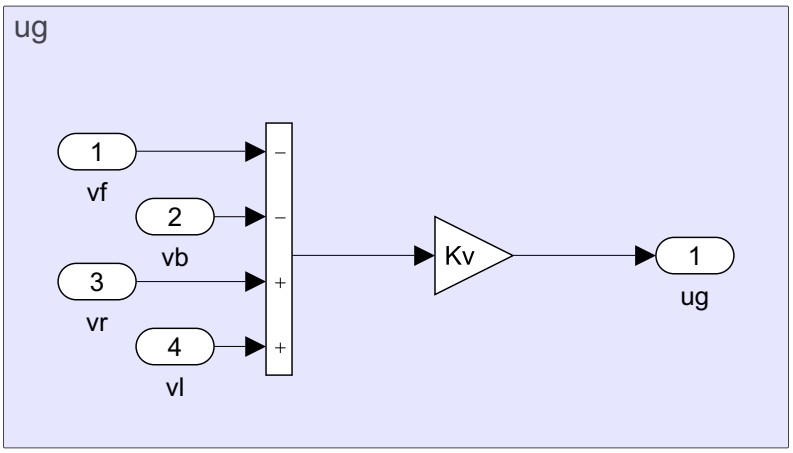

f4
4
ug
Ir/Jr
1
rtau
1/Jr
1
2
pdot
3
ydot
((Jp-Jy)/Jr)

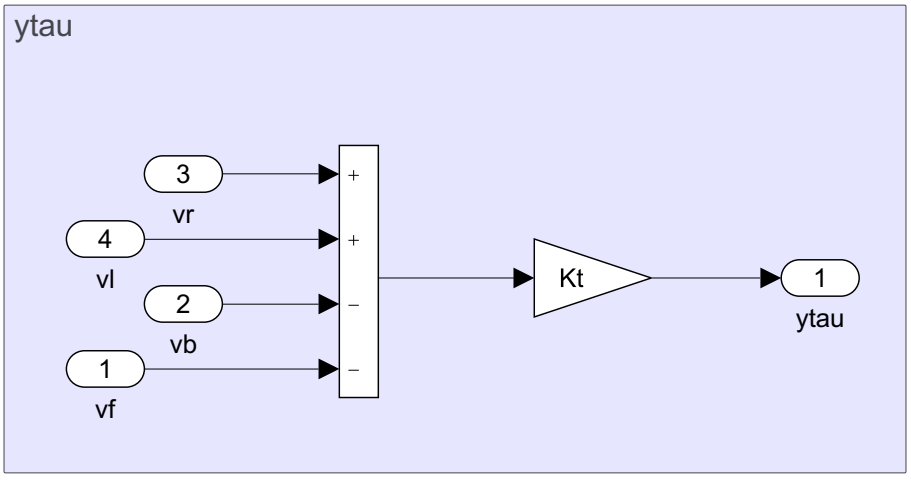

f5
4
ug
Ir/Jr
1
ptau
1/Jp
1
2
ydot
3
rdot
((Jy-Jr)/Jp)


f6
1
ytau
1/Jy
1
3
rdot
2
pdot
((Jr-Jp)/Jy)


ptau
1
vf
2
vb
+
−
L*Kf
1
ptau


rtau
1
vr
2
vl
+
−
L*Kf
1
rtau


ug
1
vf
2
vb
3
vr
4
vl
Kv
1
ug


ytau
3
vr
4
vl
2
vb
1
vf
+
+
−
−
Kt
1
ytau