12-15-2016

Fuzzy Attitude Control of Solar Sail

Joshua Baculi
Santa Clara University

Santa Clara University

Department of Mechanical Engineering

Date: December 15, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Joshua Baculi

ENTITLED

Fuzzy Attitude Control of Solar Sail

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

M. A. Ayoubi

Thesis Advisor
Dr. Mohammad A. Ayoubi

Thesis Reader
Dr. Maryam Khanbaghi

Chairman of Department
Dr. Drazen Fabris

Fuzzy Attitude Control of Solar Sail

By

Joshua Baculi

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master's of Science
in Mechanical Engineering
in the School of Engineering at
Santa Clara University, 2016

Santa Clara, California

Acknowledgments

I would first like to thank my thesis advisor Dr. Ayoubi of the Department of Mechanical Engineering at Santa Clara University. The door to Prof. Ayoubi's office was always open whenever I ran into a trouble spot or had a question about my research or writing.

I would also like to thank the NASA Aeronautics Research Mission Directorate for funding this research by awarding him the Undergraduate Aeronautics Scholarship (Grant Number NNX14AT05H).

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Fuzzy Attitude Control of Solar Sail

Joshua Baculi

Department of Mechanical Engineering
Santa Clara University
Santa Clara, California
2016

ABSTRACT

This study presents various fuzzy type controllers for a solar sail. First, a Twin Parallel Distributed Compensator (TPDC) is built for a Takagi-Sugeno fuzzy model of the solar-sail. The T-S fuzzy model is constructed by linearizing the existing nonlinear equations fo motion of the solar sail. The T-S fuzzy model is used to solve a set of linear matrix inequalities to derive state feedback controller gains. The TPDC controls the solar sail using a combination of reaction wheels and roll stabilizer bars for attitude control and trim masses for disturbance rejection. The TPDC tracks and stabilizes the solar sail to any desired state in the presence of parameter uncertainties and external disturbances while satisfying actuator constraints. The performance of the TPDC is compared to a Ziegler-Nichols tuned proportional-integral-derivative (PID) controller. Numerical simulation shows the TPDC outperforms he PID controller when stabilizing the solar sail to a desired state. When compared to the particle swarm optimized PID controller, the TPDC has a slower response, irrespective of the initial conditions and desired states.

To control the solar sail using trim masses, a hybrid fuzzy-logic supervisor with a PID attitude controller is built. Particle swarm optimization is also used to obtain gains for roll, pitch and yaw PID controllers. The calculated PID gains are used to build a fuzzy supervisor that tunes the PID controller gains based on the Euler angle error and error rate. The proposed PID controller stabilizes the solar sail about any commanded input

from any initial condition. The supervisory fuzzy PID controller is shown to be robust model uncertainties and outperforms a particle swarm optimized PID controller when stabilized about a non-optimal state from a non-optimal initial condition.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

As mankind's scientific ambitions grow, so must the technology implemented evolve to match, which is why the propellantless solar-sail has been a rising development in space travel. Some solar-sail applications proposed by Wie [Wie04a] include high performance missions to neighboring satellites in the inner solar system such as Mercury and Venus, and high velocity missions to the outer planets. The first solar sail, IKAROS, was successfully launched on May 21, 2010 [YT11]. NASA's Space Launch System (SLS), set to launch in 2018, has two secondary payloads that will utilize solar sails for both rendezvous and orbit missions. One of SLS's payloads, the Near Earth Asteroid (NEA) Scout, will rendezvous with a small asteroid for observation. The other payload, the Lunar Flashlight, will utilize a solar sail similar to the NEA Scout to search for resources on the moon. The solar sail will orbit the moon while illuminating shadowed areas of the moon with the sunlight that is reflected off its sail. Also, the Planetary Society successfully deployed their Light Sail in June 2015 and are planning on a full solar sail demonstration some time in 2016.

1.1 Solar Sail Background

The appeal of solar sails is evident by their premise. Similar to how kites are pushed higher into the sky by wind, solar sails utilize the solar radiation pressure (SRP) from the Sun as a means of thrust. Because solar sails rely on the SRP from the sun, it has

a constant source of energy to propel it, thus eliminating the need for fueling stations during long trips [Wie08]. This enables solar sails to be capable of round-trip missions to other satellites in our solar system. There is also a thermal radiation pressure whose effect will not be studied in this paper [vdHMTM15]. Not only does the SRP provide thrust, it can also be utilized for attitude control of the sail. This is done by creating an offset between the sail's center of mass *cm* and center of pressure *cp*. This offset can be manipulated to create various torques for attitude control.

Several types of missions for solar sails have been categorized based on the trajectory. Macdonald and McInnes identified solar sail applications into seven categories: planet-centered and short orbit periods, highly non-Keplerian orbits, inner solar system rendezvous, outer solar system rendezvous, outer solar system flyby, solar missions, and beyond Neptune [MM10]. For scope of this research, we focus on the first category for an Earth-centered sun-synchronous orbit. Some missions for this type of orbit include the GeoSail concept and the Mercury Sun-Synchronous Orbiter. The former aims to achieve long-term study of the Earth's magnetosphere [MHM+07], while the latter strives to study the least-explored planet in the solar system: Mercury [LSL+96].

Similar to how there are many different shapes for kites, there are different shapes for solar sails as well. The three main shapes shown in Figure 1.1 are heliogyro, square, and disk. Price et al. summarized the pros and cons of each shape [PAG+01]. The square sail is capable of fast turn rates and controlled deployments but is limited to about 100,000 m^2 in sail area. The spinning disk sail has a very stable safe mode but has very slow turn rates and a more difficultly controlled deployment. The heliogyro has the simplest and most controlled deployment but also has a very difficult control system and high film stress. Because most near-term solar sails will be small in size, the square solar sail is chose for this study.

Within the square solar sail category, Wie goes on to elaborate on several attitude con-

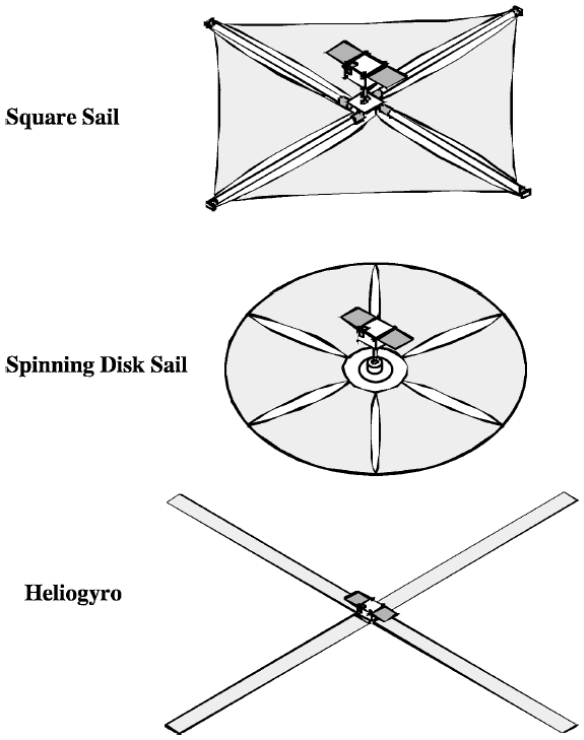


Fig. 1.1: The three main solar sail shapes [Wie04a]

trol systems (ACS) [Wie04a, Wie04b]. These control methods include control vanes, gimbaled control boom, shifting and tilting panels, reflectivity modulation, and translating masses. IKAROS uses reflectivity modulation to change the reflectivity coefficient of various areas of its membranes, thus changing the location of the *cp* and creating a controllable offset with the *cm*. A novel idea proposed by Fu and Eke [FE15] is the wing tip method where the wing tips of the sail are pulled along the masts towards the center. This creates a billowing effect in the wing that changes the location of the *cp*. The appeal of the wing tip method is its robustness to sail size, but its equations of motion are highly complicated and nonlinear [FGE], and so it will not be used for this study.

Recently Fu and Eke [FE15] summarized the disadvantages of various control methods for solar sails described by Wie [Wie04b]. For the control vanes, gimbaled masses, shifted panels, controllable film reflectivity, and sliding masses methods, larger sail areas will

require larger mass to control the sail. This limits the size of the sail in relation to the mass of the control elements.

For this study, we choose the translating or sliding masses configuration on a 40m×40m square solar sail. The disadvantages to the translating masses technique previously mentioned are unconcerning at this size. This technique changes the location of the *cm* by sliding masses within the ballasts of the sail. The offset between *cm* and *cp* can allow for the sail's pitch and yaw to be controlled. Pitch and yaw are controllable by the masses because their axes are coplanar on the sail, whereas the roll axis is perpendicular to the sail's membrane. To control the solar sail's roll angle, roll stabilizer bars are employed. Furthermore, the sail's pitch and yaw cannot be controlled when the face of the solar sail is pointing away from the Sun in a manner that the SRP does not hit any of the sail's membrane. In order to compensate for this, secondary torques in the pitch and yaw directions must be employed if stabilization around a zero-thrust attitude is desired. These secondary torques are assumed to be readily available through reaction wheels. When the reaction wheels act as the primary torque inputs, the trim masses can be set to steady-state values to cancel out SRP disturbances. Other forms of secondary torques include pulsed-plasma thrusters (PPT) [WMPT04] and magnetic torquers [Wie08].

1.2 Attitude Control Techniques

The focus of this paper is on control techniques for attitude stabilization of the solar sail. Two control techniques are applied to a solar sail with translating masses: the Twin Parallel Distributed Compensation (TPDC) technique and a fuzzy-logic supervisory proportional-integral-derivative (PID) controller. A basic PID controller is also built to act as a baseline controller.

The Twin Parallel Distributed Compensator (TPDC) [TTYW99a] with actuator con-

straints is employed to achieve servo control of the solar sail. The Takagi-Sugeno (T-S) fuzzy model approximates a global nonlinear model with local linear models that are blended using a set of fuzzy rules and membership functions. The Parallel Distributed Compensation technique is applied to the T-S fuzzy model to design a full-state feedback controller for each local linear model. These local controllers are then blended to form a global asymptotically stable controller in the same manner the T-S model approximates the global nonlinear model. The TPDC controls the solar sail to follow a reference input, in this case a desired state of Euler angles, by having a primary PDC regulate the dynamics of the solar sail and a secondary PDC to regulate the reference model. This allows the solar sail to be commanded to any desired state from any initial condition. Fuzzy control of spacecraft has been shown to allow for attitude control [MA, SA15] that is robust to model uncertainties [SA14]. The simple PDC technique was also applied to stabilize a solar sail to an arbitrary state using a change of variables [BA16a].

Due to the simple structure and easy implementation of the PID controllers, we were motivated to choose a combination of a PID controller with a Fuzzy-Logic Supervisor (FLS) for attitude control. The FLS is a Mamdani controller which utilizes some heuristic rules to fine tune the gains of the PID controllers [PYR98]. The FLS interpolates between a set of PID gains based on system performance characteristics such as error, error rate, and overshoot. Particle swarm optimization and Ziegler-Nichols are used to find the range of PID gains that are employed to build the fuzzy supervisor. Supervisory controllers have been employed in various dynamics problems. For instance, Wai et al. utilized a supervisory controller with sliding-mode technique for levitated and propulsive control of a maglev transportation system [WCL10]. Kumar and Raja employed a fuzzy supervisor with a PID controller to control the 5DOF of a robot arm [KR14]. Rahnamai, Arabshahi, and Gray also used a fuzzy supervisor on an optimal regulator

for spacecraft formation flying [RAG03].

Chapter 2

Mathematical Model

2.1 Model Description

Consider the solar sail in a dawn-dusk sun-synchronous (DDSS) orbit around the Earth, as shown in Figure 2.1. The sun-synchronous orbit has its plane near perpendicular to the Earth's orbit plane and its normal in the direction of the SRP. Figure 2.1 shows the solar sail's coordinate frame relative to the Earth where the roll axis is always perpendicular to the sail's area and its yaw axis points towards the Earth. Figure 2.2 shows the sail in full-thrust mode (left) at a yaw angle of $\pm 90°$ and zero-thrust mode (right) at $0°$.

The solar sail configuration shown in Figure 2.1 employs sliding masses for attitude control and to offset the SRP disturbance in pitch and yaw created by the sail's inherent cm/cp offset.

2.2 Equations of Motion

The equations of motion for a solar sail with moving control masses are given by [Wie08]:

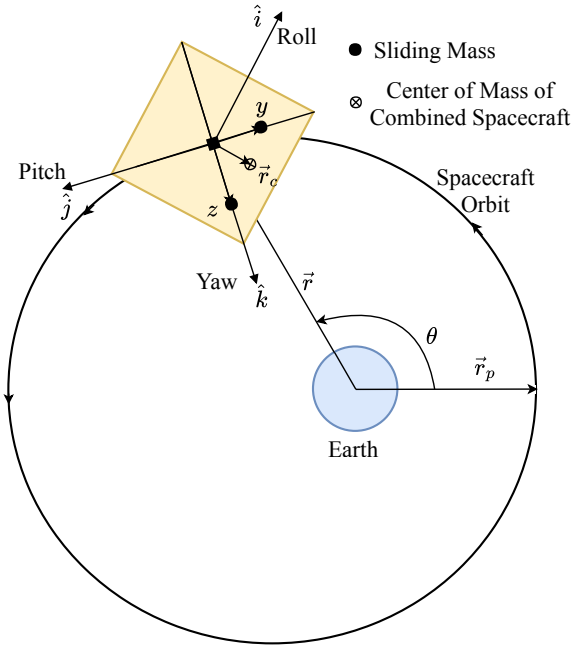


Fig. 2.1: Solar sail with translating ballast masses in an Earth-centered orbit.

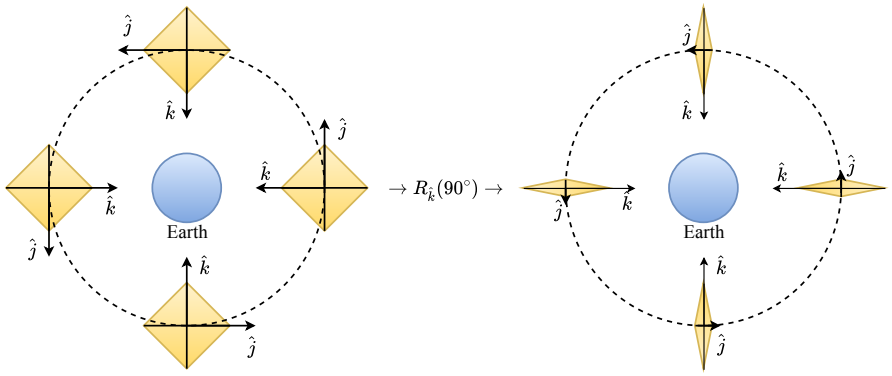


Fig. 2.2: (Left) Solar-sail at full-thrust mode. (Right) Solar-sail at zero thrust mode.

$$J_x\dot{\omega}_x = (J_y - J_z)\omega_y\omega_z - 3n^2(J_y - J_z)\phi + 0.5\epsilon F + T_x \tag{2.1}$$

$$J_y\dot{\omega}_y = (J_z - J_x)\omega_z\omega_x - 3n^2(J_x - J_z)\theta + \frac{m}{M+2m}zF + \epsilon F + T_y \tag{2.2}$$

$$J_z\dot{\omega}_z = (J_x - J_y)\omega_x\omega_y - \frac{m}{M+2m}yF + \epsilon F + Tz \tag{2.3}$$

where the moments of inertia J are functions of the principle moments of inertia of the spacecraft without trim masses I_x, I_y, I_z, the trim mass locations y and z, and the reduced mass m_r and are defined as

$$
\begin{aligned}
J_x &\triangleq I_x + m_r(y^2 + z^2) \\
J_y &\triangleq I_y + m_r z^2 \\
J_z &\triangleq I_z + m_r y^2 \\
m_r &\triangleq \frac{m(M+m)}{M+2m}.
\end{aligned}
$$

Because $M >> m$, it can be assumed that $m_r \approx m$.

For small roll and pitch maneuvers (less than 10°) with large yaw maneuvers (less than 90°), the kinematical equations for Euler angles are simplified to

$$\omega_x \approx \dot{\phi} - n \sin\psi \tag{2.4}$$

$$\omega_y \approx \dot{\theta} - n \cos\psi \tag{2.5}$$

$$\omega_z \approx \dot{\psi} - n(\theta \sin\psi - \phi \cos\psi) \tag{2.6}$$

Substitute Equations (2.4), (2.5), and (2.6) into Equations (2.1), (2.2), and (2.3). The

final equations of motion are as follows:

$$J_x\ddot{\phi} + n^2(J_y - J_z)(3 + \cos^2\psi)\phi - n^2(J_y - J_z)(\cos\psi\sin\psi)\theta$$
$$- n(J_x - J_y + J_z)(\cos\psi)\dot{\psi} = 0.5\epsilon F + T_x \tag{2.7}$$
$$J_y\ddot{\theta} + n^2(J_x - J_z)(3 + \sin^2\psi)\theta - n^2(J_x - J_z)(\cos\psi\sin\psi)\phi$$
$$- n(J_x - J_y - J_z)(\sin\psi)\dot{\psi} = \frac{m}{M + 2m}zF + \epsilon F + T_y \tag{2.8}$$
$$J_z\ddot{\psi} + n^2(J_y - J_x)\sin\psi\cos\psi + n(J_x - J_y + J_z)(\cos\psi)\dot{\phi}$$
$$+ n(J_x - J_y - J_z)(\sin\psi)\dot{\theta} = -\frac{m}{M + 2m}yF + \epsilon F + T_z \tag{2.9}$$

where $\vec{F}$ denotes the solar radiation force and can be written as $\vec{F} = -F\vec{i}$ where $F = F_s\cos^2\alpha$, $\alpha = \frac{\pi}{2} + \psi$, and n is an orbital parameter that is a function of the gravitational parameter and orbit radius, $\sqrt{\mu/a^3}$. The parameter ϵ denotes the cm/cp offset inherent with uncertainties. When the trim masses are used for disturbance cancellation, z and y are set to steady-state values and T_x, T_y, and T_z are the control inputs. When the trim masses are the control inputs, T_y and T_z are set to 0.

Chapter 3

Takagi-Sugeno (T-S) Fuzzy Model

3.1 Building the T-S Fuzzy Model

The T-S fuzzy model implemented by Tanaka and Wang [TW01] is used in this study:. Each model rule has the following format:

Model Rule i:

IF $z_1(t)$ is about $\mu_{i1}[z_1(t)]$, and $\ldots$, $z_p(t)$ is about $\mu_{ip}[z_p(t)]$, THEN

$$\dot{x}(t) = A_i x(t) + B_i u(t) \; ; \quad (i = 1, 2, \ldots, r) \tag{3.1}$$

where r is the number of rules, $z(t) = [z_1(t), \ldots, z_p(t)]$ is the vector of premise variables, and μ_{ip} is the fuzzy membership function for each rule i and each premise variable $z_p(t)$. $A_i \in \mathbb{R}^{n \times n}$ is the nominal system matrix, $B_i \in \mathbb{R}^{n \times m}$ is the nominal control matrix, $x(t) \in \mathbb{R}^{n \times 1}$ is the state vector, and $u(t) \in \mathbb{R}^{m \times 1}$ is the control input.

The $T - norm$ product is used to determine the firing strength of each rule:

$$w_i[z(t)] = \prod_{j=1}^{p} \mu_{ij}[z_j(t)]; \quad 0 \leq \mu_{ij}(z_j) \leq 1 \tag{3.2}$$

and the fuzzy basis functions are determined from

$$h_i[z(t)] = \frac{w_i[z(t)]}{\sum_{i=1}^{r} w_i[z(t)]} \tag{3.3}$$

where the fuzzy basis functions, $h_i[z(t)]$, have the following properties:

$$\sum_{i=1}^{r} h_i[z(t)] = 1, \qquad h_i[z(t)] \geq 0; \quad (i = 1, \ldots, r) \tag{3.4}$$

The overall system can be approximated by combining the T-S model

$$\Sigma_{TS} : \begin{cases} \dot{x}(t) = \sum_{i=1}^{r} h_i[z(t)] \; [A_i x(t) + B_i u(t)] \\ y(t) = \sum_{i=1}^{r} h_i[z(t)] C_i x(t) \end{cases} \tag{3.5}$$

where the state vector is defined as $x \triangleq [\phi \quad \theta \quad \psi \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T$.

The T-S fuzzy model for the solar sail can be constructed by choosing the premise variables, $z_1 \triangleq \phi$, $z_2 \triangleq \theta$, and $z_3 \triangleq \psi$. The fuzzy membership functions for each $z_j(t)$ for $(j = 1, 2, 3)$ are shown in Figures 3.1 and 3.2. Because Equations (2.7)–(2.9) assume small roll and pitch angles and large yaw angles, z_1 and z_2 consist of two rules whereas z_3 has seven rules, resulting in 28 total rules. The universe of discourse for the fuzzy premise variables are $\phi(t), \theta(t) \in [-5°, 5°]$ and $\psi(t) \in [-105°, -85°, -65°, -45°, -25°, -5°, 15°]$.

We linearize Equations (2.7)–(2.9) to find matrices A and B. We then use the trim inputs for each rule to determine the fuzzy A_i and B_i matrices. The parametric forms of A and B are

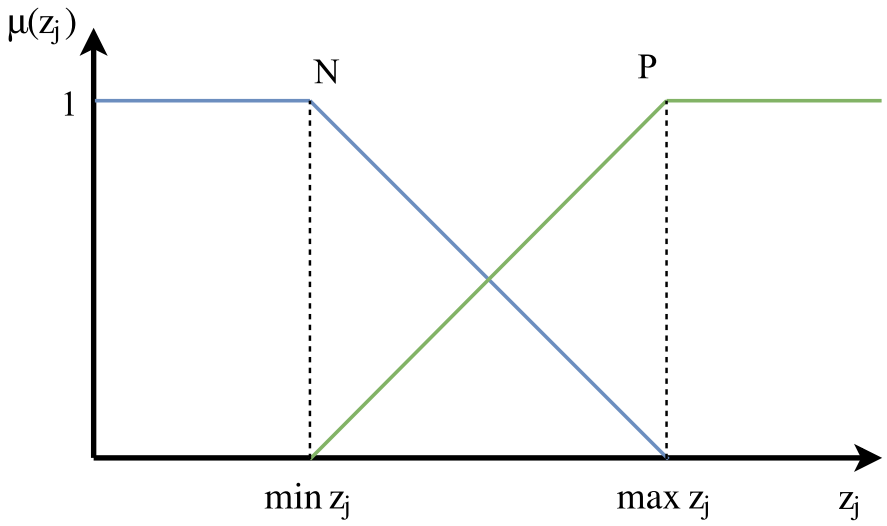


Fig. 3.1: Fuzzy membership function of premise variables ϕ and θ. "N" denotes negative value and "P" denotes positive value.

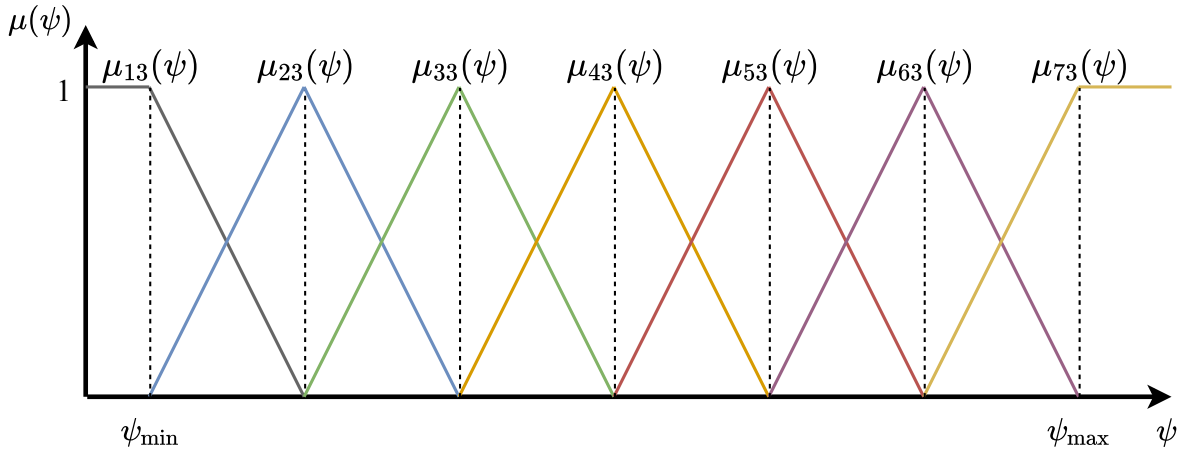


Fig. 3.2: Fuzzy membership function of premise variable ψ.

$$f_1(x,u,t) = \dot{\phi} \tag{3.6}$$

$$f_2(x,u,t) = \dot{\theta} \tag{3.7}$$

$$f_3(x,u,t) = \dot{\psi} \tag{3.8}$$

$$f_4(x,u,t) = \frac{1}{J_x}[-n^2(J_y - J_z)(3+\cos^2\psi)\phi + n^2(J_y - J_z)(\cos\psi\sin\psi)\theta$$
$$+ n(J_x - J_y + J_z)(\cos\psi)\dot{\psi} + 0.5\epsilon F + Tx] \tag{3.9}$$
$$f_5(x,u,t) = \frac{1}{J_y}[-n^2(J_x - J_z)(3+\sin^2\psi)\theta + n^2(J_x - J_z)(\cos\psi\sin\psi)\phi$$
$$+ n(J_x - J_y - J_z)(\sin\psi)\dot{\psi} + \frac{m}{M+2m}zF + \epsilon F + Ty] \tag{3.10}$$
$$f_6(x,u,t) = \frac{1}{J_z}[-n^2(J_y - J_x)\sin\psi\cos\psi - n(J_x - J_y + J_z)(\cos\psi)\dot{\phi}$$
$$- n(J_x - J_y - J_z)(\sin\psi)\dot{\theta} - \frac{m}{M+2m}yF + \epsilon F + T_z] \tag{3.11}$$

The Jacobian matrix A can be determined as

$$A = \frac{\partial f(x,u,t)}{\partial x}$$
$$= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ a_{41} & a_{42} & a_{43} & 0 & 0 & a_{46} \\ a_{51} & a_{52} & a_{53} & 0 & 0 & a_{56} \\ 0 & 0 & a_{63} & a_{64} & a_{65} & 0 \end{bmatrix} \tag{3.12}$$

where the coefficients in the Jacobian matrix are defined as

$$a_{41} = -\frac{n^2(J_y - J_z)(3+\cos^2\psi)}{J_x}$$

$$a_{51} = \frac{n^2(J_x - J_z)\cos\psi\sin\psi}{J_y}$$

$$a_{42} = \frac{n^2(J_y - J_z)\cos\psi\sin\psi}{J_x}$$

$$a_{52} = -\frac{n^2(J_x - J_z)(3+\sin^2\psi)}{J_y}$$

$$a_{43} = \frac{n^2(J_y - J_z)\theta\cos^2\psi}{J_x} - \frac{n(J_x - J_y + J_z)\dot{\psi}\sin\psi}{J_x} + \frac{2n^2(J_y - J_z)\phi\cos\psi\sin\psi}{J_x} + \frac{F_s\epsilon\cos\psi\sin\psi}{J_x} - \frac{n^2(J_y - J_z)\theta\sin^2\psi}{J_x}$$

$$a_{53} = \frac{n(J_x - J_y - J_z)\dot{\psi}\cos\psi}{J_y} + \frac{n^2(J_x - J_z)\phi\cos^2\psi}{J_y} + \frac{2F_s mz\cos\psi\sin\psi}{(2m+M)J_y} - \frac{2n^2(J_x - J_z)\theta\cos\psi\sin\psi}{J_y} + \frac{2F_s\epsilon\cos\psi\sin\psi}{J_y} - \frac{n^2(J_x - J_z)\phi\sin^2\psi}{J_y}$$

$$a_{63} = -\frac{n(J_x - J_y - J_z)\dot{\theta}\cos\psi}{J_z} - \frac{n^2(J_y - J_x)\cos^2\psi}{J_z} + \frac{n(J_x - J_y + J_z)\dot{\phi}\sin\psi}{J_z} - \frac{2F_s my\cos\psi\sin\psi}{(2m+M)J_z} + \frac{2F_s\epsilon\cos\psi\sin\psi}{J_z} + \frac{n^2(J_y - J_x)\sin^2\psi}{J_z}$$

$$a_{64} = -\frac{n(J_x - J_y + J_z)\cos\psi}{J_z}$$

$$a_{65} = -\frac{n(J_x - J_y - J_z)\sin\psi}{J_z}$$

$$a_{46} = \frac{n(J_x - J_y + J_z)\cos\psi}{J_x}$$

$$a_{56} = \frac{n(J_x - J_y - J_z)\sin\psi}{J_y}$$

For the T-S fuzzy model, the presence of the trim masses in the moments of inertia make linearization complex. Thus, the inputs for the T-S fuzzy model are T_x, T_y, and

T_z. The control B matrix is

$$B = \frac{\partial f(x,u,t)}{\partial u} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_z} \end{bmatrix} \tag{3.13}$$

3.2 Model Validation

The Takagi-Sugeno fuzzy model used to approximate the solar sail consisted of 28 rules with 2 linearization points each for ϕ and θ and 7 linearization points for ψ. To Validate the T-S fuzzy model of the solar sail, we simulate and compare the open-loop response of the T-S fuzzy model with the numerical solution of Equations (2.7)–(2.9) in Figure 3.3. While the responses lose consistency over time, the pitch and yaw responses almost completely overlap for the hour-long simulation. The T-S fuzzy model is an adequate approximation of the nonlinear solar sail model.

3.3 The Servo Control Problem

The nonlinear reference model can also be described as a T-S fuzzy model [TTYW99a]:

Reference Model Rule k:

IF $z_{R_1}(t)$ is about $\mu_{R_{k1}}[z_{R_1}(t)]$, and $\ldots$, $z_{R_p}(t)$ is about $\mu_{R_{kp}}[z_{R_p}(t)]$, THEN

$$\dot{x}_R(t) = D_k x_R(t) \; ; \quad (k = 1, 2, \ldots, r_R) \tag{3.14}$$

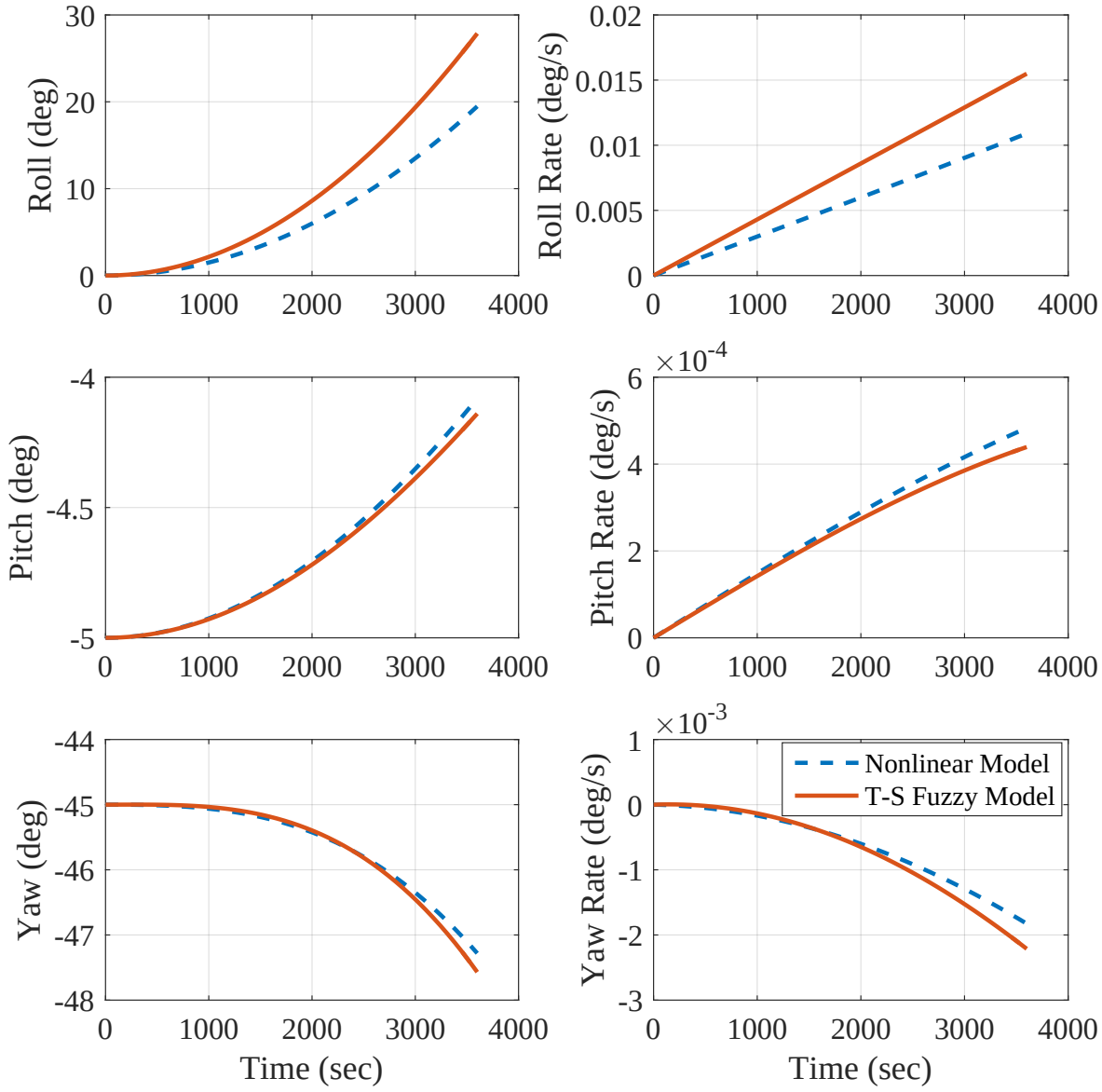


Fig. 3.3: Comparison of the Euler angle and Euler angle rate open loop response for the nonlinear model and the Takagi-Sugeno fuzzy model. The simulation time ran for one hour.

where z_{R_j} are the reference premise variables and can be functions of states, external disturbances, and/or time. $z_R(t)$ is the vector of all the premise variables ($j = 1, \ldots, p_R$), r_R is the number of reference rules, $x_R(t) \in \mathbb{R}^{n_R}$, and $D_k \in \mathbb{R}^{n_R \times n_R}$.

Because the servo control problem consists of a constant input, the T-S fuzzy reference model is comprised of a single rule with the same premise variables as the solar sail T-S fuzzy model. Therefore, the fuzzy reference model can also be called the fuzzy reference input and has the simple form

$$\dot{x}_R(t) = Dx(t) \tag{3.15}$$

Chapter 4

Twin Parallel Distributed Compensation (TPDC)

The Parallel Distributed Compensation (PDC) introduced by Wang et al. [WTG95] provides a procedure to design a fuzzy full-state feedback controller from a given T-S fuzzy model. The PDC regulates the T-S fuzzy model to $x = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$. To stabilize a the system to a nonzero state, a change of variable method can be used when constructing the T-S fuzzy model [BA16a].

To eliminate the need to reconstruct a T-S fuzzy model for every desired state, the Twin Parallel Distributed Compensation (TPDC) technique is chosen as the controller. The TPDC allows for nonlinear reference model following [TTYW99a], but for the purpose of this study a constant linear reference model will be used. The TPDC consists of a primary PDC that has the same form as the PDC studied by Wang et al., but adds a secondary PDC for the reference model. Figure 4.1 shows the process of obtaining the TPDC gains from the reference input and solar sail equations of motion.

4.1 Primary PDC

The T-S fuzzy model is used to construct the primary feedback control law for each model rule. The same premise variables and membership functions are used for the controller and fuzzy model, meaning the same "IF" statements are used but with different "THEN" consequents. The general structure of each control rule is as follows

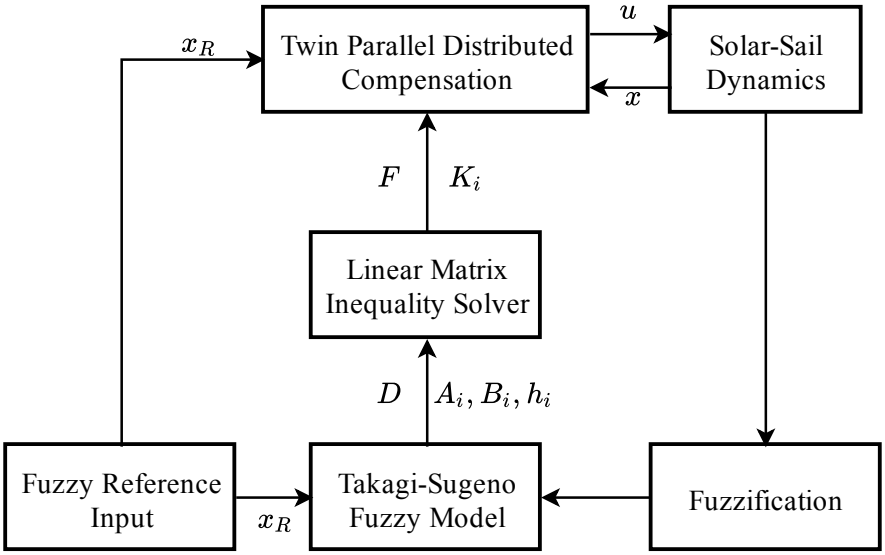


Fig. 4.1: Schematic diagram of the fuzzy system with TPDC.

Control Rule i:

IF $z_1(t)$ is $\mu_{i1}(z_1)$, and $\ldots$, $z_p(t)$ is $\mu_{ip}(z_p)$, THEN

$$u_A(t) = -F_i x(t) \; ; \quad (i = 1, 2, \ldots, r) \tag{4.1}$$

where $F_i \in \mathbb{R}^{m \times n}$ represents the state feedback gain, $x(t)$ denotes the state vector. The primary state feedback controller can be fully written as

$$u_A(t) = -\sum_{i=1}^{r} \frac{w_i[z(t)]}{\sum_{i=1}^{r} w_i[z(t)]} F_i x(t) = -\sum_{i=1}^{r} h_i[z(t)] F_i x(t) \tag{4.2}$$

4.2 Secondary PDC

The secondary PDC that allows for nonlinear reference model following is derived in the same manner as the primary PDC.

Reference Control Rule k:

IF $z_{R_1}(t)$ is about $\mu_{R_{k1}}[z_{R_1}(t)]$, and ..., $z_{R_p}(t)$ is about $\mu_{R_{kp}}[z_{R_p}(t)]$, THEN

$$u_B(t) = -K_k x_R(t) \; ; \quad (k = 1, 2, \ldots, r_R) \tag{4.3}$$

where $K_i \in \mathbb{R}^{m \times n_R}$ represents the state feedback gain and $x_R(t)$ denotes the desired state vector. As previously mentioned, the fuzzy reference model consists of a single rule and is called the fuzzy reference input. The secondary state feedback controller can therefore be fully written as

$$u_B(t) = K x_R \tag{4.4}$$

The total input to the system is simply

$$u(t) = u_A(t) + u_B(t) \tag{4.5}$$

$$= -\sum_{i=1}^{r} h_i[z(t)] F_i x(t) + K x_R \tag{4.6}$$

4.3 The Fuzzy Controller Design Problem

To determine the feedback gain matrices F_i and K, we use the linear matrix inequalities (LMIs) used by Taniguchi et al [TTYW99a]. These LMIs are a relaxed approach to the conservative LMIs previously used by Taniguchi et al. [TTYW99b] and do not require linearization of the error system. The relaxed approach is used to satisfy the condition

$$\frac{1}{2}\{G_{ii} + G_{jj}\} = \frac{1}{2} A_i - B_i F_j + A_j - B_j F_i, \quad \forall j \quad s.t. \quad i < j \leq r \tag{4.7}$$

$$G_{ii} = D_k - B_i K_k, \quad \forall k \tag{4.8}$$

instead of the more conservative condition

$$G = A_i - B_i F_i, \quad \forall i \tag{4.9}$$

$$= \frac{1}{2}(A_i - B_i F_j + A_j - B_j F_i),$$

$$\forall i, j \quad s.t. \quad i < j \leq r \tag{4.10}$$

$$= D_k - B_i K_k \tag{4.11}$$

$$G_{ii} = G_{jj} = \frac{G_{ij} + G_{ji}}{2} \tag{4.12}$$

With the relaxed conditions of Equations (4.7) and (4.8), each G_{ii} matrix can be different. The error system is

$$\dot{e}(t) = \sum_{i=1}^{r} h_i[z(t)] G_{ii} e(t) \tag{4.13}$$

The following LMIs are used to determine the F_i and K_i gains that guarantee the stability of the control system and conditions (4.7) and (4.8):

$$\underset{X, M_i, N_k}{\text{minimize}} \quad \beta$$

$$\text{subject to} \quad \beta > 0, \quad X \geq 0,$$

$$A_i X + X A_i^T - B_i M_i - M_i^T B_i^T \leq 0, \quad \forall i \tag{4.14}$$

$$A_i X + X A_i^T - B_i M_j - M_j^T B_i^T$$

$$+ (A_j X + X A_j^T - B_j M_i - M_i^T B_j^T)^T \leq 0,$$

$$\forall i, j \quad s.t. \quad i < j \tag{4.15}$$

$$\begin{bmatrix} 4\beta I & & \\ \begin{pmatrix} A_iX - B_iM_i + A_jX - B_jM_j \\ -(A_iX - B_iM_j + A_jX - B_jM_i) \end{pmatrix} & & \\ & \begin{pmatrix} A_iX - B_iM_i + A_jX - B_jM_j \\ -(A_iX - B_iM_j + A_jX - B_jM_i) \end{pmatrix}^T \\ & I \end{bmatrix} \geq 0,$$

$$\forall j \quad s.t. \quad i < j \leq r \tag{4.16}$$

$$\begin{bmatrix} \beta I & \\ \{A_iX - B_iM_i - (D_kX - B_iN_k)\} & \\ & \{A_iX - B_iM_i - (D_kX - B_iN_k)\}^T \\ & I \end{bmatrix} \geq 0,$$

$$(k = 1, 2, \ldots, r_r) \tag{4.17}$$

where $F_i = M_iX^{-1}$ and $K_k = N_kX^{-1}$.

Amplitude constraints are also considered during the construction of F_i and K_k using the following LMI:

$$\begin{bmatrix} X & 0 & M_i^T \\ 0 & X & -N_k^T \\ M_i & -N_k & \mu^2 I \end{bmatrix} \geq 0 \tag{4.18}$$

If Equation (4.18) holds true, then

$$||u(t)||_2 \leq \mu \left|\left| \begin{bmatrix} x(t) \\ x_R(t) \end{bmatrix} \right|\right|_2, \quad t \geq 0 \tag{4.19}$$

is enforced at all times $t \geq 0$.

4.4 TPDC Numerical Simulation

Matlab/Simulink is used to simulate the response of the solar sail and CVX [CR12] is used to solve the system of linear matrix inequalities and Matlab. Table 4.1 lists the values used in the simulation of the solar sail in Simulink. All parameters are for a 40 m scalable solar sail being developed by ATK Space Systems [Wie08]. A sail of this size is capable of the low-Earth sun-synchronous orbit that is studied in this paper.

Table 4.1: Geometric, mass, and inertia properties of the solar sail used for the simulation [Wie08].

Parameter	Variable	Value
Sail Size, m	L	40
Mast Length, m	y_{max}, z_{max}	28
Sail Area, m^2	A	1200
Max Thrust, N	F_s	0.01
cm/cp Offset, m	ϵ	0.1
Trim Mass, kg	m	1
Sailcraft Mass, kg	M	148
Total Mass, kg	M_{tot}	150
Roll Moment, kg·m^2	I_x	4340
Pitch Moment, kg·m^2	I_y	2171
Yaw Moment, kg·m^2	I_z	2171
Orbital Rate, rad/s	n	6.311×10^{-5}

The reference model was chosen to be a constant input $x_R(t) = c$. When c is a nonzero vector, this is known as the servo control problem. When $c = 0$ it is called the regulator control problem. For both cases, T-S fuzzy reference model—and consequently the secondary PDC—consist of a single rule $r_R = 1$. The state matrix for the T-S fuzzy

reference model is thus

$$D = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.20}$$

The attitude maneuver chosen for this study was a reorientation from full-thrust mode to an orbit raising orientation. Small initial roll and pitch angles were also selected so that $x_0 = [\ 5^\circ \quad -5^\circ \quad -90^\circ \quad 0 \quad 0 \quad 0\ ]$. From these initial conditions, small roll and pitch maneuvers with a relatively large yaw maneuver were executed. The desired states were $x_R = [\ 0^\circ \quad 0^\circ \quad -55^\circ \quad 0 \quad 0 \quad 0\ ]$. The control inputs T_x, T_y, and T_z acted as the primary inputs and the trim masses were set at $z = -14.9$ m and $y = 14.9$ m to offset the pitch and yaw disturbances.

Table 4.2: PID gains obtained through the Ziegler-Nichols method.

	K_p	$K_i \times 10^{-6}$	K_d
T_x	0.006	2.487	3.620
T_y	0.006	3.838	2.345
T_z	0.006	2.422	3.716

The Euler angles, Euler angle rates, and controller inputs are shown in Figures 4.2 and 4.3. The TPDC and PID controller are both used to control the nonlinear system. Figure 4.2 shows the TPDC is able to stabilize the solar sail much quicker and with less overshoot and oscillation than the PID controller. The difference in initial input values shown in Figure **??** comes from the different gains between the TPDC and PID controllers and the fact that actuator dynamics are not included.

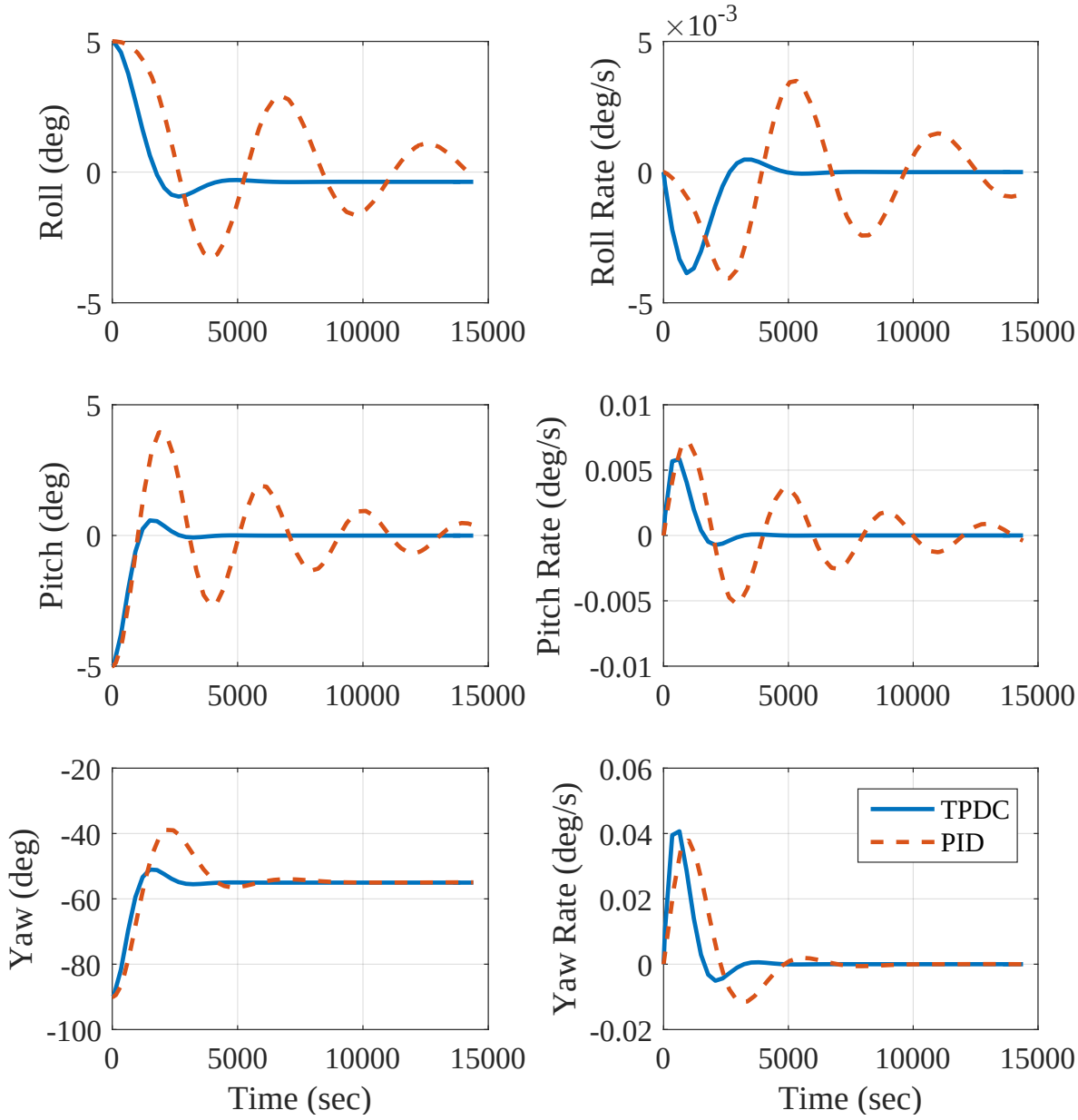


Fig. 4.2: Euler angle and Euler angle rate comparison of the TPDC and PID controllers in moving the solar sail from a full-thrust position to an orbit raising position.

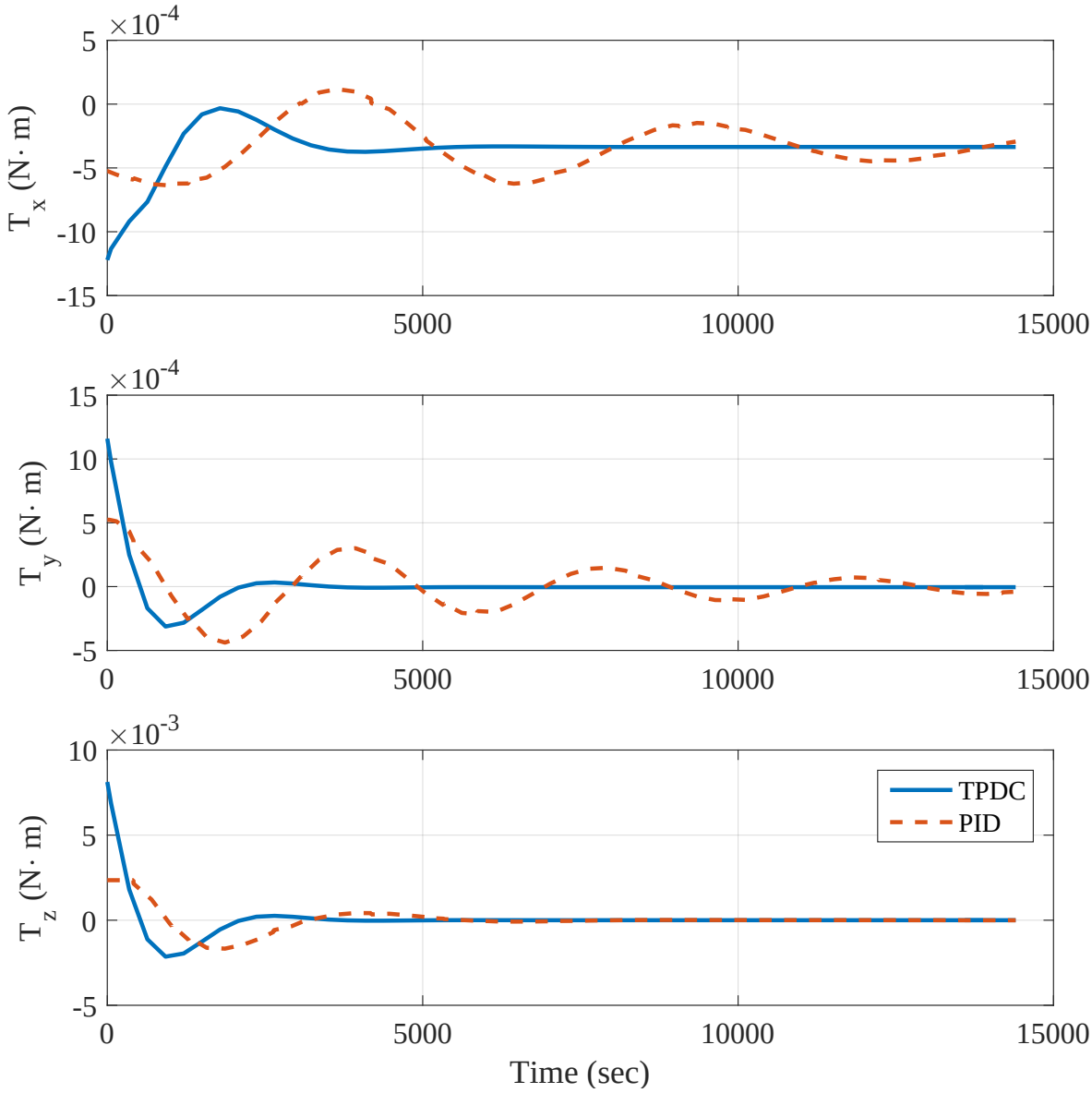


Fig. 4.3: Comparison of the TPDC and PID controller inputs in moving the solar sail from a full-thrust position to an orbit raising position.

Table 4.3: Uncertainty in the nominal values of the principle moments of inertia I_x^*, I_y^*, and I_z^*.

Case	I_x, kg·m^2	I_y, kg·m^2	I_z, kg·m^2
I	$0.8I_x^*$	$0.4I_y^*$	$0.3I_z^*$
II	$1.6I_x^*$	$0.8I_y^*$	$0.6I_z^*$
III	$2.4I_x^*$	$1.2I_y^*$	$0.9I_z^*$
IV	$3.2I_x^*$	$1.6I_y^*$	$1.2I_z^*$

4.4.1 TPDC Robustness Results

We add variation to the moments of inertia I_x, I_y, and I_z to test the robustness of both the TPDC controllers to parameter uncertainties in the system. Table 4.3 shows the uncertainties in the moments of inertia that were used by Baculi and Ayoubi [BA16a] to test their PDC controller. This test also had initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[0 0 -55° 0 0 0]. Figure 4.4 shows the TPDC is capable of stabilizing the solar sail to the desired state in the presence of model uncertainties.

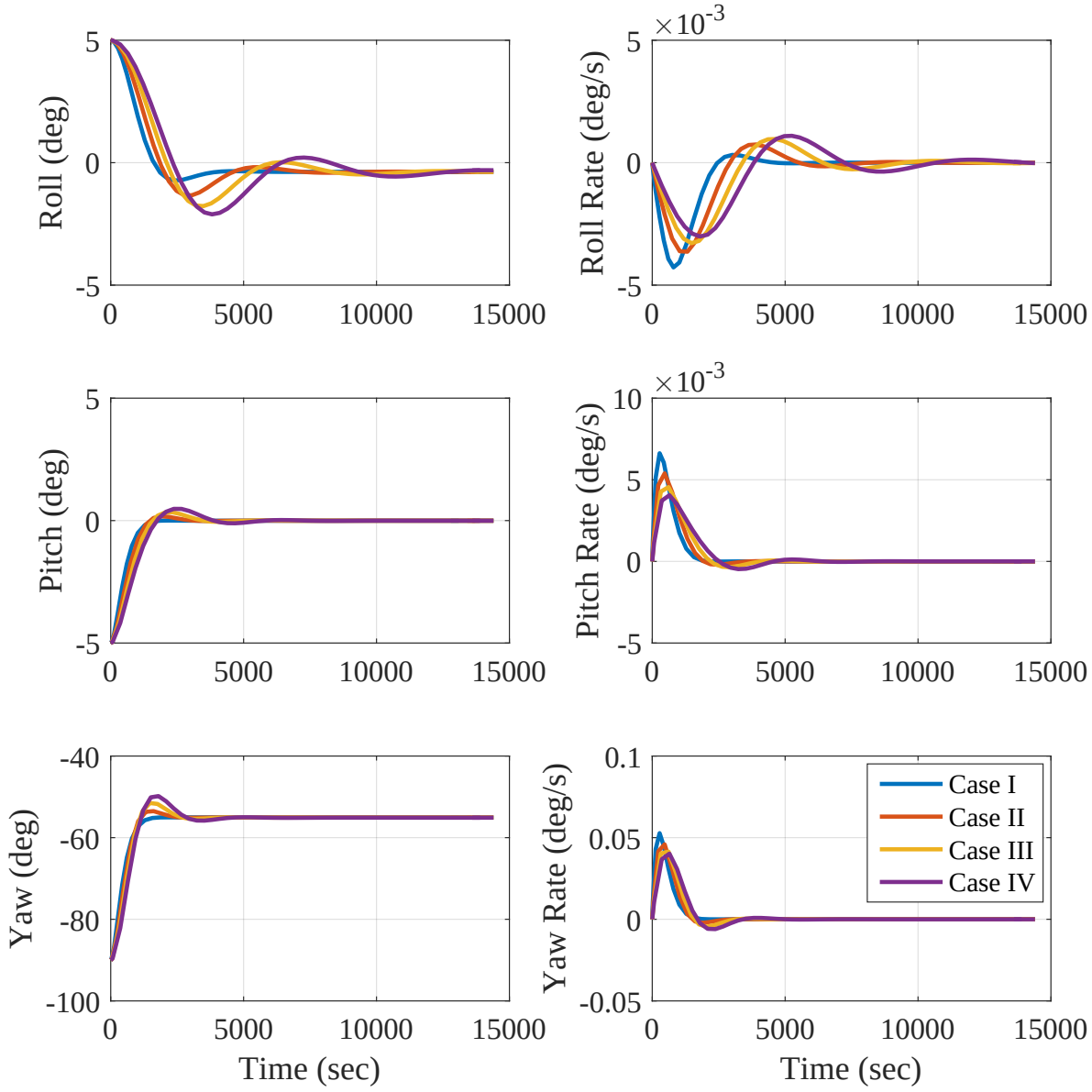


Fig. 4.4: Robustness of the TPDC controller to the uncertainties in the moments of inertia with initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[0 0 -55° 0 0 0].

Chapter 5

Supervisory-Fuzzy-PID Controller Design

Figure 5.1 shows the schematic diagram of the FLS-PID controllers implemented on the solar sail. The PID controllers require nine gains, three gains for each of the three torque inputs. Optimization procedures are conducted to obtain gains that minimize error about certain operating points. Particle swarm optimization (PSO) is used in this study to optimize the proportional, integral, and derivative gains by minimizing transient error. PSO is a stochastic optimization technique first proposed by Kennedy and Eberhart [KE95] in 1995. This technique was inspired by bird flocks that continuously dance between moving synchronously and scattering, seemingly at random. By having the particles of the optimizer—in this case the PID gains—move randomly through space, a cost function—in this case the error—can be minimized. These optimal PID gains act as the maximum gain values used by FLS. The minimum values are obtained using the Ziegler-Nichols tuning method.

While the T-S fuzzy model did not yield a good approximation of the solar sail when the trim masses are inputs, the PID controllers work on the nonlinear system and so losing model accuracy with the trim masses as inputs are not a concern. The PID controllers

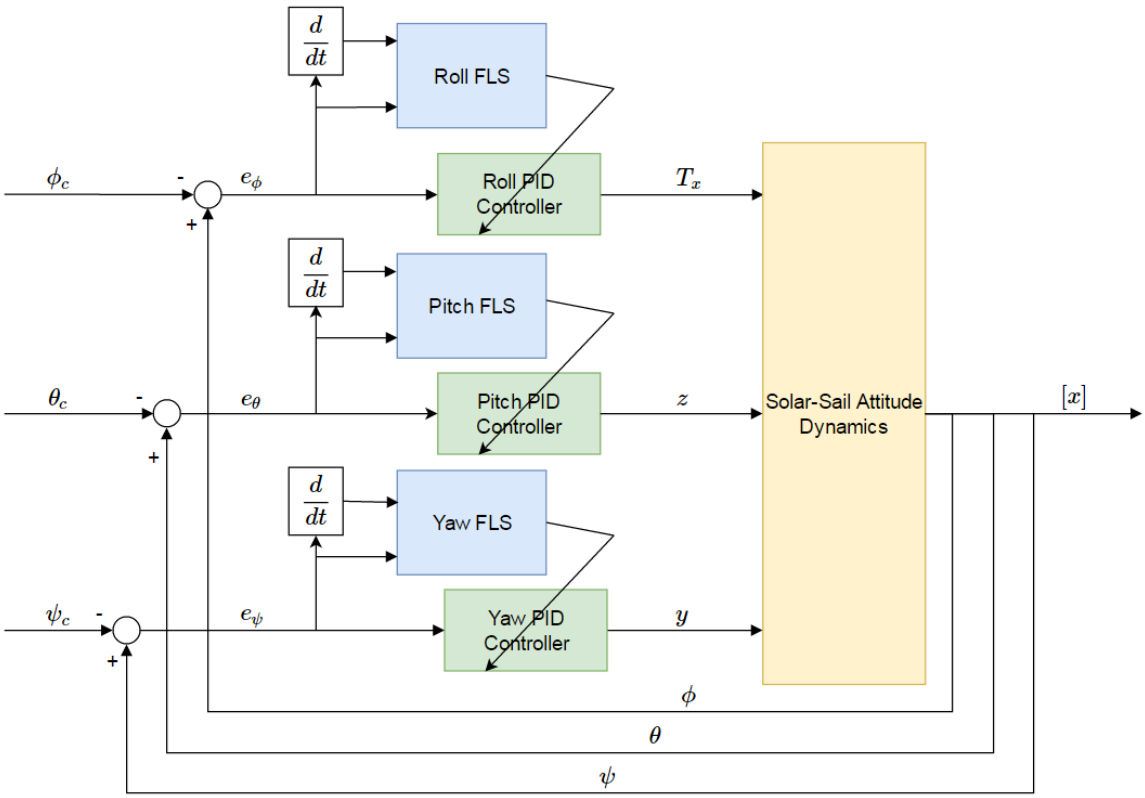


Fig. 5.1: Block diagram of the fuzzy logic supervisor with the PID controllers.

have the following forms

$$u_{T_x} = -\left[K_{p_{T_x}}(\phi - \phi_c) + K_{i_{T_x}}\int(\phi - \phi_c)dt + K_{d_{T_x}}\frac{d}{dt}(\phi - \phi_c)\right] \tag{5.1}$$

$$u_z = -\left[K_{p_z}(\theta - \theta_c) + K_{i_z}\int(\theta - \theta_c)dt + K_{d_z}\frac{d}{dt}(\theta - \theta_c)\right] \tag{5.2}$$

$$u_y = \left[K_{p_y}(\psi - \psi_c) + K_{i_y}\int(\psi - \psi_c)dt + K_{d_y}\frac{d}{dt}(\psi - \psi_c)\right] \tag{5.3}$$

where the subscript "c" denotes the commanded attitude. The gains for the simple PID controller without an FLS are optimized using particle swarm optimization about an initial condition and desired state. The gains are adapted using the fuzzy logic supervisor, as shown in Figure 5.1. The supervisor utilizes T_x, z, and y as the main attitude control input. The FLS-PID controller is implemented to tune the gains depending on the error and error rate. Varying the PID gains during the closed-loop response improves the control authority of the trim masses at small yaw angles and increases the robustness of the PID controller. The design of the FLS and PID controller can be done in the following steps:

1. Build a simple PID controller for the solar sail with inputs T_x, z, and y. There should be three separate controllers—one each for roll, pitch, and yaw. The PID gains can be found using the Ziegler-Nichols (Z-N) method for specific initial condition, x_0, and commanded input, x_c.

2. Use the particle swarm optimization technique to minimize a cost function and obtain the optimal gains for the same x_0 and x_c.

3. Repeat Steps 1 and 2 for different x_0 and x_c.

4. Build the Fuzzy Logic Supervisor with the calculated gains and initial and desired states.

5.1 Particle Swarm Optimization (PSO)

The Fuzzy Logic Supervisor requires a range of PID gains over which the membership functions can interpolate gains. This range of gains was determined using Ziegler-Nichols tuning for the min values and Particle Swarm Optimization (PSO) for the max values. A PSO algorithm was developed to calculate the optimal gains of the solar sail to two different reference states from the same initial conditions.

Table 5.1: PID gains calculation using the closed loop Ziegler-Nichols method.

Controller	K_p	K_i	K_d
PID	$0.6Ku$	$2K_p/P_u$	$K_pP_u/8$

The gains for the controller are first obtained heuristically using the Ziegler-Nichols method shown in Table 5.1. This method is chosen to provide initial guesses for the gains to be used in PSO. With the Z-N gains calculated, they are used as the first "current position" in the PSO algorithm. Therefore, the swarm operates in 9-dimensional space (3 gains $\times$ 3 inputs $= 9 = D$). The PSO algorithm can be executed with the following steps.

1. Select parameters for swarm size n, number of steps *step*, cognitive component c_1 social component c_2, and inertia w [TA12].

2. Calculate the fitness of each particle and determine the personal best fitness *pbest* of each particle. Because fitness in this study is a cost function, *pbest* is the minimum cost function between the particle's current fitness and its fitness from the previous step.

3. Find global best fitness *gbest* of the swarm and, if necessary, update the swarm using the following equations [TA12]

$$v_j^{i+1} = wv_j^i + c_1r_1(pbest_j - p_j^i) + c_2r_2(gbest_j - p_j^i) \tag{5.4}$$

$$p_j^{i+1} = v_j^{i+1} + p_j^i \tag{5.5}$$

where r_1 and r_2 are normalized random matrices of size $D \times n$, $i = 1 \ldots step$, and $j = 1 \ldots n$.

4. Repeat from step 2 until $i = step$.

Note that for the first iteration, the velocity is assumed to be

$$v_j^1 = 10 p_j^1 r_0$$

where r_0 is another normalized random matrix of size $D \times n$.

We choose our cost function as the Integral-of-Time-multiplied-by-Absolute-Error (ITAE) which is defined as

$$ITAE_k = \int_0^\infty t|e_k(t)|dt; \quad (k = \phi, \theta, \psi) \tag{5.6}$$

as the fitness index due to its performance compared to other functions in the study conducted by Solihin et al [STK11]. The ITAE outperformed the integral of the square of the error (ISE), integral of the absolute magnitude of the error (IAE), and integral of time multiplied by squared error (ITSE) cost functions in settling time and yielded the second smallest overshoot [STK11]. Due to the three PID controllers for the three responses shown in Figure 5.1, three cost functions are calculated for each particle (roll, pitch, and yaw). The fitness of the particle can be found by simply taking the average of the three ITAE cost functions.

Table 5.2: Effects of increasing PID gains on closed-loop system response.

	Rise Time	Overshoot	Settling Time	Steady-State Error
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Small Change	Decrease	Decrease	No Change

5.2 Designing Fuzzy-Logic Supervisor (FLS)

The fuzzy logic supervisor shown in Figure 5.1 is used to change the gains of the PID controllers based on the error and error rate. Table 5.2 shows the effects of increasing each gain on the system's closed-loop response. For example, increasing the proportional gain decreases the rise time, meaning the system responds quicker, but also increases the overshoot. Conversely, increasing the derivative gain decreases the overshoot without causing significant effects on the rise time. If rise time and overshoot were the primary concerns when designing a PID controller, one may ask why simply having large gains would not suffice. While Table 5.2 shows the effects on the response, tuning gains also influences the effects of the other gains. Because of this, the supervisor is implemented to improve performance.

Figure 5.2 breaks down the closed-loop response of a system to a step input. Region 1 consists of the response with a positive error and negative error rate. During this region, a high proportional gain is desirable to decrease rise time. Through Region 2 while the system is overshooting, the error is now negative while the error rate remains positive. A smaller proportional gain with a larger derivative gain would decrease the overshoot over Region 2 as well as Region 3.

The FLS is implemented using a similar approach as the Takagi-Sugeno (T-S) fuzzy model implemented by Tanaka and Wang [TW01]. However, because the rule base heuristic, i.e. the gain is either "Big" or "Small", the FLS is not a T-S fuzzy model. The FLS is designed as follows:

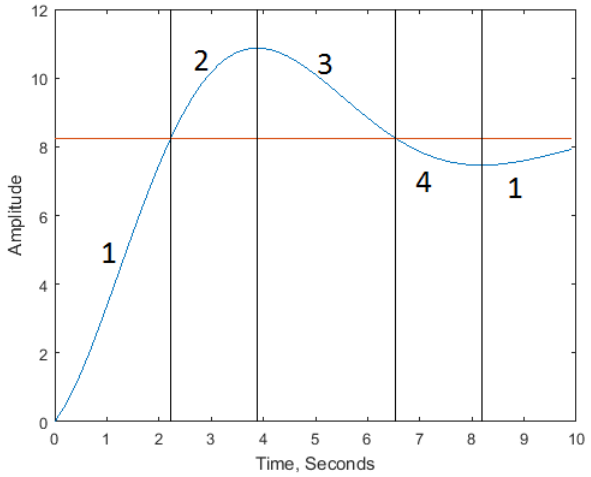


Fig. 5.2: Underdamped step response showing the four regions of error and error rate signs.

Model Rule i for each FLS:

IF e is about $\mu_{i1}[e]$ and $\dot{e}$ is about $\mu_{i2}[\dot{e}]$, THEN

$$K(t) = K_i(t); \quad (i = 1, 2, \ldots, r) \tag{5.7}$$

where μ_i is the membership function for each rule, and $K_i \in \mathbb{R}^{3\times 1}$ is the PID gain vector for the FLS. In this case, the premise variables are error (e) and error rate ($\dot{e}$). The firing strength of each rule can be determined using a $T-norm$ product as follows

$$w_i[z(t)] = \prod_{j=1}^{p} \mu_{ij}[z_j(t)]; \quad 0 \leq \mu_{ij}(z_j) \leq 1 \tag{5.8}$$

and the fuzzy basis functions are determined from

$$h_i[z(t)] = \frac{w_i[z(t)]}{\sum_{i=1}^{r} w_i[z(t)]} \tag{5.9}$$

Table 5.3: Fuzzy control matrices for K_p supervisor. "B" denotes "Big" and "S" denotes "Small".

K_p		e		
		Negative	Zero	Positive
$\dot{e}$	Negative	B	S	B
	Zero	B	B	B
	Positive	B	S	B

where $z(t) = [e \quad \dot{e}]$, and the fuzzy basis functions, $h_i[z(t)]$, have the following properties:

$$\sum_{i=1}^{r} h_i[z(t)] = 1, \qquad h_i[z(t)] \geq 0; \quad (i = 1, \ldots, r) \tag{5.10}$$

After combining the rules for the fuzzy models, the final gain values are found to be

$$K(t) = \sum_{i=1}^{r} h_i[z(t)] K_i(t) \tag{5.11}$$

Work has been conducted to optimize the membership functions and rules of the FLS [EG12], but was not done in this study. Instead, a simplified version of the Macvicar-Whlean matrix was chosen to build the fuzzy rule sets [MW76]. The simplified fuzzy rule sets for the PID gains are shown in Tables 5.3–5.5. The Macvicar-Whelan matrix actually consists of 81 rules (the two premise variables with nine membership functions per premise variable). Due to the solar sail being a MIMO system consisting of three PID controllers, three membership functions per premise variable were chosen. Therefore, the fuzzy supervisor consists of 27 rules total; i.e. $r = 27$. The "Small" and "Big" values of the PID gains K are determined using Ziegler-Nichols tuning and particle swarm optimization, respectively. Utilizing a heuristically tuned set of gains with optimal gains broadens the range of possible gains, increasing robustness. Figure 5.3 shows the triangular membership functions utilized by the premise variables.

The Negative and Positive values of the error premise variable are determined by the

Table 5.4: Fuzzy control matrix for K_i supervisor. "B" denotes "Big" and "S" denotes "Small".

K_i		e		
		Negative	Zero	Positive
$\dot{e}$	Negative	S	B	S
	Zero	S	B	S
	Positive	S	B	S

Table 5.5: Fuzzy control matrix for K_d supervisor. "B" denotes "Big" and "S" denotes "Small".

K_d		e		
		Negative	Zero	Positive
$\dot{e}$	Negative	S	B	S
	Zero	S	B	S
	Positive	S	B	S

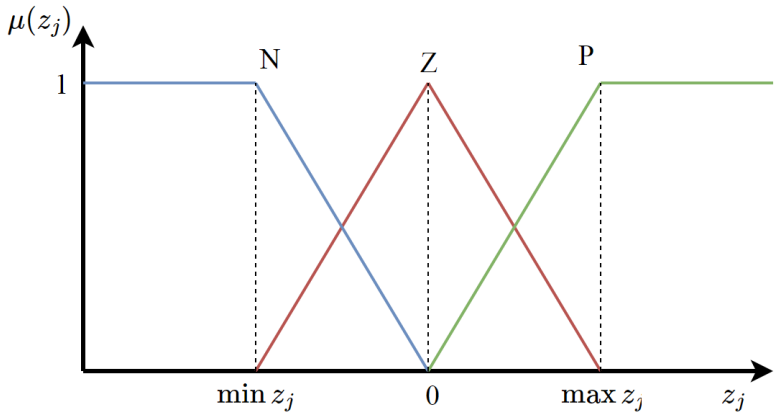


Fig. 5.3: Membership function of premise variables. "N", "Z", and "P" denote "Negative", "Zero", and "Positive", respectively.

initial conditions x_0 and desired states x_c

$$\text{Negative} = -|x_0 - x_c| \tag{5.12}$$

$$\text{Postive} = |x_0 - x_c|. \tag{5.13}$$

For the error rate, the Negative and Positive values were simply ± 0.05 deg/s, which is the max angular rate allowed for the solar sail [Wie08].

5.3 FLS Numerical Simulation

The same solar sail parameters shown in Table 4.1 are used to test the closed loop performance of the FLS-PID controller. Table 5.6 shows the parameter values used for particle swarm optimization of the PID gains. A swarm size of 10–50 was stated to provide "reasonably similar performance" [STK11]. Values for c_1, c_2, and w were recommended for convergence by Clerc [Cle99]. The number of steps were chosen after numerous tests with a varying number of steps.

Table 5.6: Parameters used for particle swarm optimization.[STK11, Cle99]

Parameter	Variable	Value
Swarm Size	n	50
Number of Steps	*step*	150
Cognitive Component	c_1	1.494
Social Component	c_2	1.494
Inertia	w	0.729

Two sets of gains were calculated to comprise the range of gains for the FLS and were in the form $K = [K_{p_{T_x}} \quad K_{d_{T_x}} \quad K_{i_{T_x}} \quad K_{p_z} \quad K_{d_z} \quad K_{i_z} \quad K_{p_y} \quad K_{d_y} \quad K_{i_y}]$ The first set of gains, K_1, were obtained using PSO with x_0=[5° -5° -90° 0 0 0] and x_c=[0 0 -15° 0 0 0] deg. The initial state was chosen to simulate a maneuver from a full-thrust orientation

Table 5.7: Values of the gains used for the Fuzzy Logic Supervisor PID controller as well as the comparison PSO-PID controller.

Gain	$K_1 \times 10^6$	$K_2 \times 10^4$	$K_{PSO} \times 10^5$
$K_{p_{T_x}}$	0.000000110870580	0.000000600000000	0.000000956294894
$K_{d_{T_x}}$	0.000032486945066	0.000345600000000	0.000259318792317
$K_{i_{T_x}}$	0.000000000003033	0.000000000260417	0.000000000075565
K_{p_z}	0.001003578091345	0.003000000000000	0.011655112126712
K_{d_z}	0.480749420434239	4.320000000000000	3.839189494542947
K_{i_z}	0.000000147802548	0.000000520833333	0.000000639451501
K_{p_y}	0.000434177085802	0.003000000000000	0.004456965998984
K_{d_y}	1.190067849066839	7.560000000000000	5.517868067201267
K_{i_y}	-0.000000001205386	0.000000297619048	-0.000000195023569

similar to the one shown in Figure 2.2. The second set of gains, K_2, that provided an acceptable range of gains were determined using the Ziegler-Nichols method. The second set of gains were tuned to x_0=[5° -5° -90° 0 0 0] and x_c=[0 0 -75° 0 0 0]. Table 5.7 shows the values of K_1 and K_2 pertaining to each PID gain that must be sorted to obtain $K_{\min}$ and $K_{\max}$. Also shown in Table 5.7 are the gains K_{PSO} used for a comparison PSO-PID controller. This PSO-PID controller was optimized for x_0=[5° -5° -90° 0 0 0] and x_c=[0 0 -55° 0 0 0]. The supervisory PID controller was tested by comparing its response to PSO-PID controller.

For the responses shown in Figures 5.4 and 5.5, the initial condition was x_0=[5° -5° -90° 0 0 0] with commanded state x_c=[0 0 -55° 0 0 0]. A yaw value of -55° puts the solar sail in an attitude for orbit raising[Wie08]. Such a maneuver would move the solar sail from a full-thrust position to one where the solar sail's orbital radius will increase. Figure 5.4 shows the supervisory PID controller (FLS-PID) is outperformed by the optimized PID controller (PSO-PID) in pitch and yaw. This type of behavior is to be expected as the PSO-PID is optimized for these conditions whereas the FLS-PID interpolates between optimal gains. Figure 5.5 shows the control inputs for roll, pitch, and yaw.

To showcase the performance of the FLS-PID controller, a commanded input different

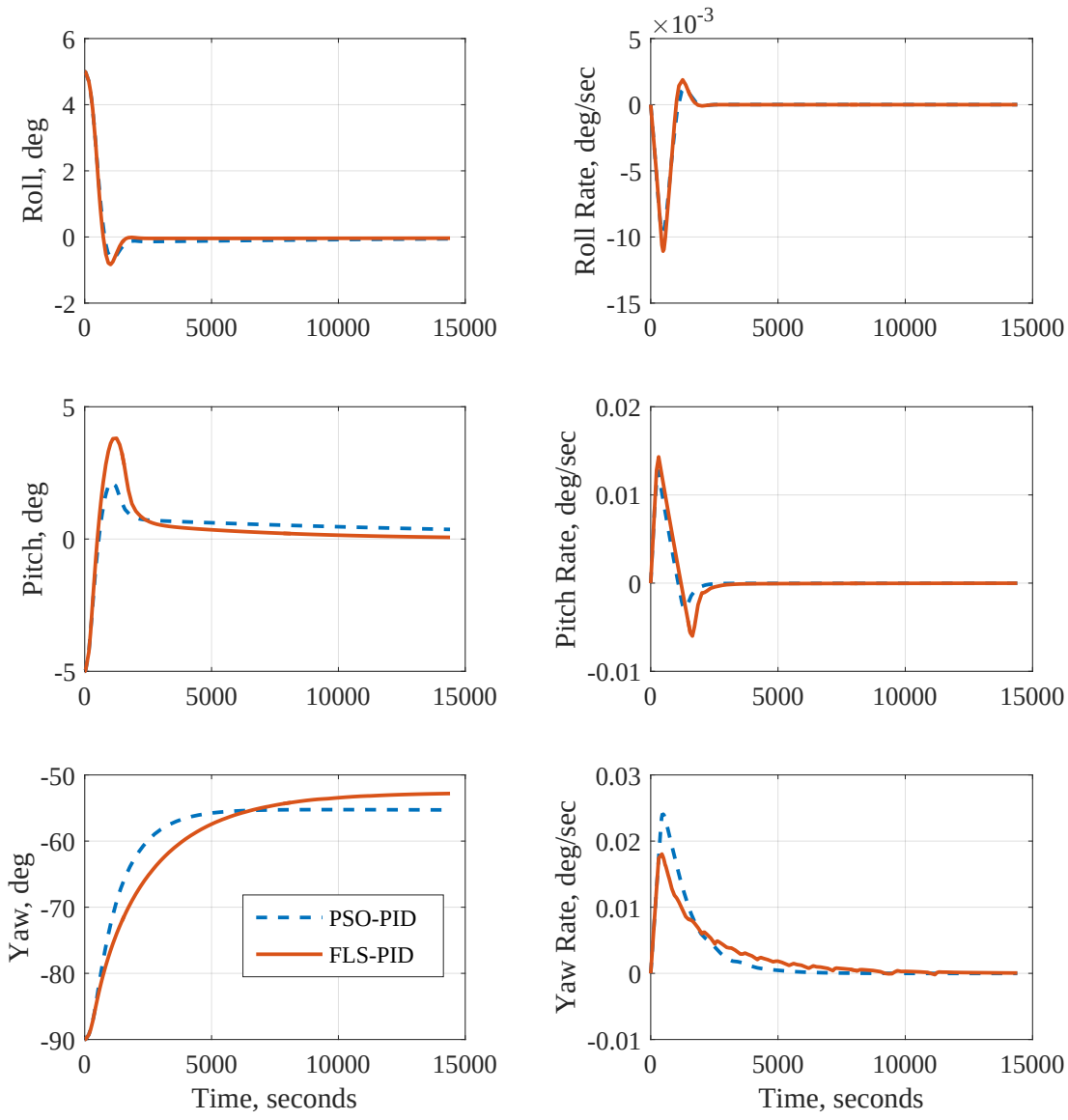


Fig. 5.4: Comparison of the closed-loop responses of the PSO-PID and the FLS-PID controllers with initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[0 0 -55° 0 0 0].

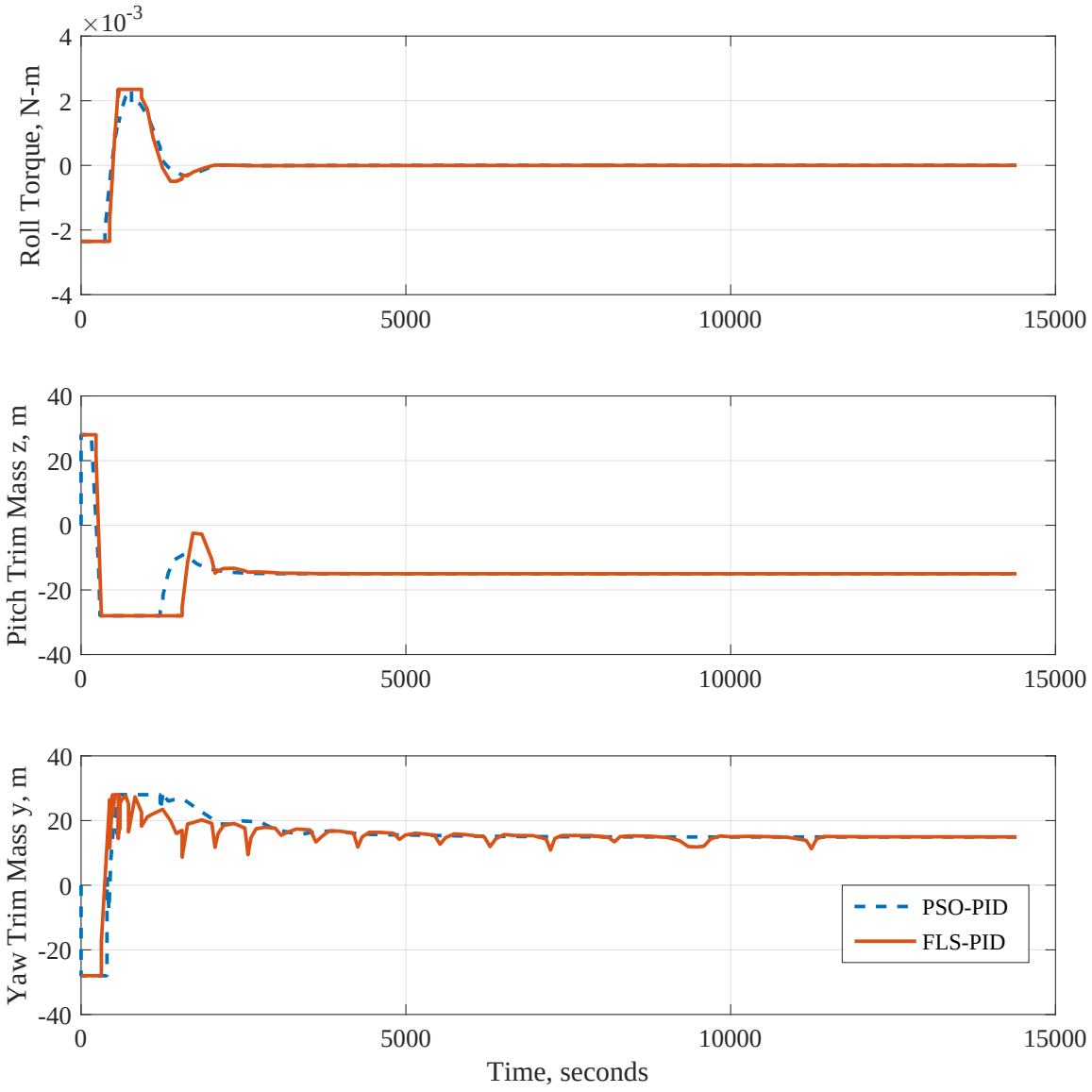


Fig. 5.5: Control inputs of the PSO-PID and the FLS-PID controllers with initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[0 0 -55° 0 0 0].

than the x_c used to optimize K_{PSO} was chosen. Figure 5.6 shows the responses to x_c=[-2° 2° -35° 0 0 0] and Figure 5.7 shows the control inputs. With the PSO-PID moving the solar sail to a state it was not optimized for, the FLS-PID outperforms the PSO-PID for pitch and yaw. Although the FLS-PID has a greater overshoot than the PSO-PID for pitch, the FLS-PID settles much quicker. With the yaw command 20° away from the PSO-PID's optimized commanded state, it does not even stabilize to -35°.

To examine the robustness of the FLS-PID controller, we change the initial conditions from the x_0 used to optimize K_{PSO}. The maneuver simulated in Figure 5.8 moves the yaw angle from -55° to -90°, which would reorient the sail from an orbit raising orientation to full-thrust mode. The roll response for both controllers match up well, but the PSO-PID fails to reach the commanded pitch and yaw angles. Although it does not completely settle in the alotted time, the FLS-PID brings both pitch and yaw towards the commanded inputs.

5.3.1 FLS Robustness Results

Variation to the moments of inertia is added to the solar sail model to also test the robustness of the FLS-PID controller. The same variations shown in Table 4.3 are used here. The same variations were again used by Baculi and Ayoubi in testing their FLS-PID controller [BA16b]. This test also had initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[0 0 -55° 0 0 0]. The FLS-PID is also shown in Figure 5.10 to stabilize at x_c even with variations in the moments of inertia.

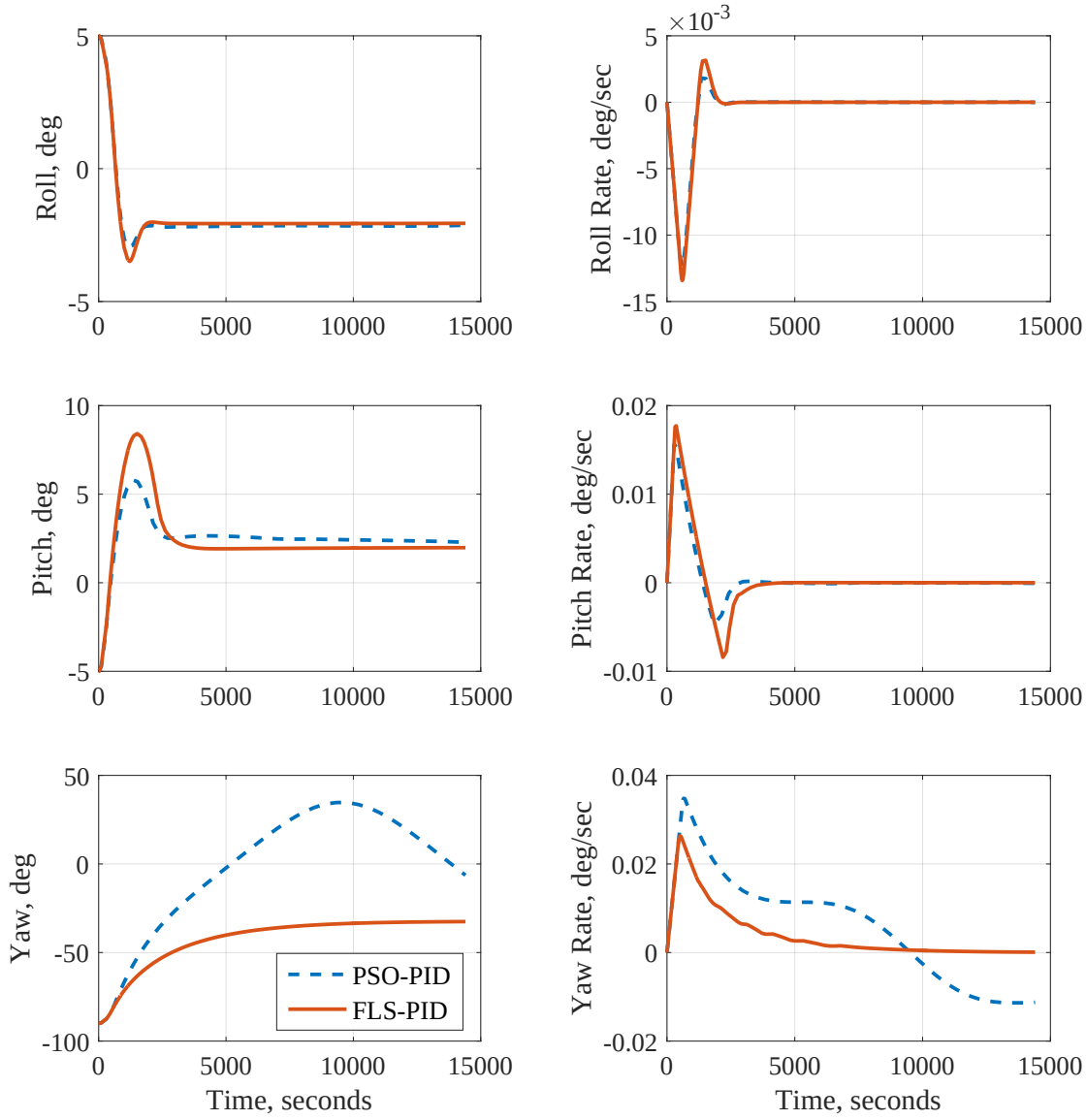


Fig. 5.6: Robustness comparison of the closed-loop responses of the PSO-PID and the FLS-PID controllers with initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[-2° 2° -35° 0 0 0].

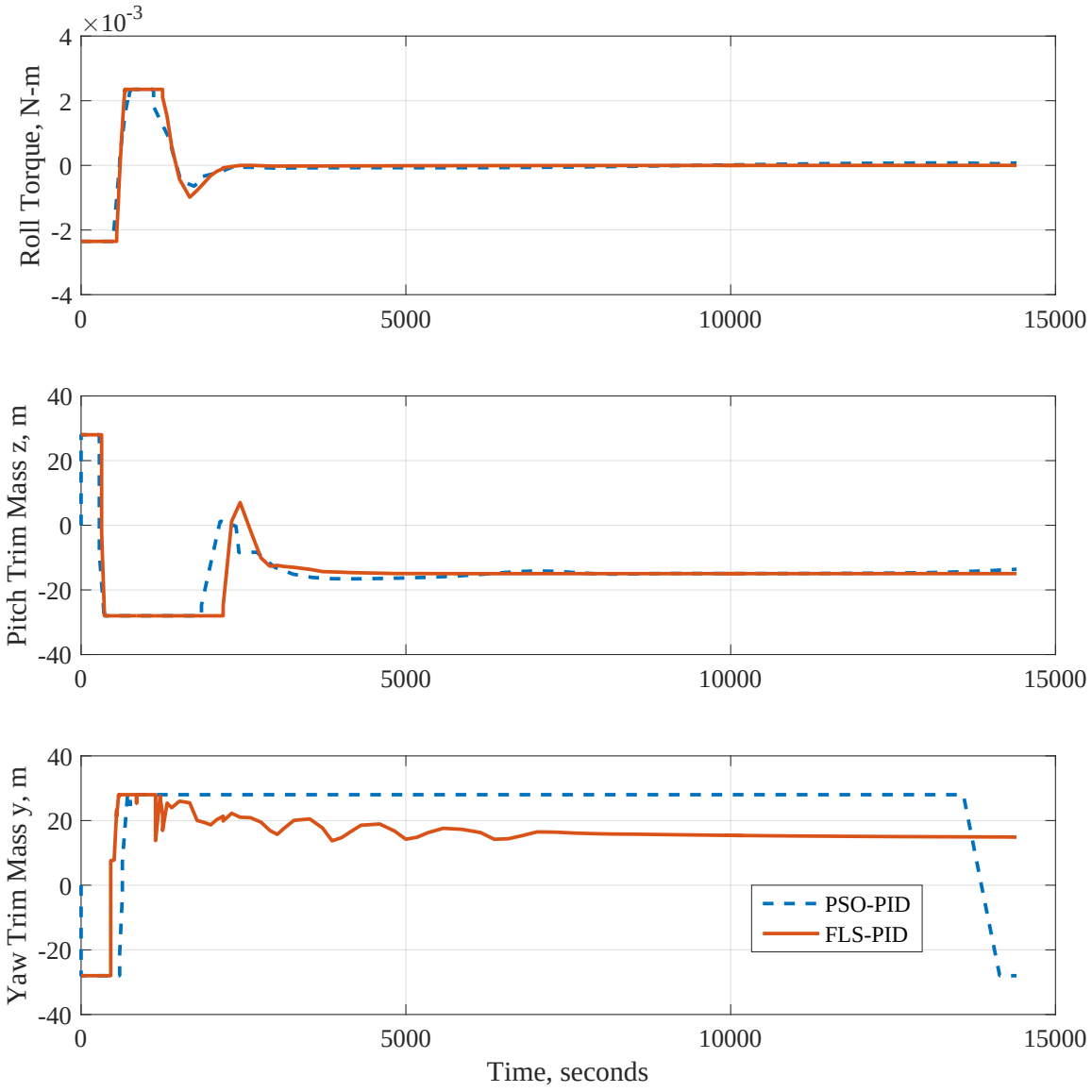


Fig. 5.7: Control inputs of the PSO-PID and the FLS-PID controllers with initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[-2° 2° -35° 0 0 0].

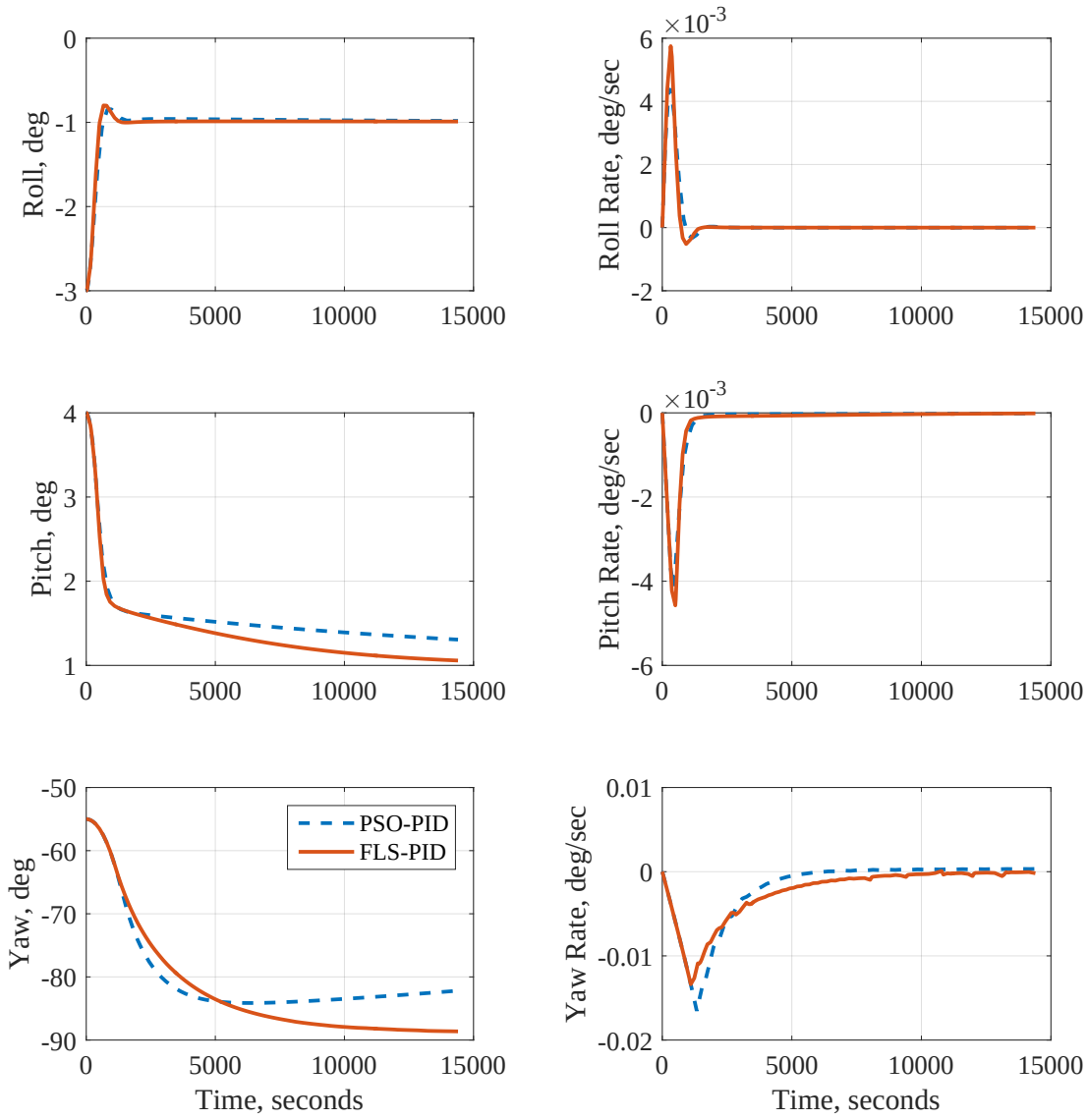


Fig. 5.8: Robustness comparison of the closed-loop responses of the PSO-PID and the FLS-PID controllers with initial condition x_0=[-3° 4° -55° 0 0 0] and commanded input x_c=[-1° 1° -90° 0 0 0].

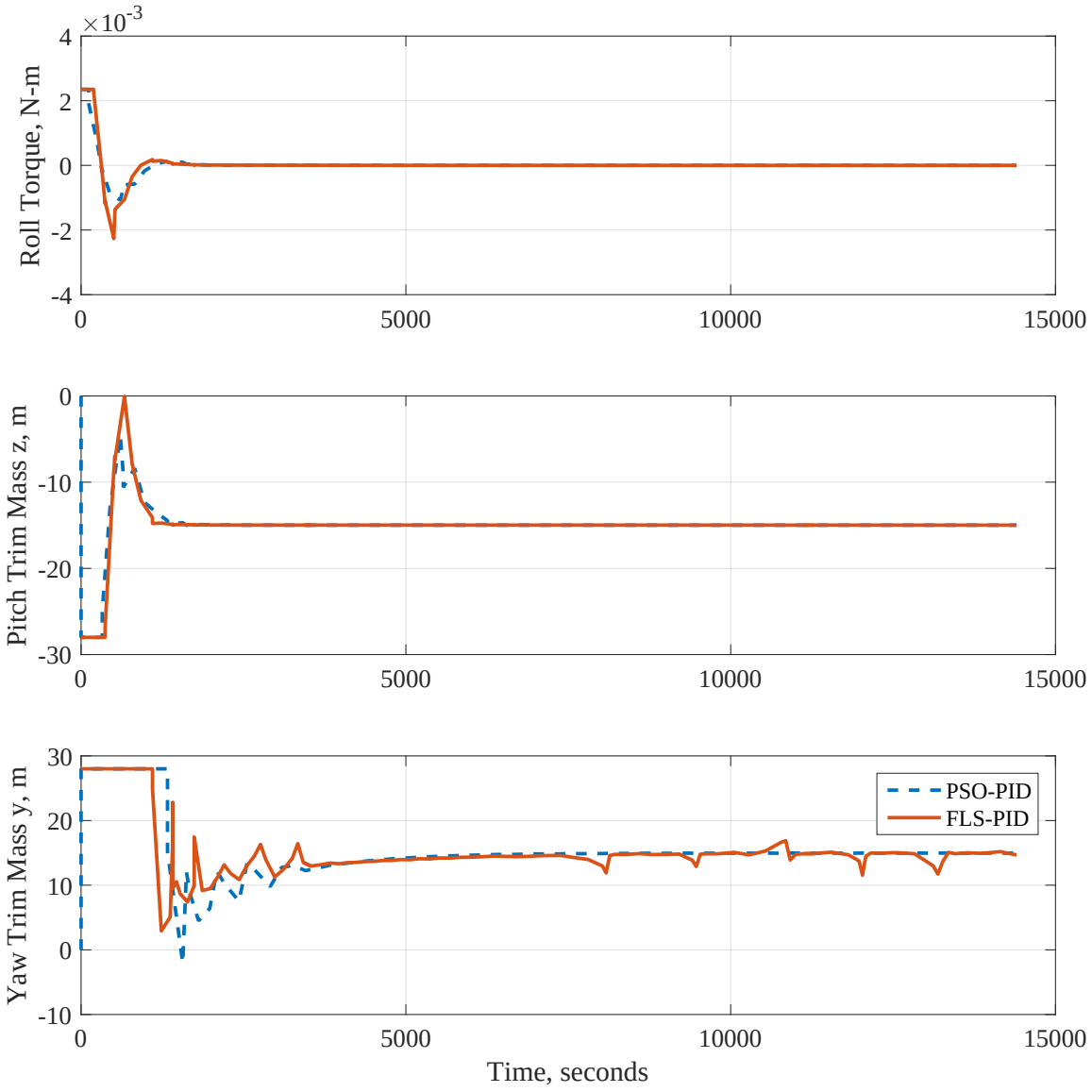


Fig. 5.9: Control inputs of the PSO-PID and the FLS-PID controllers with initial condition x_0=[-3° 4° -55° 0 0 0] and commanded input x_c=[-1° 1° -90° 0 0 0].

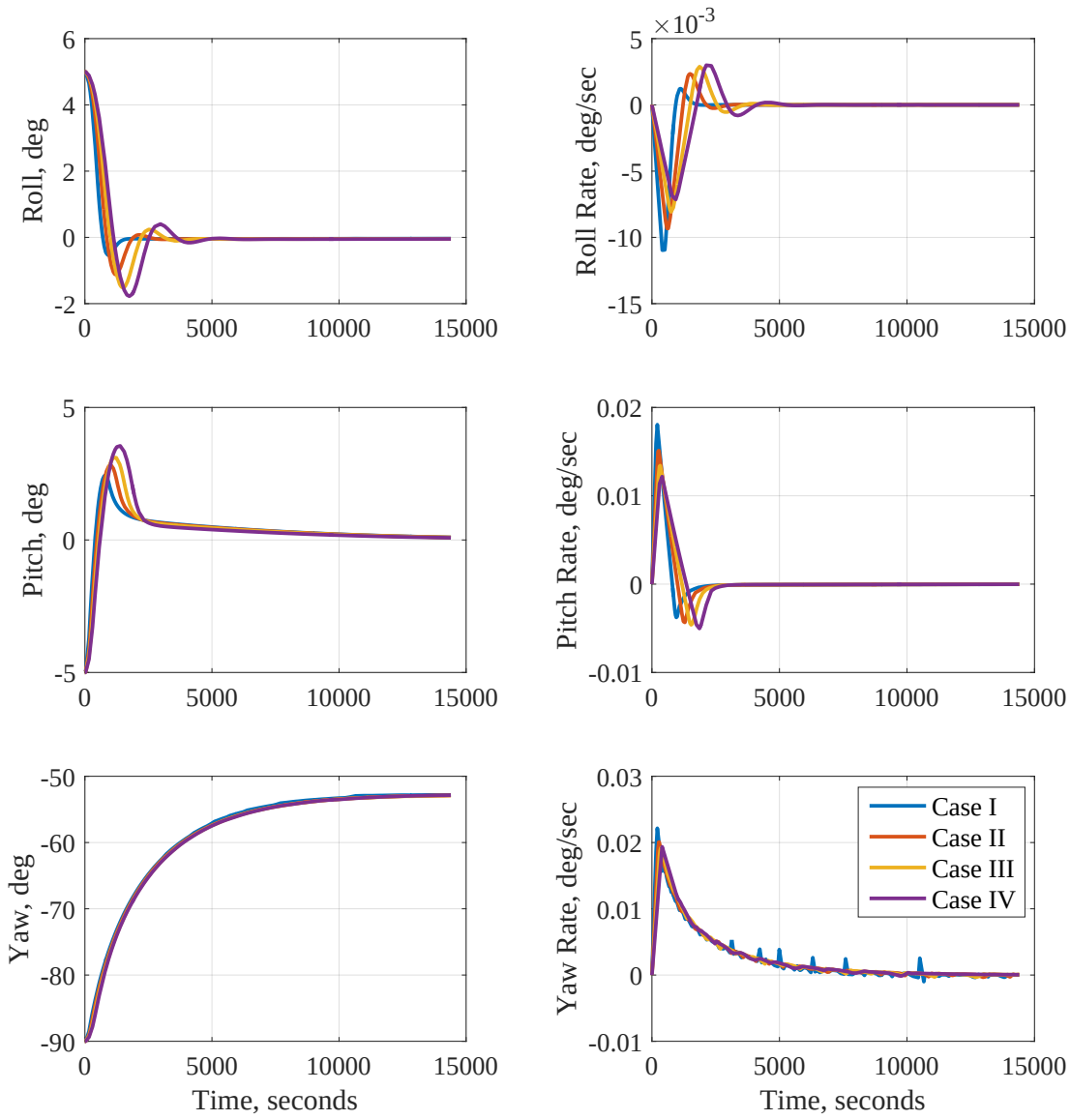


Fig. 5.10: Robustness of the PSO-PID controller to the uncertainties in the moments of inertia with initial condition x_0=[5° -5° -90° 0 0 0] and commanded input x_c=[0 0 -55° 0 0 0].

Chapter 6
Conclusion

In this study, we presented a Takagi-Sugeno type fuzzy attitude controller for a typical solar sail with reaction wheels and translating masses. The T-S fuzzy model rules were built by linearizing the nonlinear equations of motion about 28 Euler angle points. Full state feedback control was implemented by the Twin Parallel Distributed Compensation technique to stabilize the solar sail about any desired state with arbitrary initial conditions. The gains for the two feedback controllers were derived by solving a set of linear matrix inequalities with actuator constraints considered. The performance of the TPDC on the nonlinear system was compared to a PID controller that was tuned using the Ziegler-Nichols method. The results of the numerical simulation show that the TPDC is stable and has a satisfactory performance. The TPDC had a faster settling time with less overshoot than the PID controller. We also examined the robustness of the TPDC by adding uncertainties to the solar sail's principle moments of inertia. Simulation results showed the TPDC is robust to these uncertainties and was able to stabilize the solar sail to the desired state with adequate performance.

We experienced some numerical difficulties in obtaining the feedback gain matrices for the solar sail. It was found that repeated nominal input matrices B_i yielded asymptotically stable gain matrices. We also observed that the effect of the disturbance on the construction of the T-S fuzzy model requires at least 7 linearization points for ψ in order to have an adequate model validation.

We also built a Fuzzy Logic Supervisory (FLS) PID controller to control the attitude of a

solar sail with translating masses. The supervisor tunes the gains of PID controller based on the error and error rate of the Euler angles in order to improve performance. Particle swarm optimization (PSO) and Ziegler-Nichols were used to obtain a range of the PID gains to build the FLS-PID controller. To compare the performance and robustness of the controller, we use a baseline PSO-PID controller. The FLS-PID performed just as well as the PSO-PID controller over its optimal operating points and outperformed the PSO-PID controller over other operating points. Furthermore, the FLS-PID was shown to be robust to model uncertainties by varying the moments of inertia.

Both controllers had advantages and disadvantages over the other. The gains for the TPDC were easily obtained by solving the set of LMIs for the T-S fuzzy model where as the range of gains for the FLS-PID were obtained through trial and error by testing different gains. However, because the TPDC calculates gains based on an approximated fuzzy model, the LMIs could not calculate stable gains for the T-S fuzzy model when the trim masses are inputs due to the complexity of the linearization. Conversely, because the FLS-PID gains were always tested using the full nonlinear model, they were able to use the trim masses as inputs to the solar sail. Regardless, both control techniques were able to stabilize their respective models to the desired states, even in the presence of uncertainties.

Future work would be to include the actuator dynamics for the roll stabilizer bar and trim masses. A more accurate T-S fuzzy model with the trim masses as inputs may be tested using more rules. Also, the number of membership functions for the FLS could be increased from three to five.

Bibliography

[BA16a] Joshua Baculi and Mohammad A Ayoubi. Fuzzy attitude control of solar sail with translating control masses via linear matrix inequalities. In *AAS/AIAA Space Flight Mechanics Meeting*, 2016. 5, 19, 28

[BA16b] Joshua Baculi and Mohammad A Ayoubi. Fuzzy-logic supervisory pid attitude control of solar-sail. In *AIAA/AAS Astrodynamics Specialist Conference*, page 5646, 2016. 43

[Cle99] Maurice Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999. x, 39

[CR12] Inc. CVX Research. CVX: Matlab software for disciplined convex programming, version 2.0. http://cvxr.com/cvx, Aug 2012. 24

[EG12] N Ebrahimi and A Gharaveisi. Optimal fuzzy supervisor controller for an active suspension system. *International Journal of Soft Computing and Engineering*, 2(4):36–39, 2012. 37

[FE15] Bo Fu and Fidelis O. Eke. Attitude control methodology for large solar sails. *Journal of Guidance, Control, and Dynamics*, 38(4):662–670, 2015. 3

[FGE] Bo Fu, Gilbert Gede, and Fidelis O. Eke. Controllability of a square solar sail with moveable membrane. 3

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimizationl. *Proc. IEEE Int. Conf. Neural Networks*, 1995. 30

[KR14] GUV Ravi Kumar and Mr Ch VN Raja. Comparison between fsc and pid controller for 5dof robot arm. *International Journal of Emerging Trends in Electrical and Electronics (IJETEE–ISSN: 2320-9569) Vol*, 10:1–6, 2014. 5

[LSL+96] M Leipold, W Seboldt, S Lingner, E Borg, A Herrmann, A Pabsch, O Wagner, and J Brückner. Mercury sun-synchronous polar orbiter with a solar sail. *Acta Astronautica*, 39(1):143–151, 1996. 2

[MA] Lilit Mazmanyan and Mohammad A Ayoubi. Takagi-sugeno fuzzy model-based attitude control of spacecraft with partially-filled fuel tank. In *AIAA/AAS Astrodynamics Specialist Conference*. 5

[MHM+07] Malcolm Macdonald, Gareth Hughes, Colin McInnes, Aleksander Lyngvi, Peter Falkner, and Alessandro Atzei. Geosail: an elegant solar sail demonstration mission. *Journal of Spacecraft and Rockets*, 44(4):784–796, 2007. 2

[MM10] Malcolm Macdonald and Colin R McInnes. Solar sail mission applications and future advancement. *2nd International Symposiun on Solar Sailing, ISSS 2010*, 2010. 2

[MW76] PJ MacVicar-Whelan. Fuzzy sets for man-machine interaction. *International Journal of Man-Machine Studies*, 8(6):687–697, 1976. 37

[PAG+01] H Price, J Ayon, C Garner, G Klose, E Mettler, and G Sprague. Design for a solar sail demonstration mission. 2001. 2

[PYR98] Kevin M Passino, Stephen Yurkovich, and Michael Reinfrank. *Fuzzy control*, volume 20. Citeseer, 1998. 5

[RAG03] Kourosh Rahnamai, Payman Arabshahi, and Andrew Gray. Fuzzy supervised optimal regulator for spacecraft formation flying. In *Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American*, pages 329–334. IEEE, 2003. 6

[SA14] Chokri Sendi and Mohammad A Ayoubi. Robust fuzzy logic-based tracking control of a flexible spacecraft with h-inf performance criteria. In *AIAA SPACE 2014 Conference and Exposition*, page 4417, 2014. 5

[SA15] Chokri Sendi and Mohammad A Ayoubi. Robust-optimal fuzzy model-based control of flexible spacecraft with actuator amplitude and rate constraints. In *ASME 2015 Dynamic Systems and Control Conference*, pages V001T06A005–V001T06A005. American Society of Mechanical Engineers, 2015. 5

[STK11] Mahmud Iwan Solihin, Lee Fook Tack, and Moey Leap Kean. Tuning of pid controller using particle swarm optimization (pso). *International Journal on Advanced Science, Engineering and Information Technology*, 1(4):458–461, 2011. x, 34, 39

[TA12] Eric Ting and Mohammad Ayoubi. An optimal fuzzy-pid controller for aircraft pitch control. In *AIAA Infotech@ Aerospace Conference*, 2012. 33

[TTYW99a] Tadanari Taniguchi, Kazuo Tanaka, Kazuo Yamafuji, and Hua O Wang. A new pdc for fuzzy reference models. In *Fuzzy Systems Conference*

Proceedings, 1999. FUZZ-IEEE'99. 1999 IEEE International, volume 2, pages 898–903. IEEE, 1999. 4, 16, 19, 21

[TTYW99b] Tadanari Taniguchi, Kazuo Tanaka, Kazuo Yamafuji, and Hua O Wang. Nonlinear model following control via takagi-sugeno fuzzy model. In *American Control Conference, 1999. Proceedings of the 1999*, volume 3, pages 1837–1841. IEEE, 1999. 21

[TW01] Kazuo Tanaka and Hua O. Wang. *Fuzzy Control Systems Design and Analysis*. John Wiley and Sons, Inc., 605 Third Avenue New York, NY, 1 edition, 2001. 11, 35

[vdHMTM15] Jozef van der Ha, Yuya Mimasu, Yuichi Tsuda, and Osamu Mori. Solar and thermal radiation models and flight evaluation for ikaros solar sail. *Journal of Spacecraft and Rockets*, 52(3):958–967, 2015. 2

[WCL10] Rong-Jong Wai, Kun-Lun Chuang, and Jeng-Dao Lee. On-line supervisory control design for maglev transportation system via total sliding-mode approach and particle swarm optimization. *Automatic Control, IEEE Transactions on*, 55(7):1544–1559, 2010. 5

[Wie04a] Bong Wie. Solar sail attitude control and dynamics, part 1. *Journal of Guidance, Control, and Dynamics*, 27(4):526–535, 2004. viii, 1, 3

[Wie04b] Bong Wie. Solar sail attitude control and dynamics, part 2. *Journal of Guidance, Control, and Dynamics*, 27(4):526–535, 2004. 3

[Wie08] Bong Wie. *Space Vehicle Dynamics and Control*. AIAA, 1801 Alexander Bell Drive reston, VA, 2 edition, 2008. x, 2, 4, 7, 24, 39, 40

[WMPT04] Bong Wie, David Murphy, Michael Paluszek, and Stephanie Thomas. Robust attitude control systems design for solar sails, part 2: Microppt-

based secondary acs. In *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 16–19, 2004. 4

[WTG95] Hua O Wang, K Tanaka, and M Griffin. Parallel distributed compensation of nonlinear systems by takagi-sugeno fuzzy model. In *Fuzzy Systems, 1995. International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE Int*, volume 2, pages 531–538. IEEE, 1995. 19

[YT11] R Funase et al. Y Tsuda, O Mori. Flight status of ikaros deep space solar sail demonstrator. *Acta Astronautica*, 69(9-10):833–840, 2011. 1

Appendix A

Twin Parallel Distributed Compensation Code

A.1 Takagi-Sugeno Fuzzy Matrices

This code creates the fuzzy rules that act as the linearization points for the Takagi-Sugeno (T-S) Fuzzy Model. These points are then used to calculate the A and B matrices that build the T-S Fuzzy Model

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

Initialize parameters

```
clear,clc
close all
%...Mass and torque properties for a 40m solar sail (pg 781)
sail_size=40; %m %Sail size = 40m x 40m
sf=75; %percent %Scallop factor
Area=1600; %m^2 % Sail area
Fs=0.01; %N %Sail thrust force (eta*P*A)
Ix=4340; %kg-m^2
Iy=2171; %kg-m^2
Iz=2171; %kg-m^2
epsilon=0.1; %m %cp-cp offset
Tpy=1.0; %mN*m %Pitch/yaw solar disturbance torque
Tr=0.5; %mN*m %Roll solar disturbance torque
%...End mass and torque propertis from pg 781

%...Control parameters for a 40m solar sail (pg 795)
m=1; %kg %Trim control mass (TCM)
M=148; %kg %Main-body mass
v_TCM=0.05; %m/s %TCM speed limit
y_max=28; %m %TCM y_max=+-28 m
z_max=y_max;
y_ss=14.9; %m %Steady-state trim value to counter epsilon
z_ss=y_ss; %m
T=560; %s %Actuator time constant
%...End control parameters from pg 795

%...Roll control parameters for a 1m RSB (pg 797)
Theta_max=45*pi/180; %rad % RSB max deflection angle=+-45 deg
l_RSB=1; %m %RSB moment arm length
T_max=(0.5/20)*13.3*Fs*sin(Theta_max);
%...End roll control parameters from pg 797

%...Parameters from pg 805 of the book
omega_max=0.05*pi/180; %rad
%...End parameters from pg 805

% Other values
mr=m*(M+m)/(M+2*m); %kg %Reduced mass
n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer orbit
 (SSTO)
                  %Value for this mission taken from pg 762
P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1 AU
 from
                  %the sun (pg. 793)
```

Creating the fuzzy rules

```
format long
% Defining linearization range
phi0=[-5 5]*pi/180;
theta0=[-5 5]*pi/180;
psi0=[-105 -85 -65 -45 -25 -5 15]*pi/180; %TRY MORE RULES

% Number of rules
nr=length(phi0)*length(theta0)*length(psi0);

%=========================================================================
% Following code for making matrix consisting of rule combinations
 obatined
% from http://stackoverflow.com/questions/21895335/generate-a-matrix-
containing-all-combinations-of-elements-taken-from-n-vectors?lq=1
%=========================================================================
vectors={[phi0], [theta0], [psi0]}; % input data: cell array of
 vectors
nvec=numel(vectors); %number of vectors
rule=cell(1,nvec); %pre-define to generate comma-separated list
[rule{end:-1:1}]=ndgrid(vectors{end:-1:1}); %the reverse order in
 these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
rule = cat(nvec+1, rule{:}); %concat the n n-dim arrays along
 dimension n+1
rule = reshape(rule,[],nvec); %reshape to obtain desired matrix
rule
%=========================================================================

%{
Defining states
x1=phi
x2=theta
x3=psi
x4=phi-dot
x5=theta-dot
x6=psi-dot
%}
```

Creating the A and B matrices

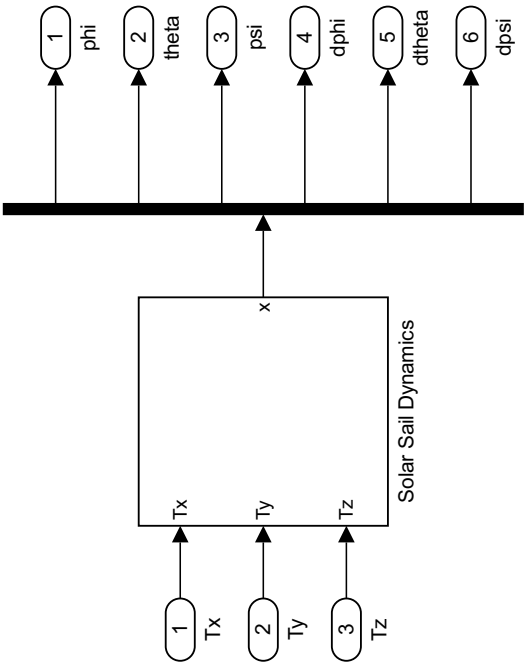
```
% Initialize A and B matrices
A=[];
B=[];

% Initialize state and input vectors
x=[];
u=[];

for r=1:nr
    r %counter for iterations
```

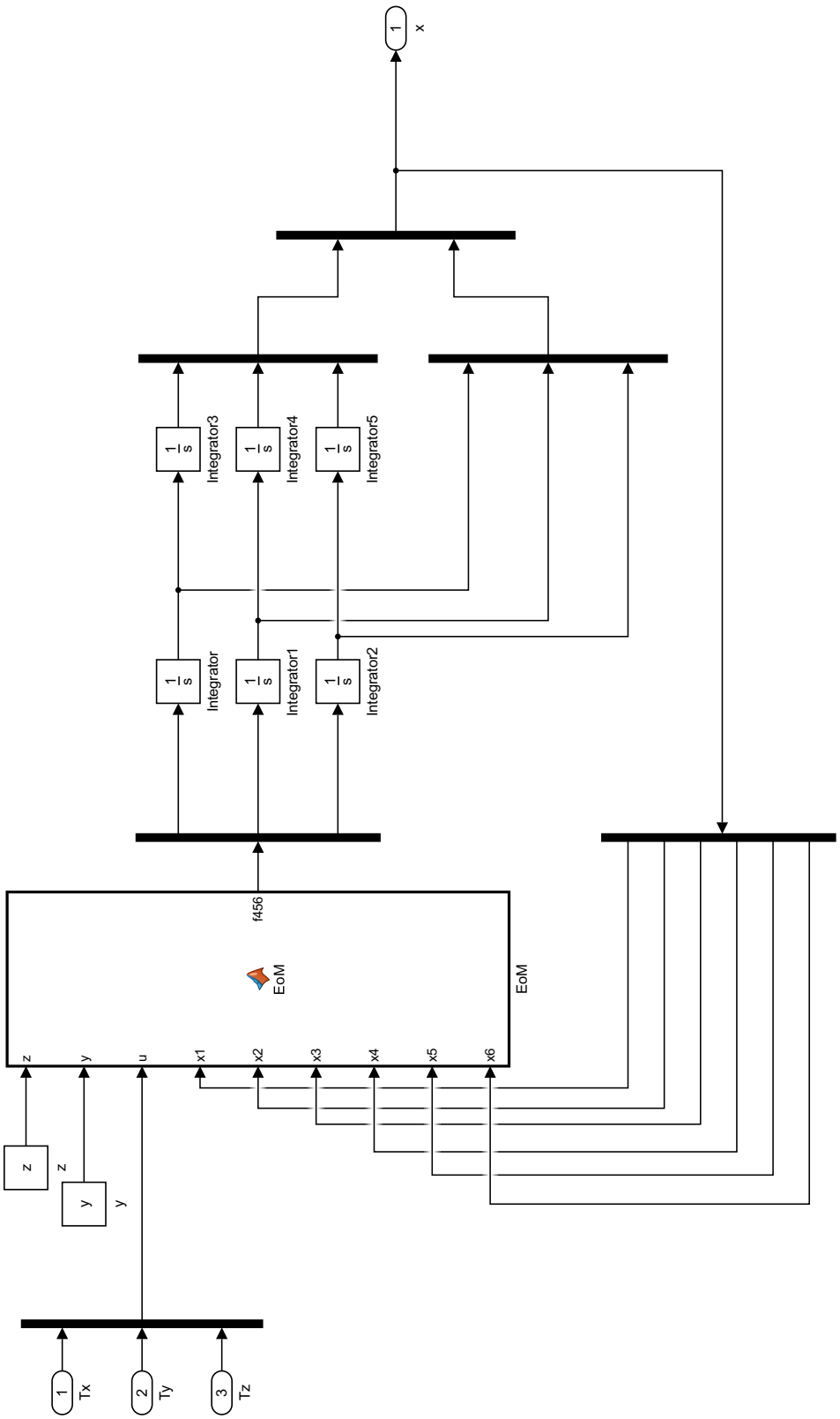
```
    % Defining the Euler angles as states
    x1=rule(r,1); %phi
    x2=rule(r,2); %theta
    x3=rule(r,3); %psi

    % Assume Euler angle derivatives are 0 for now
    x4=0;
    x5=0;
    x6=0;

    z=-z_ss;
    y=y_ss;

    % Calculating the J's
    Jx=Ix+mr*(y^2+z^2);
    Jy=Iy+mr*z^2;
    Jz=Iz+mr*y^2;

    Tx = ((Jy-Jz)*(n^2)*(3+cos(x3)^2)*x1) - ((Jy-
Jz)*(n^2)*cos(x3)*sin(x3)*x2) - ((Jx-Jy+Jz)*n*cos(x3)*x6) -
 (0.5*Fs*epsilon*sin(x3)^2);
    Ty = ((Jx-Jz)*(n^2)*(3+sin(x3)^2)*x2) - ((Jx-
Jz)*(n^2)*cos(x3)*sin(x3)*x1) - ((Jx-Jy-Jz)*n*sin(x3)*x6) -
 (Fs*m*z*sin(x3)^2)/((2*m+M)) - (Fs*epsilon*sin(x3)^2);
    Tz = ((-Jx+Jy)*(n^2)*cos(x3)*sin(x3)) + ((Jx-Jy+Jz)*n*cos(x3)*x4)
 + ((Jx-Jy-Jz)*n*sin(x3)*x5) + (Fs*m*y*sin(x3)^2)/((2*m+M)) -
 (Fs*epsilon*sin(x3)^2);

    xoperpt=[x1 x2 x3 x4 x5 x6]';
    x0=xoperpt;
    x0=[0 0 0 0 0 0];
    utrim=[Tx Ty Tz]';

 [A(:,:,r),B(:,:,r),C(:,:,r),D(:,:,r)]=linmod('SS_Txyz',xoperpt,utrim);

    % Saving state and input vectors
    x(:,r)=[x1 x2 x3 x4 x5 x6]';
    u(:,r)=[Tx Ty Tz]';

    % Nonlinear equations that will be linearized
    F1=x4;
    F2=x5;
    F3=x6;
    F4=Tx/Jx - ((Jy-Jz)*(n^2)*(3+cos(x3)^2)*x1)/Jx + ((Jy-
Jz)*(n^2)*cos(x3)*sin(x3)*x2)/Jx + ((Jx-Jy+Jz)*n*cos(x3)*x6)/Jx +
 (0.5*Fs*epsilon*sin(x3)^2)/Jx;
    F5=Ty/Jy - ((Jx-Jz)*(n^2)*(3+sin(x3)^2)*x2)/Jy + ((Jx-
Jz)*(n^2)*cos(x3)*sin(x3)*x1)/Jy + ((Jx-Jy-Jz)*n*sin(x3)*x6)/Jy +
 (m*Fs*z*sin(x3)^2)/((2*m+M)*Jy) + (Fs*epsilon*sin(x3)^2)/Jy;
    F6=Tz/Jz - ((-Jx+Jy)*(n^2)*cos(x3)*sin(x3))/Jz - ((Jx-
Jy+Jz)*n*cos(x3)*x4)/Jz - ((Jx-Jy-Jz)*n*sin(x3)*x5)/Jz -
 (m*Fs*y*sin(x3)^2)/((2*m+M)*Jz) + (Fs*epsilon*sin(x3)^2)/Jz;
    F(:,r)=[F1 F2 F3 F4 F5 F6]';
```

```
end
```

Published with MATLAB® R2015b

A.2 Solar Sail Nonlinear Model




1
x
Integrator3
Integrator4
Integrator5
Integrator
Integrator1
Integrator2
f456
EoM
EoM
z
y
u
x1
x2
x3
x4
x5
x6
z
z
y
y
1
Tx
2
Ty
3
Tz

This code consists of the equations of motion used in the "EoM" box in the SS_Txyz.slx file

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

EoM

```
function f456 = EoM(z,y,u,x1,x2,x3,x4,x5,x6)
%SS_Txyz
%=========================================================================
% Parameters
%=========================================================================
%...Mass and torque properties for a 40m solar sail (pg 781)
sail_size=40; %m %Sail size = 40m x 40m
sf=75; %percent %Scallop factor
Area=1600; %m^2 % Sail area
Fs=0.01; %N %Sail thrust force (eta*P*A)
Ix=4340; %kg-m^2
Iy=2171; %kg-m^2
Iz=2171; %kg-m^2
epsilon=0.1; %m %cp-cp offset
Tpy=1.0; %mN*m %Pitch/yaw solar disturbance torque
Tr=0.5; %mN*m %Roll solar disturbance torque
%...End mass and torque propertis from pg 781

%...Control parameters for a 40m solar sail (pg 795)
m=1; %kg %Trim control mass (TCM)
M=148; %kg %Main-body mass
v_TCM=0.05; %m/s %TCM speed limit
y_max=28; %m %TCM y_max=+-28 m
z_max=y_max;
y_ss=14.9; %m %Steady-state trim value to counter epsilon
z_ss=y_ss; %m
T=560; %s %Actuator time constant
%...End control parameters from pg 795

%...Roll control parameters for a 1m RSB (pg 797)
Theta_max=45; %rad % RSB max deflection angle=+-45 deg
l_RSB=1; %m %RSB moment arm length
T_max=(0.5/20)*13.3*Fs*sind(Theta_max);
%...End roll control parameters from pg 797

%...Parameters from pg 805 of the book
omega_max=0.05*pi/180; %rad
%...End parameters from pg 805

%...end parameters

% Other values
mr=m*(M+m)/(M+2*m); %kg %Reduced mass
n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer orbit
 (SSTO)
                  %Value for this mission taken from pg 762
```

```
P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1 AU
 from
                    %the sun (pg. 793)
%==========================================================================

%==========================================================================
% Equations
%==========================================================================
% Calculating the J's
Jx=Ix+mr*(y^2+z^2);
Jy=Iy+mr*z^2;
Jz=Iz+mr*y^2;

Tx=u(1);
Ty=u(2);
Tz=u(3);

f1 = x4;
f2 = x5;
f3 = x6;
f4 = Tx/Jx - ((Jy-Jz)*(n^2)*(3+cos(x3)^2)*x1)/Jx + ((Jy-
Jz)*(n^2)*cos(x3)*sin(x3)*x2)/Jx + ((Jx-Jy+Jz)*n*cos(x3)*x6)/Jx +
 (0.5*Fs*epsilon*sin(x3)^2)/Jx;
f5 = Ty/Jy - ((Jx-Jz)*(n^2)*(3+sin(x3)^2)*x2)/Jy + ((Jx-
Jz)*(n^2)*cos(x3)*sin(x3)*x1)/Jy + ((Jx-Jy-Jz)*n*sin(x3)*x6)/Jy +
 (m*Fs*z*sin(x3)^2)/((2*m+M)*Jy) + (Fs*epsilon*sin(x3)^2)/Jy;
f6 = Tz/Jz - ((-Jx+Jy)*(n^2)*cos(x3)*sin(x3))/Jz - ((Jx-
Jy+Jz)*n*cos(x3)*x4)/Jz - ((Jx-Jy-Jz)*n*sin(x3)*x5)/Jz -
 (m*Fs*y*sin(x3)^2)/((2*m+M)*Jz) + (Fs*epsilon*sin(x3)^2)/Jz;

f456=[f4 f5 f6]';

%SS_Txyz
end
```

Published with MATLAB® R2015b

A.3 Takagi-Sugeno Model Validation

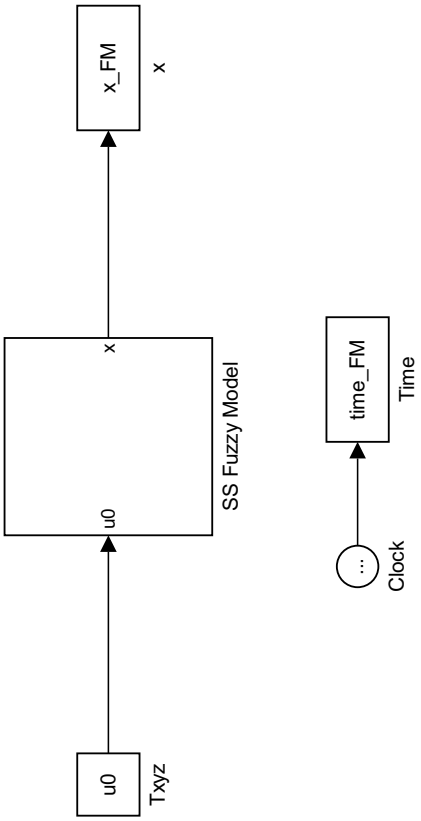
This code plots the open loop simulation of both the T-S Fuzzy Model and the nonlinear model for model validation

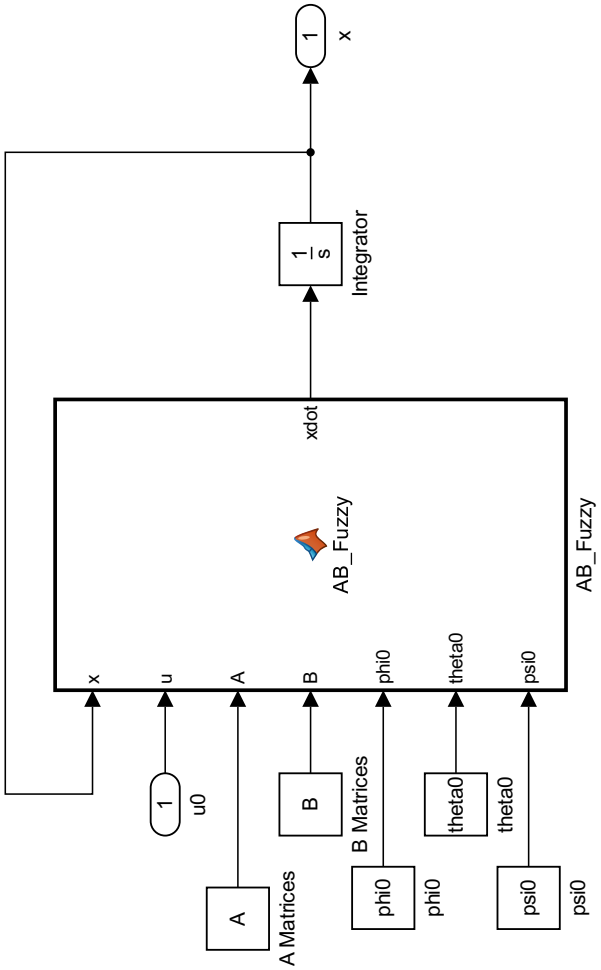
```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

Open Loop Simulation

```
clear,clc
close all

load('AB_3PV28R_Txyz_zyss_105-15_dist.mat')
T=1*3600;
R=7;
% x0=x(:,R);
x0=[0 0 -45 0 0 0]'*pi/180;
clear x
% u0=u(:,R);
u0=[0 0 0]';
sim('SS_NL_Txyz')
sim('SS_Fuzzy_Txyz')

figure
% Euler angles
subplot(3,2,1)
plot(time_NL,x_NL(:,1)*180/pi,'--',time_FM,x_FM(:,1)*180/pi)
grid on
ylabel('Roll (deg)')
subplot(3,2,3)
plot(time_NL,x_NL(:,2)*180/pi,'--',time_FM,x_FM(:,2)*180/pi)
grid on
ylabel('Pitch (deg)')
subplot(3,2,5)
plot(time_NL,x_NL(:,3)*180/pi,'--',time_FM,x_FM(:,3)*180/pi)
grid on
ylabel('Yaw (deg)')
xlabel('Time (sec)')

% Euler angle derivatives
subplot(3,2,2)
plot(time_NL,x_NL(:,4)*180/pi,'--',time_FM,x_FM(:,4)*180/pi)
grid on
ylabel('Roll Rate (deg/s)')
subplot(3,2,4)
plot(time_NL,x_NL(:,5)*180/pi,'--',time_FM,x_FM(:,5)*180/pi)
grid on
ylabel('Pitch Rate (deg/s)')
subplot(3,2,6)
plot(time_NL,x_NL(:,6)*180/pi,'--',time_FM,x_FM(:,6)*180/pi)
grid on
ylabel('Yaw Rate (deg/s)')
xlabel('Time (sec)')
legend('Nonlinear Model','T-S Fuzzy Model')
```

A.4 Open Loop Takagi-Sugeno Fuzzy Model




1
x
1
s
Integrator
xdot
AB_Fuzzy
AB_Fuzzy
x
u
A
B
phi0
theta0
psi0
1
u0
B
B Matrices
theta0
theta0
A
A Matrices
phi0
phi0
psi0
psi0

This code builds the T-S fuzzy model for open loop model validation in the "AB_Fuzzy" box in the SS_Fuzzy_Txyz.slx file

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

AB_Fuzzy

```
function xdot = AB_Fuzzy(x,u,A,B,phi0,theta0,psi0)
%SS_Fuzzy_Txyz

% This functions builds the T-S Fuzzy Model
% This is part of the example system from the Taniguchi paper for a 2
 rule
% fuzzy model with a step input

% ========================================================================
% T-S Fuzzy Model
% ========================================================================
[n,n,r]=size(A);

rphi0=length(phi0);
if  rphi0==2
    x11=phi0(1);
    x12=phi0(2);
    mu1=zeros(1,2);
    if x(1)>=x11 & x(1)<=x12
        mu1(1,1)=(x(1)-x12)/(x11-x12);
        mu1(1,2)=(x(1)-x11)/(x12-x11);
    elseif x(1)>x12
        mu1(1,1)=0;
        mu1(1,2)=1;
    else
        mu1(1,1)=1;
        mu1(1,2)=0;
    end
elseif rphi0==3
    x11=phi0(1);
    x12=phi0(2);
    x13=phi0(3);
    mu1=zeros(1,3);
    if x(3)>=x11 & x(3)<=x12
        mu1(1,1)=(x(3)-x12)/(x11-x12);
        mu1(1,2)=(x(3)-x11)/(x12-x11);
        mu1(1,3)=0;
    elseif x(3)>x12 & x(3)<=x13
        mu1(1,1)=0;
        mu1(1,2)=(x(3)-x13)/(x12-x13);
        mu1(1,3)=(x(3)-x12)/(x13-x12);
    elseif x(3)>x13
        mu1(1,1)=0;
        mu1(1,2)=0;
        mu1(1,3)=1;
```

```
    elseif x(3)<x11
        mu1(1,1)=1;
        mu1(1,2)=0;
        mu1(1,3)=0;
    end
end

rtheta0=length(theta0);
if rtheta0==2
    x21=theta0(1);
    x22=theta0(2);
    mu2=zeros(1,2);
    if x(2)>=x21 & x(2)<=x22
        mu2(1,1)=(x(2)-x22)/(x21-x22);
        mu2(1,2)=(x(2)-x21)/(x22-x21);
    elseif x(2)>x22
        mu2(1,1)=0;
        mu2(1,2)=1;
    else
        mu2(1,1)=1;
        mu2(1,2)=0;
    end
elseif rtheta0==3
    x21=theta0(1);
    x22=theta0(2);
    x23=theta0(3);
    mu2=zeros(1,3);
    if x(3)>=x21 & x(3)<=x22
        mu2(1,1)=(x(3)-x22)/(x21-x22);
        mu2(1,2)=(x(3)-x21)/(x22-x21);
        mu2(1,3)=0;
    elseif x(3)>x22 & x(3)<=x23
        mu2(1,1)=0;
        mu2(1,2)=(x(3)-x23)/(x22-x23);
        mu2(1,3)=(x(3)-x22)/(x23-x22);
    elseif x(3)>x23
        mu2(1,1)=0;
        mu2(1,2)=0;
        mu2(1,3)=1;
    elseif x(3)<x21
        mu2(1,1)=1;
        mu2(1,2)=0;
        mu2(1,3)=0;
    end
end

rpsi0=length(psi0);
if rpsi0==2
    x31=psi0(1);
    x32=psi0(2);
    if x(3)>=x31 & x(3)<=x32
        mu(3,1)=(x(3)-x32)/(x31-x32);
        mu(3,2)=-(x(3)-x31)/(x32-x31);
    elseif x(3)>x32
```

```
        mu(3,1)=0;
        mu(3,2)=1;
    else
        mu(3,1)=1;
        mu(3,2)=0;
    end
elseif rpsi0==3
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    mu3=zeros(1,3);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
    elseif x(3)>x33
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
    end
elseif rpsi0==4
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    mu3=zeros(1,4);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
    elseif x(3)>x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
```

```
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
    end
elseif rpsi0==5
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    x35=psi0(5);
    mu3=zeros(1,5);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
        mu3(1,5)=0;
    elseif x(3)>x34 & x(3)<=x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=(x(3)-x35)/(x34-x35);
        mu3(1,5)=(x(3)-x34)/(x35-x34);
    elseif x(3)>x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    end
elseif rpsi0==6
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
```

```
        x35=psi0(5);
        x36=psi0(6);
        mu3=zeros(1,6);
        if x(3)>=x31 & x(3)<=x32
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
            mu3(1,4)=(x(3)-x33)/(x34-x33);
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x34 & x(3)<=x35
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=(x(3)-x35)/(x34-x35);
            mu3(1,5)=(x(3)-x34)/(x35-x34);
            mu3(1,6)=0;
        elseif x(3)>x35 & x(3)<=x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=(x(3)-x36)/(x35-x36);
            mu3(1,6)=(x(3)-x35)/(x36-x35);
        elseif x(3)>x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        end
    elseif rpsi0==7
```

```
x31=psi0(1);
x32=psi0(2);
x33=psi0(3);
x34=psi0(4);
x35=psi0(5);
x36=psi0(6);
x37=psi0(7);
mu3=zeros(1,7);
if x(3)>=x31 & x(3)<=x32
    mu3(1,1)=(x(3)-x32)/(x31-x32);
    mu3(1,2)=(x(3)-x31)/(x32-x31);
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x32 & x(3)<=x33
    mu3(1,1)=0;
    mu3(1,2)=(x(3)-x33)/(x32-x33);
    mu3(1,3)=(x(3)-x32)/(x33-x32);
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x33 & x(3)<=x34
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=(x(3)-x34)/(x33-x34);
    mu3(1,4)=(x(3)-x33)/(x34-x33);
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
```

```
            mu3(1,6)=(x(3)-x37)/(x36-x37);
            mu3(1,7)=(x(3)-x36)/(x37-x36);
        elseif x(3)>x37
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
        end
    elseif rpsi0==9
        x31=psi0(1);
        x32=psi0(2);
        x33=psi0(3);
        x34=psi0(4);
        x35=psi0(5);
        x36=psi0(6);
        x37=psi0(7);
        x38=psi0(8);
        x39=psi0(9);
        mu3=zeros(1,9);
        if x(3)>=x31 & x(3)<=x32
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
```

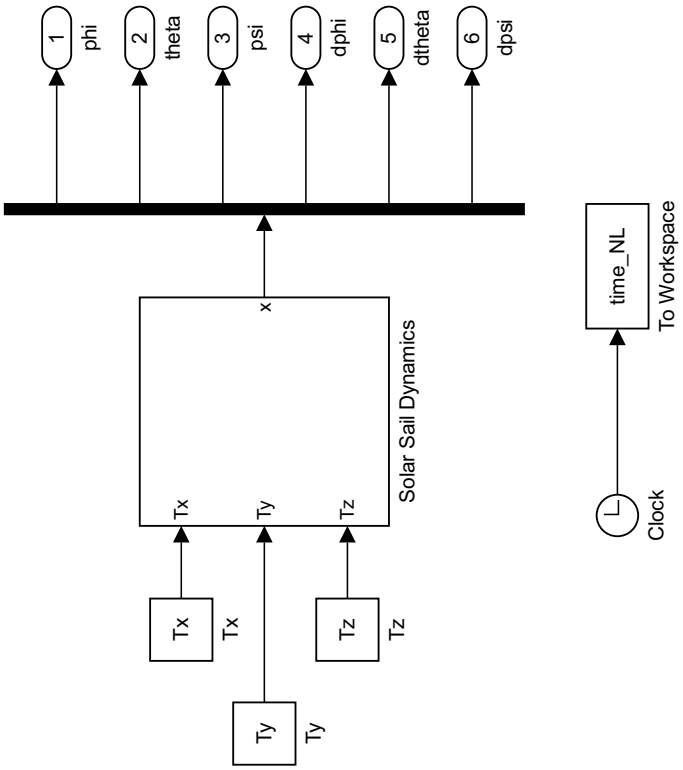
```
    mu3(1,4)=(x(3)-x33)/(x34-x33);
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=(x(3)-x37)/(x36-x37);
    mu3(1,7)=(x(3)-x36)/(x37-x36);
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x37 & x(3)<=x38
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=(x(3)-x38)/(x37-x38);
    mu3(1,8)=(x(3)-x37)/(x38-x37);
    mu3(1,9)=0;
elseif x(3)>x38 & x(3)<=x39
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
```

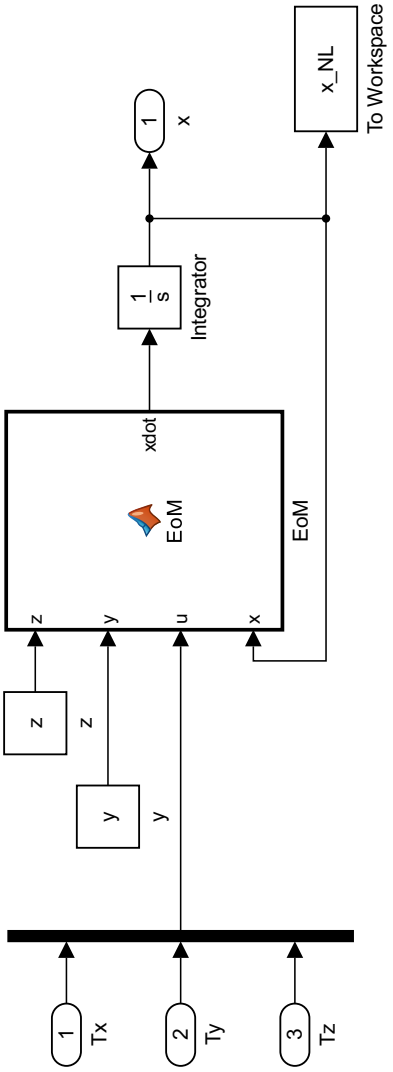
```
        mu3(1,8)=(x(3)-x39)/(x38-x39);
        mu3(1,9)=(x(3)-x38)/(x39-x38);
    elseif x(3)>x39
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=0;
    end
end

%============================
% CHECK MEMBERSHIP FUNCTIONS %
%============================
% Combining the fuzzy membership functions
Mu={[mu1(1,:)], [mu2(1,:)], [mu3(1,:)]}; % input data: cell array of
 vectors
nvec=numel(Mu); %number of vectors
w=cell(1,nvec); %pre-define to generate comma-separated list
[w{3:-1:1}]=ndgrid(Mu{end:-1:1}); %the reverse order in these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
w = cat(nvec+1, w{:}); %concat the n n-dim arrays along dimension n+1
w = reshape(w,[],nvec); %reshape to obtain desired matrix
w=prod(w')';

h=zeros(size(w));
Afi=zeros(size(A));
Bfi=zeros(size(B));
[n,n,r]=size(A);
for i=1:r
    h(i)=w(i)/sum(w);
    Afi(:,:,i)=h(i).*A(:,:,i);
    Bfi(:,:,i)=h(i).*B(:,:,i);
end
Af=sum(Afi,3);
Bf=sum(Bfi,3);
% ========================================================================
```

```
xdot=zeros(6,1);
xdot=Af*x+Bf*u;

%SS_Fuzzy_Txyz
end
```

Published with MATLAB® R2015b

A.5 Open Loop Solar Sail Nonlinear Model




1
Tx
2
Ty
3
Tz
y
y
z
z
z
y
u
x
EoM
EoM
xdot
1
s
Integrator
1
x
x_NL
To Workspace

This code consists of the equations of motion used in the "EoM" box in the SS_NL_Txyz.slx file

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

EoM

```
function xdot  = EoM(z,y,u,x)
%SS_NL_Txyz
%=========================================================================
% Parameters
%=========================================================================
%...Mass and torque properties for a 40m solar sail (pg 781)
sail_size=40; %m %Sail size = 40m x 40m
sf=75; %percent %Scallop factor
Area=1600; %m^2 % Sail area
Fs=0.01; %N %Sail thrust force (eta*P*A)
Ix=4340; %kg-m^2
Iy=2171; %kg-m^2
Iz=2171; %kg-m^2
epsilon=0.1; %m %cp-cp offset
Tpy=1.0; %mN*m %Pitch/yaw solar disturbance torque
Tr=0.5; %mN*m %Roll solar disturbance torque
%...End mass and torque propertis from pg 781

%...Control parameters for a 40m solar sail (pg 795)
m=1; %kg %Trim control mass (TCM)
M=148; %kg %Main-body mass
v_TCM=0.05; %m/s %TCM speed limit
y_max=28; %m %TCM y_max=+-28 m
z_max=y_max;
y_ss=14.9; %m %Steady-state trim value to counter epsilon
z_ss=y_ss; %m
T=560; %s %Actuator time constant
%...End control parameters from pg 795

%...Roll control parameters for a 1m RSB (pg 797)
Theta_max=45*pi/180; %rad % RSB max deflection angle=+-45 deg
l_RSB=1; %m %RSB moment arm length
T_max=(0.5/20)*13.3*Fs*sin(Theta_max);
%...End roll control parameters from pg 797

%...Parameters from pg 805 of the book
omega_max=0.05*pi/180; %rad
T_max=(0.5/20)*13.3*Fs*sin(Theta_max); %Nm
%...End parameters from pg 805

%...end parameters

% Other values
mr=m*(M+m)/(M+2*m); %kg %Reduced mass
n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer orbit
 (SSTO)
```

```
                    %Value for this mission taken from pg 762
P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1 AU
 from
                    %the sun (pg. 793)
%=========================================================================

%=========================================================================
% Equations
%=========================================================================
% Calculating the J's
Jx=Ix+mr*(y^2+z^2);
Jy=Iy+mr*z^2;
Jz=Iz+mr*y^2;

x1=x(1);
x2=x(2);
x3=x(3);
x4=x(4);
x5=x(5);
x6=x(6);

% Defining states and inputs
Tx=u(1);
Ty=u(2);
Tz=u(3);
f1=x4;
f2=x5;
f3=x6;
f4=Tx/Jx - ((Jy-Jz)*(n^2)*(3+cos(x3)^2)*x1)/Jx + ((Jy-
Jz)*(n^2)*cos(x3)*sin(x3)*x2)/Jx + ((Jx-Jy+Jz)*n*cos(x3)*x6)/Jx +
 (0.5*Fs*epsilon*sin(x3)^2)/Jx;
f5=Ty/Jy - ((Jx-Jz)*(n^2)*(3+sin(x3)^2)*x2)/Jy + ((Jx-
Jz)*(n^2)*cos(x3)*sin(x3)*x1)/Jy + ((Jx-Jy-Jz)*n*sin(x3)*x6)/Jy +
 (m*Fs*z*sin(x3)^2)/((2*m+M)*Jy)+ (Fs*epsilon*sin(x3)^2)/Jy;
f6=Tz/Jz - ((-Jx+Jy)*(n^2)*cos(x3)*sin(x3))/Jz - ((Jx-Jy
+Jz)*n*cos(x3)*x4)/Jz - ((Jx-Jy-Jz)*n*sin(x3)*x5)/Jz -
 (m*Fs*y*sin(x3)^2)/((2*m+M)*Jz) + (Fs*epsilon*sin(x3)^2)/Jz;


xdot=[f1 f2 f3 f4 f5 f6]';
%SS_NL_Txyz
end
```

Published with MATLAB® R2015b

A.6 TPDC Linear Matrix Inequalities

This code solves the relaxed set of LMIs to calculate the feedback gain matrices F and K for the T-S Fuzzy Model and the Fuzzy Reference Model, respectively

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

LMI Solver

```
clear,clc
close all
load('AB_3PV28R_Txyz_zyss_105-15_dist.mat')
clear x

[n,n]=size(A(:,:,1));
[n,m]=size(B(:,:,1));
[q,n]=size(C(:,:,1));
nr2=nr*nr;

nR=1;
D=zeros(size(A(:,:,1)));
D(1,4)=1;
D(2,5)=1;
D(3,6)=1;

% Actuator constraint
mu=25;

cvx_clear
cvx_precision best
cvx_solver sdpt3
% cvx_solver sedumi %sedumi doesn't solve well
cvx_begin sdp
    variable X(n,n) symmetric
    variable M(m,n,nr)
    variable N(m,n,nR)
    variable bp

    minimize bp
    X>=0;
    for i=1:nr
        for j=1:nr
           A(:,:,i)*X+X*A(:,:,i)'-B(:,:,i)*M(:,:,i)-
M(:,:,i)'*B(:,:,i)'<=0;
           if i<j
               A(:,:,i)*X+X*A(:,:,i)'-B(:,:,i)*M(:,:,j)-
M(:,:,j)'*B(:,:,i)'+(A(:,:,j)*X+X*A(:,:,j)'-B(:,:,j)*M(:,:,i)-
M(:,:,i)'*B(:,:,j)')'<=0
               eq18=[A(:,:,i)*X-B(:,:,i)*M(:,:,i)+A(:,:,j)*X-
B(:,:,j)*M(:,:,j)-(A(:,:,i)*X-B(:,:,i)*M(:,:,j)+A(:,:,j)*X-
B(:,:,j)*M(:,:,i))];
               [4*bp*eye(n) eq18';
                eq18 eye(n)]>=0
           end
```

```
            for k=1:nR
               [bp*eye(n) (A(:,:,i)*X-B(:,:,i)*M(:,:,i)-(D(:,:,k)*X-
B(:,:,i)*N(:,:,k)))';
                A(:,:,i)*X-B(:,:,i)*M(:,:,i)-(D(:,:,k)*X-
B(:,:,i)*N(:,:,k)) eye(n)]>=0;
               [    X      zeros(n,n)  M(:,:,i)';
               zeros(n,n)      X     -N(:,:,k)';
               M(:,:,i)   -N(:,:,k) mu^2*eye(m,m)]>=0;
            end
         end
    end

cvx_end

F=[];
for i=1:nr
    F(:,:,i)=M(:,:,i)*inv(X);
end

K=[];
for k=1:nR
   K(:,:,k)=N(:,:,k)*inv(X);
end
```

Published with MATLAB® R2015b

A.7 TPDC Closed Loop Plotting

This code plots the closed loop simulation of the solar sail model with a Twin Parallel Distributed Compensator (TPDC). The TPDC can be applied to the T-S Fuzzy Model or the nonlinear model.

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

Closed Loop Simulation

```
clear,clc
close all

load('TLMI2_3PV28R_Txyz_zyss_105-15_dist_X0sym_25.mat')

clear x u M m n
% Other values
m=1; %kg %Trim control mass (TCM)
M=148; %kg %Main-body mass
n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer orbit
 (SSTO)
                    %Value for this mission taken from pg 762
P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1 AU
 from
                    %the sun (pg. 793)

% Running simulation
T=3600*4;
x0=[5 -5 -90 0 0 0]'*pi/180;
xR=[0 0 -55 0 0 0]'*pi/180;
Thetax0=0;
% z=-z_ss;
% y=y_ss;

KpTx=0.01;
KpTy=0.01;
KpTz=0.01;
KdTx=0;
KiTx=0;
KdTy=0;
KiTy=0;
KdTz=0;
KiTz=0;

% Z-N Gains
KuTx=KpTx; % Tx gains
TuTx=(10982-6156);
KpTx=0.6*KuTx;
KdTx=KpTx*(TuTx/8);
KiTx=KpTx*2/(TuTx);
KuTy=KpTy; % Ty gains
TuTy=(4606-1479);
KpTy=0.6*KuTy;
KdTy=KpTy*(TuTy/8);
KiTy=KpTy*2/(TuTy);
```

```
KuTz=KpTz; % Tz gains
TuTz=(7543-2589);
KpTz=0.6*KuTz;
KdTz=KpTz*(TuTz/8);
KiTz=KpTz*2/(TuTz);

K_ZN=[KpTx KiTx KdTx KpTy KiTy KdTy KpTz KiTz KdTz]'

%-------------------------------------------------------------------------
% Parameter changes for robustness test
%-------------------------------------------------------------------------
% Only plot the ROBUSTNESS PLOTTING code
% Plot for i=1,2,3,4
rob=[0.8 1.6 2.4 3.2;
     0.4 0.8 1.2 1.6;
     0.3 0.6 0.9 1.2];
i=4;
Ix=rob(1,i)*4340; %kg-m^2
Iy=rob(2,i)*2171; %kg-m^2
Iz=rob(3,i)*2171; %kg-m^2
%-------------------------------------------------------------------------

% sim('SS_TPDC_Txyz')
sim('SS_NL_TPDC_Txyz')
sim('SS_PID_Txyz')

u=u';

x=x'*180/pi;

u_PID=u_PID';
x_PID=x_PID'*180/pi;
```

Closed Loop Plotting

```
% Euler angles
figure(1)
subplot(3,2,1)
plot(time,x(1,:),time_PID,x_PID(1,:),'--')
grid on
ylabel('Roll (deg)')
subplot(3,2,3)
plot(time,x(2,:),time_PID,x_PID(2,:),'--')
grid on
ylabel('Pitch (deg)')
subplot(3,2,5)
plot(time,x(3,:),time_PID,x_PID(3,:),'--')
grid on
ylabel('Yaw (deg)')
xlabel('Time (sec)')

% Euler angle derivatives
subplot(3,2,2)
plot(time,x(4,:),time_PID,x_PID(4,:),'--')
```

```
grid on
ylabel('Roll Rate (deg/s)')
subplot(3,2,4)
plot(time,x(5,:),time_PID,x_PID(5,:),'--')
grid on
ylabel('Pitch Rate (deg/s)')
subplot(3,2,6)
plot(time,x(6,:),time_PID,x_PID(6,:),'--')
grid on
ylabel('Yaw Rate (deg/s)')
xlabel('Time (sec)')
legend('TPDC','PID')

% Inputs
figure(2)
subplot(3,1,1)
plot(time,u(1,:),time_PID,u_PID(1,:),'--')
grid on
ylabel('T_{x} (N\cdot m)')
subplot(3,1,2)
plot(time,u(2,:),time_PID,u_PID(2,:),'--')
grid on
ylabel('T_{y} (N\cdot m)')
subplot(3,1,3)
plot(time,u(3,:),time_PID,u_PID(3,:),'--')
grid on
ylabel('T_{z} (N\cdot m)')
xlabel('Time (sec)')
legend('TPDC','PID')
```

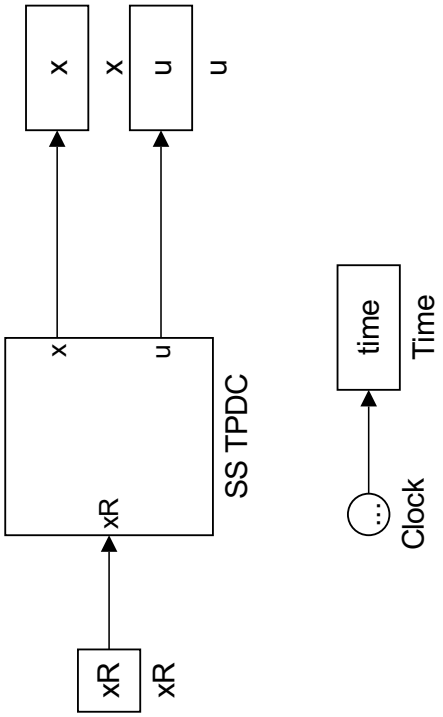
Robustness Plotting

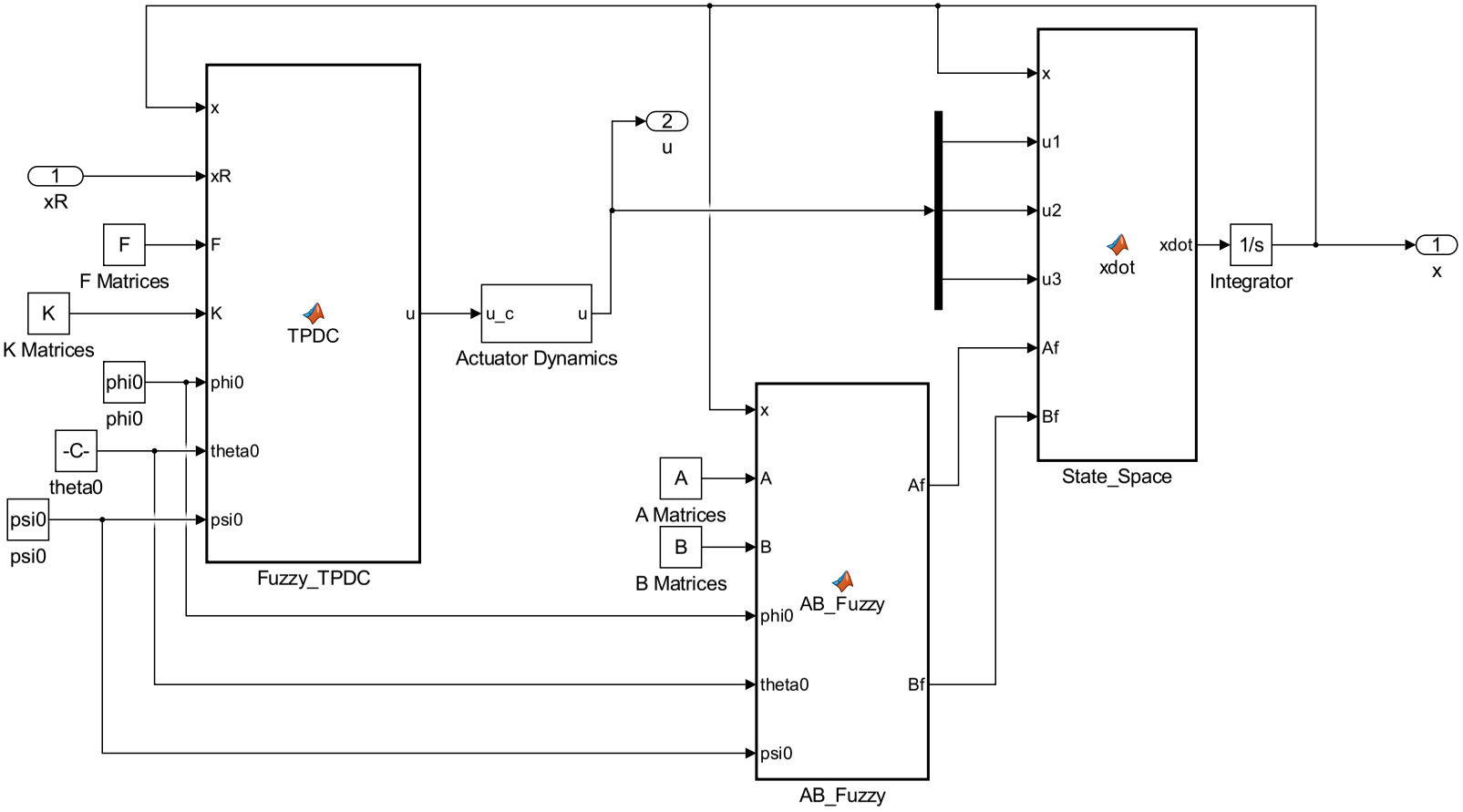
```
figure(1)
% Euler angles
subplot(3,2,1)
plot(time,x(1,:),'LineWidth',2)
ylabel('Roll (deg)')
grid on
hold on
subplot(3,2,3)
plot(time,x(2,:),'LineWidth',2)
ylabel('Pitch (deg)')
grid on
hold on
subplot(3,2,5)
plot(time,x(3,:),'LineWidth',2)
ylabel('Yaw, deg')
xlabel('Time (sec)')
grid on
hold on

% Euler angle derivatives
subplot(3,2,2)
plot(time,x(4,:),'LineWidth',2)
```

```
ylabel('Roll Rate (deg/s)')
grid on
hold on
subplot(3,2,4)
plot(time,x(5,:),'LineWidth',2)
ylabel('Pitch Rate (deg/s)')
grid on
hold on
subplot(3,2,6)
plot(time,x(6,:),'LineWidth',2)
ylabel('Yaw Rate (deg/s)')
xlabel('Time (sec)')
grid on
hold on
clear time x

legend('Class I','Class II', 'Class III', 'Class IV')
```

Published with MATLAB® R2015b

A.8 Closed Loop TPDC T-S Model




x
xR
F
K
phi0
theta0
psi0
Fuzzy_TPDC
TPDC
u
u_c
u
Actuator Dynamics
2
u
1
xR
F
F Matrices
K
K Matrices
phi0
phi0
-C-
theta0
psi0
psi0
x
u1
u2
u3
Af
Bf
State_Space
xdot
xdot
1/s
Integrator
1
x
A
A Matrices
B
B Matrices
x
A
B
phi0
theta0
psi0
AB_Fuzzy
Af
Bf
AB_Fuzzy

This code builds the TPDC in the "Fuzzy_TPDC" box in the SS_TPDC_Txyz.slx file

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

Fuzzy_TPDC

```
function u = TPDC(x,xR,F,K,phi0,theta0,psi0)
%SS_TPDC_Txyz
% This functions builds the input u(t) for the Twin PDC

% ========================================================================
% For the primary PDC
% ========================================================================
[m,n,r]=size(F);

rphi0=length(phi0);
if  rphi0==2
    x11=phi0(1);
    x12=phi0(2);
    mu1=zeros(1,2);
    if x(1)>=x11 & x(1)<=x12
        mu1(1,1)=(x(1)-x12)/(x11-x12);
        mu1(1,2)=(x(1)-x11)/(x12-x11);
    elseif x(1)>x12
        mu1(1,1)=0;
        mu1(1,2)=1;
    else
        mu1(1,1)=1;
        mu1(1,2)=0;
    end
elseif rphi0==3
    x11=phi0(1);
    x12=phi0(2);
    x13=phi0(3);
    mu1=zeros(1,3);
    if x(3)>=x11 & x(3)<=x12
        mu1(1,1)=(x(3)-x12)/(x11-x12);
        mu1(1,2)=(x(3)-x11)/(x12-x11);
        mu1(1,3)=0;
    elseif x(3)>x12 & x(3)<=x13
        mu1(1,1)=0;
        mu1(1,2)=(x(3)-x13)/(x12-x13);
        mu1(1,3)=(x(3)-x12)/(x13-x12);
    elseif x(3)>x13
        mu1(1,1)=0;
        mu1(1,2)=0;
        mu1(1,3)=1;
    elseif x(3)<x11
        mu1(1,1)=1;
        mu1(1,2)=0;
        mu1(1,3)=0;
    end
```

```
end

rtheta0=length(theta0);
if rtheta0==2
    x21=theta0(1);
    x22=theta0(2);
    mu2=zeros(1,2);
    if x(2)>=x21 & x(2)<=x22
        mu2(1,1)=(x(2)-x22)/(x21-x22);
        mu2(1,2)=(x(2)-x21)/(x22-x21);
    elseif x(2)>x22
        mu2(1,1)=0;
        mu2(1,2)=1;
    else
        mu2(1,1)=1;
        mu2(1,2)=0;
    end
elseif rtheta0==3
    x21=theta0(1);
    x22=theta0(2);
    x23=theta0(3);
    mu2=zeros(1,3);
    if x(3)>=x21 & x(3)<=x22
        mu2(1,1)=(x(3)-x22)/(x21-x22);
        mu2(1,2)=(x(3)-x21)/(x22-x21);
        mu2(1,3)=0;
    elseif x(3)>x22 & x(3)<=x23
        mu2(1,1)=0;
        mu2(1,2)=(x(3)-x23)/(x22-x23);
        mu2(1,3)=(x(3)-x22)/(x23-x22);
    elseif x(3)>x23
        mu2(1,1)=0;
        mu2(1,2)=0;
        mu2(1,3)=1;
    elseif x(3)<x21
        mu2(1,1)=1;
        mu2(1,2)=0;
        mu2(1,3)=0;
    end
end

rpsi0=length(psi0);
if rpsi0==2
    x31=psi0(1);
    x32=psi0(2);
    mu3=zeros(1,2);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=-(x(3)-x31)/(x32-x31);
    elseif x(3)>x32
        mu3(1,1)=1;
        mu3(1,2)=0;
    else
        mu3(1,1)=0;
```

```
        mu3(1,2)=1;
    end
elseif rpsi0==3
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    mu3=zeros(1,3);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
    elseif x(3)>x33
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
    end
elseif rpsi0==4
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    mu3=zeros(1,4);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
    elseif x(3)>x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
```

```
        mu3(1,4)=0;
    end
elseif rpsi0==5
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    x35=psi0(5);
    mu3=zeros(1,5);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
        mu3(1,5)=0;
    elseif x(3)>x34 & x(3)<=x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=(x(3)-x35)/(x34-x35);
        mu3(1,5)=(x(3)-x34)/(x35-x34);
    elseif x(3)>x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    end
elseif rpsi0==6
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    x35=psi0(5);
    x36=psi0(6);
    mu3=zeros(1,6);
    if x(3)>=x31 & x(3)<=x32
```

```
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
            mu3(1,4)=(x(3)-x33)/(x34-x33);
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x34 & x(3)<=x35
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=(x(3)-x35)/(x34-x35);
            mu3(1,5)=(x(3)-x34)/(x35-x34);
            mu3(1,6)=0;
        elseif x(3)>x35 & x(3)<=x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=(x(3)-x36)/(x35-x36);
            mu3(1,6)=(x(3)-x35)/(x36-x35);
        elseif x(3)>x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0
            mu3(1,6)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        end
    elseif rpsi0==7
        x31=psi0(1);
        x32=psi0(2);
        x33=psi0(3);
        x34=psi0(4);
```

```
x35=psi0(5);
x36=psi0(6);
x37=psi0(7);
mu3=zeros(1,7);
if x(3)>=x31 & x(3)<=x32
    mu3(1,1)=(x(3)-x32)/(x31-x32);
    mu3(1,2)=(x(3)-x31)/(x32-x31);
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x32 & x(3)<=x33
    mu3(1,1)=0;
    mu3(1,2)=(x(3)-x33)/(x32-x33);
    mu3(1,3)=(x(3)-x32)/(x33-x32);
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x33 & x(3)<=x34
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=(x(3)-x34)/(x33-x34);
    mu3(1,4)=(x(3)-x33)/(x34-x33);
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=(x(3)-x37)/(x36-x37);
    mu3(1,7)=(x(3)-x36)/(x37-x36);
elseif x(3)>x37
    mu3(1,1)=0;
```

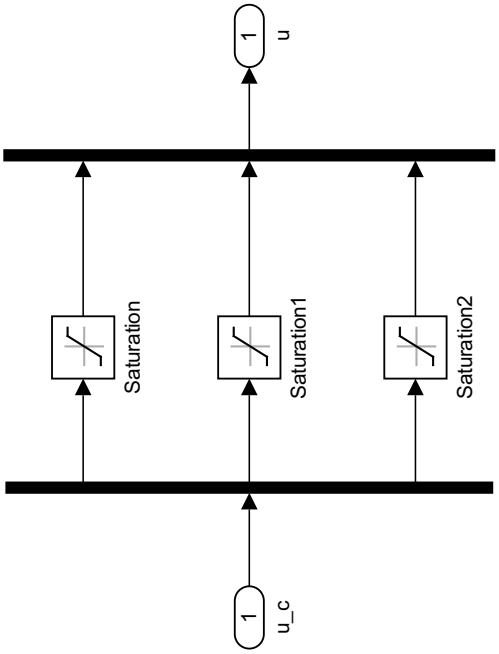
```
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0
            mu3(1,6)=0;
            mu3(1,7)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
        end
    elseif rpsi0==9
        x31=psi0(1);
        x32=psi0(2);
        x33=psi0(3);
        x34=psi0(4);
        x35=psi0(5);
        x36=psi0(6);
        x37=psi0(7);
        x38=psi0(8);
        x39=psi0(9);
        mu3=zeros(1,9);
        if x(3)>=x31 & x(3)<=x32
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
            mu3(1,4)=(x(3)-x33)/(x34-x33);
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
```

```
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=(x(3)-x37)/(x36-x37);
    mu3(1,7)=(x(3)-x36)/(x37-x36);
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x37 & x(3)<=x38
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=(x(3)-x38)/(x37-x38);
    mu3(1,8)=(x(3)-x37)/(x38-x37);
    mu3(1,9)=0;
elseif x(3)>x38 & x(3)<=x39
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=(x(3)-x39)/(x38-x39);
    mu3(1,9)=(x(3)-x38)/(x39-x38);
elseif x(3)>x39
    mu3(1,1)=0;
```

```
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=0;
    end
end

%============================
% CHECK MEMBERSHIP FUNCTIONS %
%============================
% Combining the fuzzy membership functions
Mu={[mu1(1,:)], [mu2(1,:)], [mu3(1,:)]}; % input data: cell array of
 vectors
nvec=numel(Mu); %number of vectors
w=cell(1,nvec); %pre-define to generate comma-separated list
[w{3:-1:1}]=ndgrid(Mu{end:-1:1}); %the reverse order in these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
w = cat(nvec+1, w{:}); %concat the n n-dim arrays along dimension n+1
w = reshape(w,[],nvec); %reshape to obtain desired matrix
w=prod(w')';

% There are 8 rules
uA=zeros(3,r);
h=zeros(size(w));
for i=1:r
    h(i)=w(i)/sum(w);
    uA(:,i)=-h(i).*F(:,:,i)*x;
end
% ========================================================================

% ========================================================================
% For the secondary PDC
% ========================================================================
xR1=xR(1);
xR2=xR(2);

[m,n,rR]=size(K);
```

```
N1=1;
wR=1;

v=wR/sum(wR);
uB=v*K*xR;

% ========================================================================

% ========================================================================
% Combining the inputs uA(t) and uB(t)
% ========================================================================
u=zeros(3,1);
u=sum(uA,2)+sum(uB,2);
% ========================================================================

%SS_TPDC_Txyz
end
```

Published with MATLAB® R2015b


1
u
Saturation
Saturation1
Saturation2
1
u_c

This code builds the fuzzy A and B matrices in the "AB_Fuzzy" box in the SS_TPDC_Txyz.slx file

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

AB_Fuzzy

```
function [Af,Bf] = AB_Fuzzy(x,A,B,phi0,theta0,psi0)
%SS_TPDC_Txyz
% This functions builds the T-S Fuzzy Model

% ======================================================================
% T-S Fuzzy Model
% ======================================================================
[n,n,r]=size(A);

rphi0=length(phi0);
if  rphi0==2
    x11=phi0(1);
    x12=phi0(2);
    mu1=zeros(1,2);
    if x(1)>=x11 & x(1)<=x12
        mu1(1,1)=(x(1)-x12)/(x11-x12);
        mu1(1,2)=(x(1)-x11)/(x12-x11);
    elseif x(1)>x12
        mu1(1,1)=0;
        mu1(1,2)=1;
    else
        mu1(1,1)=1;
        mu1(1,2)=0;
    end
elseif rphi0==3
    x11=phi0(1);
    x12=phi0(2);
    x13=phi0(3);
    mu1=zeros(1,3);
    if x(3)>=x11 & x(3)<=x12
        mu1(1,1)=(x(3)-x12)/(x11-x12);
        mu1(1,2)=(x(3)-x11)/(x12-x11);
        mu1(1,3)=0;
    elseif x(3)>x12 & x(3)<=x13
        mu1(1,1)=0;
        mu1(1,2)=(x(3)-x13)/(x12-x13);
        mu1(1,3)=(x(3)-x12)/(x13-x12);
    elseif x(3)>x13
        mu1(1,1)=0;
        mu1(1,2)=0;
        mu1(1,3)=1;
    elseif x(3)<x11
        mu1(1,1)=1;
        mu1(1,2)=0;
        mu1(1,3)=0;
    end
```

```
end

rtheta0=length(theta0);
if rtheta0==2
    x21=theta0(1);
    x22=theta0(2);
    mu2=zeros(1,2);
    if x(2)>=x21 & x(2)<=x22
        mu2(1,1)=(x(2)-x22)/(x21-x22);
        mu2(1,2)=(x(2)-x21)/(x22-x21);
    elseif x(2)>x22
        mu2(1,1)=0;
        mu2(1,2)=1;
    else
        mu2(1,1)=1;
        mu2(1,2)=0;
    end
elseif rtheta0==3
    x21=theta0(1);
    x22=theta0(2);
    x23=theta0(3);
    mu2=zeros(1,3);
    if x(3)>=x21 & x(3)<=x22
        mu2(1,1)=(x(3)-x22)/(x21-x22);
        mu2(1,2)=(x(3)-x21)/(x22-x21);
        mu2(1,3)=0;
    elseif x(3)>x22 & x(3)<=x23
        mu2(1,1)=0;
        mu2(1,2)=(x(3)-x23)/(x22-x23);
        mu2(1,3)=(x(3)-x22)/(x23-x22);
    elseif x(3)>x23
        mu2(1,1)=0;
        mu2(1,2)=0;
        mu2(1,3)=1;
    elseif x(3)<x21
        mu2(1,1)=1;
        mu2(1,2)=0;
        mu2(1,3)=0;
    end
end

rpsi0=length(psi0);
if rpsi0==2
    x31=psi0(1);
    x32=psi0(2);
    mu3=zeros(1,2);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=-(x(3)-x31)/(x32-x31);
    elseif x(3)>x32
        mu3(1,1)=1;
        mu3(1,2)=0;
    else
        mu3(1,1)=0;
```

```
        mu3(1,2)=1;
    end
elseif rpsi0==3
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    mu3=zeros(1,3);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
    elseif x(3)>x33
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
    end
elseif rpsi0==4
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    mu3=zeros(1,4);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
    elseif x(3)>x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
```

```
        mu3(1,4)=0;
    end
elseif rpsi0==5
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    x35=psi0(5);
    mu3=zeros(1,5);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
        mu3(1,5)=0;
    elseif x(3)>x34 & x(3)<=x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=(x(3)-x35)/(x34-x35);
        mu3(1,5)=(x(3)-x34)/(x35-x34);
    elseif x(3)>x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    end
elseif rpsi0==6
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    x35=psi0(5);
    x36=psi0(6);
    mu3=zeros(1,6);
    if x(3)>=x31 & x(3)<=x32
```

```
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
            mu3(1,4)=(x(3)-x33)/(x34-x33);
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x34 & x(3)<=x35
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=(x(3)-x35)/(x34-x35);
            mu3(1,5)=(x(3)-x34)/(x35-x34);
            mu3(1,6)=0;
        elseif x(3)>x35 & x(3)<=x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=(x(3)-x36)/(x35-x36);
            mu3(1,6)=(x(3)-x35)/(x36-x35);
        elseif x(3)>x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0
            mu3(1,6)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        end
    elseif rpsi0==7
        x31=psi0(1);
        x32=psi0(2);
        x33=psi0(3);
        x34=psi0(4);
```

```
x35=psi0(5);
x36=psi0(6);
x37=psi0(7);
mu3=zeros(1,7);
if x(3)>=x31 & x(3)<=x32
    mu3(1,1)=(x(3)-x32)/(x31-x32);
    mu3(1,2)=(x(3)-x31)/(x32-x31);
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x32 & x(3)<=x33
    mu3(1,1)=0;
    mu3(1,2)=(x(3)-x33)/(x32-x33);
    mu3(1,3)=(x(3)-x32)/(x33-x32);
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x33 & x(3)<=x34
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=(x(3)-x34)/(x33-x34);
    mu3(1,4)=(x(3)-x33)/(x34-x33);
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=(x(3)-x37)/(x36-x37);
    mu3(1,7)=(x(3)-x36)/(x37-x36);
elseif x(3)>x37
    mu3(1,1)=0;
```

```
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0
            mu3(1,6)=0;
            mu3(1,7)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
        end
    elseif rpsi0==9
        x31=psi0(1);
        x32=psi0(2);
        x33=psi0(3);
        x34=psi0(4);
        x35=psi0(5);
        x36=psi0(6);
        x37=psi0(7);
        x38=psi0(8);
        x39=psi0(9);
        mu3=zeros(1,9);
        if x(3)>=x31 & x(3)<=x32
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
            mu3(1,4)=(x(3)-x33)/(x34-x33);
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
```

```
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=(x(3)-x37)/(x36-x37);
    mu3(1,7)=(x(3)-x36)/(x37-x36);
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x37 & x(3)<=x38
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=(x(3)-x38)/(x37-x38);
    mu3(1,8)=(x(3)-x37)/(x38-x37);
    mu3(1,9)=0;
elseif x(3)>x38 & x(3)<=x39
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=(x(3)-x39)/(x38-x39);
    mu3(1,9)=(x(3)-x38)/(x39-x38);
elseif x(3)>x39
    mu3(1,1)=0;
```

```
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=0;
    end
end

%============================
% CHECK MEMBERSHIP FUNCTIONS %
%============================
% Combining the fuzzy membership functions
Mu={[mu1(1,:)], [mu2(1,:)], [mu3(1,:)]}; % input data: cell array of
 vectors
nvec=numel(Mu); %number of vectors
w=cell(1,nvec); %pre-define to generate comma-separated list
[w{3:-1:1}]=ndgrid(Mu{end:-1:1}); %the reverse order in these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
w = cat(nvec+1, w{:}); %concat the n n-dim arrays along dimension n+1
w = reshape(w,[],nvec); %reshape to obtain desired matrix
w=prod(w')';

h=zeros(size(w));
Afi=zeros(size(A));
Bfi=zeros(size(B));
[n,n,r]=size(A);
for i=1:r
    h(i)=w(i)/sum(w);
    Afi(:,:,i)=h(i).*A(:,:,i);
    Bfi(:,:,i)=h(i).*B(:,:,i);
end
Af=sum(Afi,3);
Bf=sum(Bfi,3);
% ========================================================================

%SS_TPDC_Txyz
end
```

Published with MATLAB® R2015b

This code builds state space model in the "State_Space" box in the SS_TPDC_Txyz.slx file

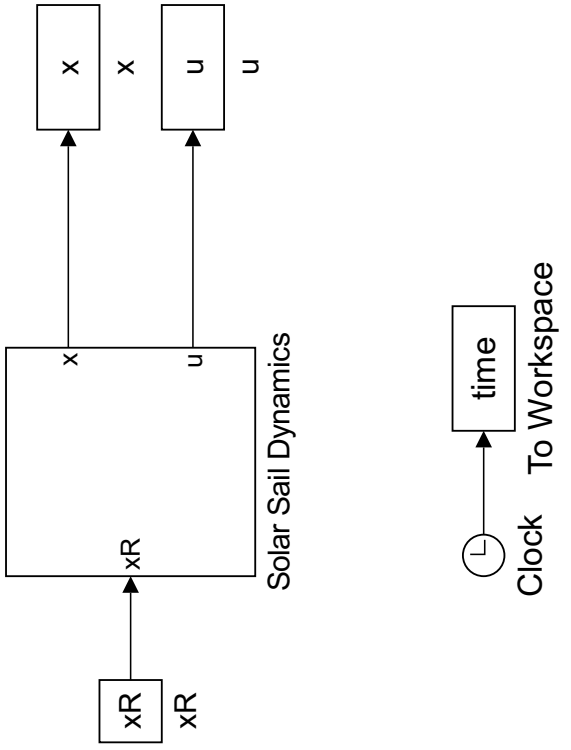
```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

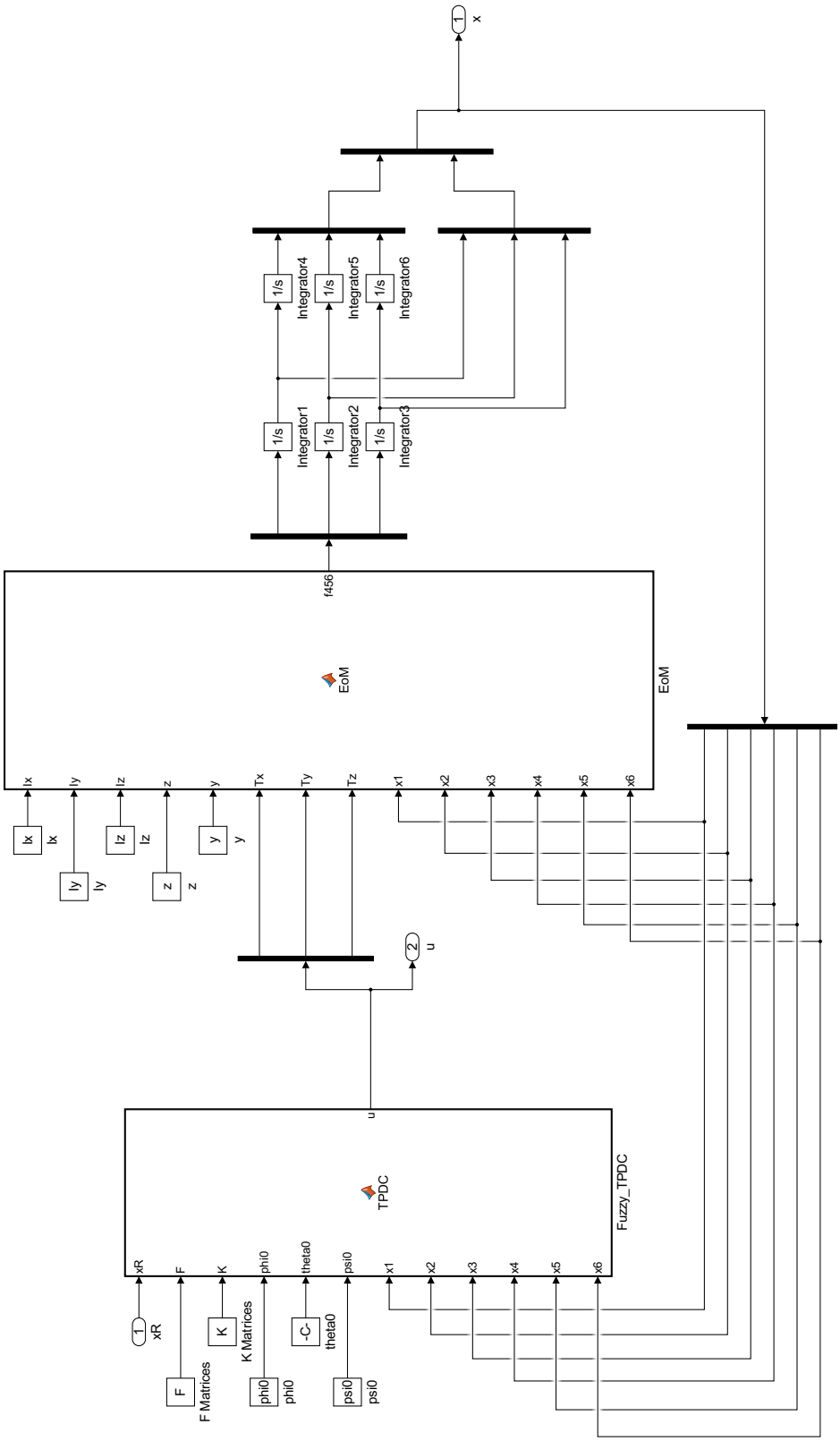
State_Space

```
function xdot= xdot(x,u1,u2,u3,Af,Bf)
%SS_TPDC_Txyz

% Combining u1, u2, and u3 into u vector
u=[u1 u2 u3]';

[n,n,r]=size(Af);

xdot=zeros(n,1);
xdot=Af*x+Bf*u;

%SS_TPDC_Txyz
end
```

Published with MATLAB® R2015b

A.9 Closed Loop TPDC Nonlinear Model




1
x
1/s
Integrator4
1/s
Integrator5
1/s
Integrator6
1/s
Integrator1
1/s
Integrator2
1/s
Integrator3
f456
EoM
EoM
lx
ly
lz
z
y
Tx
Ty
Tz
x1
x2
x3
x4
x5
x6
lx
ly
lz
z
y
2
u
u
TPDC
Fuzzy_TPDC
xR
F
K
phi0
theta0
psi0
x1
x2
x3
x4
x5
x6
1
xR
K
K Matrices
-C-
theta0
F
F Matrices
phi0
phi0
psi0
psi0

This code builds the TPDC in the "Fuzzy_TPDC" box in the SS_NL_TPDC_Txyz.slx file

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

Fuzzy_TPDC

```
function u = TPDC(xR,F,K,phi0,theta0,psi0,x1,x2,x3,x4,x5,x6)
%SS_NL_TPDC_Txyz
% This functions builds the input u(t) for the Twin PDC

% ======================================================================
% For the primary PDC
% ======================================================================
x=[x1 x2 x3 x4 x5 x6]';
[m,n,r]=size(F);
rphi0=length(phi0);
if  rphi0==2
    x11=phi0(1);
    x12=phi0(2);
    mu1=zeros(1,2);
    if x(1)>=x11 & x(1)<=x12
        mu1(1,1)=(x(1)-x12)/(x11-x12);
        mu1(1,2)=(x(1)-x11)/(x12-x11);
    elseif x(1)>x12
        mu1(1,1)=0;
        mu1(1,2)=1;
    else
        mu1(1,1)=1;
        mu1(1,2)=0;
    end
elseif rphi0==3
    x11=phi0(1);
    x12=phi0(2);
    x13=phi0(3);
    mu1=zeros(1,3);
    if x(3)>=x11 & x(3)<=x12
        mu1(1,1)=(x(3)-x12)/(x11-x12);
        mu1(1,2)=(x(3)-x11)/(x12-x11);
        mu1(1,3)=0;
    elseif x(3)>x12 & x(3)<=x13
        mu1(1,1)=0;
        mu1(1,2)=(x(3)-x13)/(x12-x13);
        mu1(1,3)=(x(3)-x12)/(x13-x12);
    elseif x(3)>x13
        mu1(1,1)=0;
        mu1(1,2)=0;
        mu1(1,3)=1;
    elseif x(3)<x11
        mu1(1,1)=1;
        mu1(1,2)=0;
        mu1(1,3)=0;
    end
```

```
end

rtheta0=length(theta0);
if rtheta0==2
    x21=theta0(1);
    x22=theta0(2);
    mu2=zeros(1,2);
    if x(2)>=x21 & x(2)<=x22
        mu2(1,1)=(x(2)-x22)/(x21-x22);
        mu2(1,2)=(x(2)-x21)/(x22-x21);
    elseif x(2)>x22
        mu2(1,1)=0;
        mu2(1,2)=1;
    else
        mu2(1,1)=1;
        mu2(1,2)=0;
    end
elseif rtheta0==3
    x21=theta0(1);
    x22=theta0(2);
    x23=theta0(3);
    mu2=zeros(1,3);
    if x(3)>=x21 & x(3)<=x22
        mu2(1,1)=(x(3)-x22)/(x21-x22);
        mu2(1,2)=(x(3)-x21)/(x22-x21);
        mu2(1,3)=0;
    elseif x(3)>x22 & x(3)<=x23
        mu2(1,1)=0;
        mu2(1,2)=(x(3)-x23)/(x22-x23);
        mu2(1,3)=(x(3)-x22)/(x23-x22);
    elseif x(3)>x23
        mu2(1,1)=0;
        mu2(1,2)=0;
        mu2(1,3)=1;
    elseif x(3)<x21
        mu2(1,1)=1;
        mu2(1,2)=0;
        mu2(1,3)=0;
    end
end

rpsi0=length(psi0);
if rpsi0==2
    x31=psi0(1);
    x32=psi0(2);
    mu3=zeros(1,2);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=-(x(3)-x31)/(x32-x31);
    elseif x(3)>x32
        mu3(1,1)=1;
        mu3(1,2)=0;
    else
        mu3(1,1)=0;
```

```
        mu3(1,2)=1;
    end
elseif rpsi0==3
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    mu3=zeros(1,3);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
    elseif x(3)>x33
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
    end
elseif rpsi0==4
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    mu3=zeros(1,4);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
    elseif x(3)>x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
```

```
        mu3(1,4)=0;
    end
elseif rpsi0==5
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    x35=psi0(5);
    mu3=zeros(1,5);
    if x(3)>=x31 & x(3)<=x32
        mu3(1,1)=(x(3)-x32)/(x31-x32);
        mu3(1,2)=(x(3)-x31)/(x32-x31);
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x32 & x(3)<=x33
        mu3(1,1)=0;
        mu3(1,2)=(x(3)-x33)/(x32-x33);
        mu3(1,3)=(x(3)-x32)/(x33-x32);
        mu3(1,4)=0;
        mu3(1,5)=0;
    elseif x(3)>x33 & x(3)<=x34
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=(x(3)-x34)/(x33-x34);
        mu3(1,4)=(x(3)-x33)/(x34-x33);
        mu3(1,5)=0;
    elseif x(3)>x34 & x(3)<=x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=(x(3)-x35)/(x34-x35);
        mu3(1,5)=(x(3)-x34)/(x35-x34);
    elseif x(3)>x35
        mu3(1,1)=0;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
    end
elseif rpsi0==6
    x31=psi0(1);
    x32=psi0(2);
    x33=psi0(3);
    x34=psi0(4);
    x35=psi0(5);
    x36=psi0(6);
    mu3=zeros(1,6);
    if x(3)>=x31 & x(3)<=x32
```

```
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
            mu3(1,4)=(x(3)-x33)/(x34-x33);
            mu3(1,5)=0;
            mu3(1,6)=0;
        elseif x(3)>x34 & x(3)<=x35
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=(x(3)-x35)/(x34-x35);
            mu3(1,5)=(x(3)-x34)/(x35-x34);
            mu3(1,6)=0;
        elseif x(3)>x35 & x(3)<=x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=(x(3)-x36)/(x35-x36);
            mu3(1,6)=(x(3)-x35)/(x36-x35);
        elseif x(3)>x36
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0
            mu3(1,6)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
        end
    elseif rpsi0==7
        x31=psi0(1);
        x32=psi0(2);
        x33=psi0(3);
        x34=psi0(4);
```

```
x35=psi0(5);
x36=psi0(6);
x37=psi0(7);
mu3=zeros(1,7);
if x(3)>=x31 & x(3)<=x32
    mu3(1,1)=(x(3)-x32)/(x31-x32);
    mu3(1,2)=(x(3)-x31)/(x32-x31);
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x32 & x(3)<=x33
    mu3(1,1)=0;
    mu3(1,2)=(x(3)-x33)/(x32-x33);
    mu3(1,3)=(x(3)-x32)/(x33-x32);
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x33 & x(3)<=x34
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=(x(3)-x34)/(x33-x34);
    mu3(1,4)=(x(3)-x33)/(x34-x33);
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=(x(3)-x37)/(x36-x37);
    mu3(1,7)=(x(3)-x36)/(x37-x36);
elseif x(3)>x37
    mu3(1,1)=0;
```

```
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0
            mu3(1,6)=0;
            mu3(1,7)=1;
        elseif x(3)<x31
            mu3(1,1)=1;
            mu3(1,2)=0;
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
        end
    elseif rpsi0==9
        x31=psi0(1);
        x32=psi0(2);
        x33=psi0(3);
        x34=psi0(4);
        x35=psi0(5);
        x36=psi0(6);
        x37=psi0(7);
        x38=psi0(8);
        x39=psi0(9);
        mu3=zeros(1,9);
        if x(3)>=x31 & x(3)<=x32
            mu3(1,1)=(x(3)-x32)/(x31-x32);
            mu3(1,2)=(x(3)-x31)/(x32-x31);
            mu3(1,3)=0;
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x32 & x(3)<=x33
            mu3(1,1)=0;
            mu3(1,2)=(x(3)-x33)/(x32-x33);
            mu3(1,3)=(x(3)-x32)/(x33-x32);
            mu3(1,4)=0;
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
            mu3(1,8)=0;
            mu3(1,9)=0;
        elseif x(3)>x33 & x(3)<=x34
            mu3(1,1)=0;
            mu3(1,2)=0;
            mu3(1,3)=(x(3)-x34)/(x33-x34);
            mu3(1,4)=(x(3)-x33)/(x34-x33);
            mu3(1,5)=0;
            mu3(1,6)=0;
            mu3(1,7)=0;
```

```
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x34 & x(3)<=x35
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=(x(3)-x35)/(x34-x35);
    mu3(1,5)=(x(3)-x34)/(x35-x34);
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x35 & x(3)<=x36
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=(x(3)-x36)/(x35-x36);
    mu3(1,6)=(x(3)-x35)/(x36-x35);
    mu3(1,7)=0;
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x36 & x(3)<=x37
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=(x(3)-x37)/(x36-x37);
    mu3(1,7)=(x(3)-x36)/(x37-x36);
    mu3(1,8)=0;
    mu3(1,9)=0;
elseif x(3)>x37 & x(3)<=x38
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=(x(3)-x38)/(x37-x38);
    mu3(1,8)=(x(3)-x37)/(x38-x37);
    mu3(1,9)=0;
elseif x(3)>x38 & x(3)<=x39
    mu3(1,1)=0;
    mu3(1,2)=0;
    mu3(1,3)=0;
    mu3(1,4)=0;
    mu3(1,5)=0;
    mu3(1,6)=0;
    mu3(1,7)=0;
    mu3(1,8)=(x(3)-x39)/(x38-x39);
    mu3(1,9)=(x(3)-x38)/(x39-x38);
elseif x(3)>x39
    mu3(1,1)=0;
```

```
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=1;
    elseif x(3)<x31
        mu3(1,1)=1;
        mu3(1,2)=0;
        mu3(1,3)=0;
        mu3(1,4)=0;
        mu3(1,5)=0;
        mu3(1,6)=0;
        mu3(1,7)=0;
        mu3(1,8)=0;
        mu3(1,9)=0;
    end
end
%============================
% CHECK MEMBERSHIP FUNCTIONS %
%============================
% Combining the fuzzy membership functions
Mu={[mu1(1,:)], [mu2(1,:)], [mu3(1,:)]}; % input data: cell array of
 vectors
nvec=numel(Mu); %number of vectors
w=cell(1,nvec); %pre-define to generate comma-separated list
[w{3:-1:1}]=ndgrid(Mu{end:-1:1}); %the reverse order in these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
w = cat(nvec+1, w{:}); %concat the n n-dim arrays along dimension n+1
w = reshape(w,[],nvec); %reshape to obtain desired matrix
w=prod(w')';

[M,n,r]=size(F);
uA=zeros(3,r);
h=zeros(size(w));
for i=1:r
    h(i)=w(i)/sum(w);
    uA(:,i)=-h(i).*F(:,:,i)*x;
end
% ========================================================================

% ========================================================================
% For the secondary PDC
% ========================================================================
[m,n,rR]=size(K);

N1=1;
wR=1;

v=wR/sum(wR);
```

```
uB=v*K*xR;
% =========================================================================

% =========================================================================
% Combining the inputs uA(t) and uB(t)
% =========================================================================
u=zeros(3,1);
u=sum(uA,2)+sum(uB,2);
% =========================================================================

%SS_NL_TPDC_Txyz
end
```

Published with MATLAB® R2015b

This code consists of the equations of motion used in the "EoM" box in the SS_NL_TPDC_Txyz.slx file

```
%  Written by: Joshua Baculi
%  Last Edited: October 20, 2016
```

EoM

```
function f456 = EoM(Ix,Iy,Iz,z,y,Tx,Ty,Tz,x1,x2,x3,x4,x5,x6)
%SS_NL_TPDC_Txyz
%=========================================================================
% Parameters
%=========================================================================
%...Mass and torque properties for a 40m solar sail (pg 781)
sail_size=40; %m %Sail size = 40m x 40m
sf=75; %percent %Scallop factor
Area=1600; %m^2 % Sail area
Fs=0.01; %N %Sail thrust force (eta*P*A)
% Ix=4340; %kg-m^2
% Iy=2171; %kg-m^2
% Iz=2171; %kg-m^2
epsilon=0.1; %m %cp-cp offset
Tpy=1.0; %mN*m %Pitch/yaw solar disturbance torque
Tr=0.5; %mN*m %Roll solar disturbance torque
%...End mass and torque propertis from pg 781

%...Control parameters for a 40m solar sail (pg 795)
m=1; %kg %Trim control mass (TCM)
M=148; %kg %Main-body mass
v_TCM=0.05; %m/s %TCM speed limit
y_max=28; %m %TCM y_max=+-28 m
z_max=y_max;
y_ss=14.9; %m %Steady-state trim value to counter epsilon
z_ss=y_ss; %m
T=560; %s %Actuator time constant
%...End control parameters from pg 795

%...Roll control parameters for a 1m RSB (pg 797)
Theta_max=45*pi/180; %rad % RSB max deflection angle=+-45 deg
l_RSB=1; %m %RSB moment arm length
T_max=(0.5/20)*13.3*Fs*sin(Theta_max);
%...End roll control parameters from pg 797

%...Parameters from pg 805 of the book
omega_max=0.05*pi/180; %rad
T_max=(0.5/20)*13.3*Fs*sin(Theta_max); %Nm
%...End parameters from pg 805

%...end parameters

% Other values
mr=m*(M+m)/(M+2*m); %kg %Reduced mass
n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer orbit
 (SSTO)
```

```
                    %Value for this mission taken from pg 762
P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1 AU
 from
                    %the sun (pg. 793)
%=========================================================================

%=========================================================================
% Equations
%=========================================================================
% Calculating the J's
Jx=Ix+mr*(y^2+z^2);
Jy=Iy+mr*z^2;
Jz=Iz+mr*y^2;

f1=x4;
f2=x5;
f3=x6;
f4=Tx/Jx - ((Jy-Jz)*(n^2)*(3+cos(x3)^2)*x1)/Jx + ((Jy-
Jz)*(n^2)*cos(x3)*sin(x3)*x2)/Jx + ((Jx-Jy+Jz)*n*cos(x3)*x6)/Jx +
 (0.5*Fs*epsilon*sin(x3)^2)/Jx;
f5=Ty/Jy - ((Jx-Jz)*(n^2)*(3+sin(x3)^2)*x2)/Jy + ((Jx-
Jz)*(n^2)*cos(x3)*sin(x3)*x1)/Jy + ((Jx-Jy-Jz)*n*sin(x3)*x6)/Jy +
 (m*Fs*z*sin(x3)^2)/((2*m+M)*Jy)+ (Fs*epsilon*sin(x3)^2)/Jy;
f6=Tz/Jz - ((-Jx+Jy)*(n^2)*cos(x3)*sin(x3))/Jz - ((Jx-Jy
+Jz)*n*cos(x3)*x4)/Jz - ((Jx-Jy-Jz)*n*sin(x3)*x5)/Jz -
 (m*Fs*y*sin(x3)^2)/((2*m+M)*Jz) + (Fs*epsilon*sin(x3)^2)/Jz;

f456=[f4 f5 f6]';

%SS_NL_TPDC_Txyz
end
```

Published with MATLAB® R2015b

Appendix B

Fuzzy Logic Supervisory PID Code

B.1 PSO Iteration Code

This code takes a previously written code for PID particle swarm optimization and implements it on a solar sail with sliding masses control inputs

```
% Solar Sail Code Written by: Joshua Baculi
% PSO Code Written by: Wael Mansour (wael192@yahoo.com)
% Last Edited: August 15, 2016

% Necessary sub-file: tracklsq_SS_yz.m
```

Initialization

```
clear
clc
%...Mass and torque properties for a 40m solar sail (pg 781)
sail_size=40; %m %Sail size = 40m x 40m
sf=75; %percent %Scallop factor
Area=1600; %m^2 % Sail area
Fs=0.01; %N %Sail thrust force (eta*P*A)
Ix=4340; %kg-m^2
Iy=2171; %kg-m^2
Iz=2171; %kg-m^2
epsilon=0.1; %m %cp-cp offset
Tpy=1.0; %mN*m %Pitch/yaw solar disturbance torque
Tr=0.5; %mN*m %Roll solar disturbance torque
%...End mass and torque propertis from pg 781

%...Control parameters for a 40m solar sail (pg 795)
m=1; %kg %Trim control mass (TCM)
M=148; %kg %Main-body mass
v_TCM=0.05; %m/s %TCM speed limit
y_max=28; %m %TCM y_max=+-28 m
z_max=y_max;
y_ss=14.9; %m %Steady-state trim value to counter epsilon
z_ss=-y_ss; %m
T=560; %s %Actuator time constant
%...End control parameters from pg 795

%...Roll control parameters for a 1m RSB (pg 797)
Theta_max=45*pi/180; %rad % RSB max deflection angle=+-45 deg
l_RSB=1; %m %RSB moment arm length
T_max=(0.5/20)*13.3*Fs*sin(Theta_max);
%...End roll control parameters from pg 797

%...Parameters from pg 805 of the book
omega_max=0.05*pi/180; %rad
T_max=(0.5/20)*13.3*Fs*sin(Theta_max); %Nm
%...End parameters from pg 805

%...end parameters

% Other values
mr=m*(M+m)/(M+2*m); %kg %Reduced mass
```

```
n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer orbit
 (SSTO)
                    %Value for this mission taken from pg 762
P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1 AU
 from
                    %the sun (pg. 793)

% Initial and desired states
x0=[5 -5 -90 0 0 0]'*pi/180;
xR=[0 0 -60 0 0 0]'*pi/180;
phi_c=xR(1);
theta_c=xR(2);
psi_c=xR(3);

% Simulation duration
T=3600*4; %sec
Tsim=0;

% % Gains from Ziegler-Nichols to be used as the intial gain position
% % Stabilizes to [0 0 -55]
K_PSO=[0.000000600000000
   0.000345600000000
   0.000000000260417
   0.003000000000000
   7.128000000000000
   0.000000315656566
   0.003000000000000
   7.776000000000000
   0.000000289351852]*10^4;

KpTx=K_PSO(1);
KdTx=K_PSO(2);
KiTx=K_PSO(3);
Kpz=K_PSO(4);
Kdz=K_PSO(5);
Kiz=K_PSO(6);
Kpy=K_PSO(7);
Kdy=K_PSO(8);
Kiy=K_PSO(9);
```

PSO Initial Iteration

```
n=50;           % Size of the swarm " no of birds "
bird_step=250;   % Maximum number of "birds steps"
dim=9;          % Dimension of the problem (3 gains for 3 states)

% These parameters obtained from "Tuning of PID Controller Using
 Particle
% Swarm Optimization (PSO)" paper  presented at Internatioal
 Scientific
% Conference 2011
c2=1.494;          % PSO parameter C1
c1=1.494;        % PSO parameter C2
w=0.729;           % pso momentum or inertia
```

```
fitness=0*ones(n,bird_step);

%-----------------------------%
%    initialize the parameter %
%-----------------------------%
R1=rand(dim, n); % Velocity coefficient to be used later on
R2=rand(dim, n); % Velocity coefficient to be used later on
current_fitness=0*ones(n,1); %initialize vector to be updated in
 "Evaluate initial population"

%-------------------------------------------------%
% Initializing swarm and velocities and position %
%-------------------------------------------------%
% Matric of Z-N gains to be used for initial position
Kpid=[];
for i=1:n
    Kpid(:,i)=[KpTx KdTx KiTx Kpz Kdz Kiz Kpy Kdy Kiy]';
end

current_position = Kpid; % Initial position
velocity = Kpid.*10.*randn(dim, n); % Initial velocity
local_best_position = current_position; % Because initial, also local
 best

%-----------------------------------------%
%     Evaluate initial population           %
%-----------------------------------------%
for i = 1:n
    current_fitness(i) = tracklsq_SS_yz(current_position(:,i),x0,xR);
end

local_best_fitness = current_fitness ; % Because intial, also local
 best
[global_best_fitness,g] = min(local_best_fitness) ;
% Global best is the best for each input
% dim(global_best_fitness)=3
global_best_fitness

global_best_position = zeros(size(local_best_position));
for i=1:n
    global_best_position(:,i)=local_best_position(:,g);
end

%-------------------%
%  VELOCITY UPDATE  %
%-------------------%
velocity = w *velocity + c1*(R1.*(local_best_position-
current_position)) + c2*(R2.*(global_best_position-current_position));

%------------------%
%   SWARMUPDATE    %
%------------------%
current_position = current_position + velocity ;
```

```
%------------------------%
%  evaluate a new swarm   %
%------------------------%
```

Main PSO Loop

```
iter=0 ;        % Iterations'counter
while (iter<bird_step)
    iter = iter + 1

    R1=rand(dim, n); % Velocity coefficient to be used later on
    R2=rand(dim, n); % Velocity coefficient to be used later on

    for i=1:n,
        current_fitness(i) =
 mean(tracklsq_SS_yz(current_position(:,i),x0,xR));
%         F=tracklsq_SS_yz_yz(current_position(:,i)); % Fitness for
 each input (dim(F)=3)
%         for f=1:3
%             current_fitness(i,f) = F(f); % Turning F into a matrix
 for each bird
%         end
    end

    for i = 1 : n
        % Seeing if new current fitness is better (less than) the
 previous
        % fitness (local_best_fitness)
        if current_fitness(i) < local_best_fitness(i)
           local_best_fitness(i)  = current_fitness(i);
           local_best_position(:,i) = current_position(:,i);
        end
        % If it isn't better, than the ith bird of local_best_fitness
 and
        % local_best_position is unchanged
    end

    % Checking again for the minimum fitness
    [current_global_best_fitness,g] = min(local_best_fitness);
    if current_global_best_fitness < global_best_fitness
       global_best_fitness = current_global_best_fitness;
        for i=1:n
            global_best_position(:,i) = local_best_position(:,g);
        end
    end

     velocity = w *velocity + c1*(R1.*(local_best_position-
current_position)) + c2*(R2.*(global_best_position-current_position));
     current_position = current_position + velocity;

     sprintf('The value of interation iter %3.0f ', iter );
     global_best_fitness
```

```
%      if global_best_fitness<=10^4
%          break
%      end
end % end of while loop its mean the end of all step that the birds
 move it

% xx=current_fitness(:,bird_step);
[Y,I] = min(current_fitness);
K_PSO=current_position(:,I)
```

Published with MATLAB® R2015b

B.2 PSO Cost Function

This code calculates the cost function for the PSO code in the PSO_SS_yz.m file

```
% Original Cost Function Code Written by: Wael Mansour
 (wael192@yahoo.com)
% Edited by: Joshua Baculi
% Last Edited: August 15, 2016
```

Cost Function

```
function F = tracklsq(pid,x0,xR)
    %...Mass and torque properties for a 40m solar sail (pg 781)
    sail_size=40; %m %Sail size = 40m x 40m
    sf=75; %percent %Scallop factor
    Area=1600; %m^2 % Sail area
    Fs=0.01; %N %Sail thrust force (eta*P*A)
    Ix=4340; %kg-m^2
    Iy=2171; %kg-m^2
    Iz=2171; %kg-m^2
    epsilon=0.1; %m %cp-cp offset
    Tpy=1.0; %mN*m %Pitch/yaw solar disturbance torque
    Tr=0.5; %mN*m %Roll solar disturbance torque
    %...End mass and torque propertis from pg 781

    %...Control parameters for a 40m solar sail (pg 795)
    m=1; %kg %Trim control mass (TCM)
    M=148; %kg %Main-body mass
    v_TCM=0.05; %m/s %TCM speed limit
    y_max=28; %m %TCM y_max=+-28 m
    z_max=y_max;
    y_ss=14.9; %m %Steady-state trim value to counter epsilon
    z_ss=-y_ss; %m
    T=560; %s %Actuator time constant
    %...End control parameters from pg 795

    %...Roll control parameters for a 1m RSB (pg 797)
    Theta_max=45*pi/180; %rad % RSB max deflection angle=+-45 deg
    l_RSB=1; %m %RSB moment arm length
    T_max=(0.5/20)*13.3*Fs*sin(Theta_max);
    %...End roll control parameters from pg 797

    %...Parameters from pg 805 of the book
    omega_max=0.05*pi/180; %rad
    T_max=(0.5/20)*13.3*Fs*sin(Theta_max); %Nm
    %...End parameters from pg 805

    %...end parameters

    % Other values
    mr=m*(M+m)/(M+2*m); %kg %Reduced mass
    n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer
 orbit (SSTO)
                        %Value for this mission taken from pg 762
```

```
    P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1
AU from
                        %the sun (pg. 793)

    % Variables a1 and a2 are shared with RUNTRACKLSQ
    KpTx = pid(1);
    KdTx = pid(2);
    KiTx = pid(3);
    Kpz = pid(4);
    Kdz = pid(5);
    Kiz = pid(6);
    Kpy = pid(7);
    Kdy = pid(8);
    Kiy = pid(9);

    % Compute function value
    simopt =
simset('solver','ode5','SrcWorkspace','Current','DstWorkspace','Current');
 % Initialize sim options
    [tout,xout,yout] = sim('PSO_PID_Control_yz',[0 6*3600],simopt);

    % Need to only look at the final ITAE value
    ITAE_roll=trapz(time,time.*abs(error_roll));
    ITAE_pitch=trapz(time,time.*abs(error_pitch));
    ITAE_yaw=trapz(time,time.*abs(error_yaw));

    % Array of the cost functions
    Fang=[ITAE_roll ITAE_pitch ITAE_yaw];

    % Taking the average of the cost functions so that the gains will
be
    % optimized dependently together
    F=mean(Fang);

    end
```

Published with MATLAB® R2015b

B.3 FLS & PID Closed Loop Plotting

This code calls the PID simulink model 'SPID_Control' to plot the inputs Tx, y, and z as well as the roll, pitch, yaw responses

```
% Written by: Joshua Baculi
% Last Edited: August 15, 2016
```

Initialize parameters

```
clear,clc
close all
%...Mass and torque properties for a 40m solar sail (pg 781)
sail_size=40; %m %Sail size = 40m x 40m
sf=75; %percent %Scallop factor
Area=1600; %m^2 % Sail area
Fs=0.01; %N %Sail thrust force (eta*P*A)
Ix=4340; %kg-m^2
Iy=2171; %kg-m^2
Iz=2171; %kg-m^2
epsilon=0.1; %m %cp-cp offset
Tpy=1.0; %mN*m %Pitch/yaw solar disturbance torque
Tr=0.5; %mN*m %Roll solar disturbance torque
%...End mass and torque propertis from pg 781

%...Control parameters for a 40m solar sail (pg 795)
m=1; %kg %Trim control mass (TCM)
M=148; %kg %Main-body mass
v_TCM=0.05; %m/s %TCM speed limit
y_max=28; %m %TCM y_max=+-28 m
z_max=y_max;
y_ss=14.9; %m %Steady-state trim value to counter epsilon
z_ss=-y_ss; %m
T=560; %s %Actuator time constant
%...End control parameters from pg 795

%...Roll control parameters for a 1m RSB (pg 797)
Theta_max=45*pi/180; %rad % RSB max deflection angle=+-45 deg
l_RSB=1; %m %RSB moment arm length
T_max=(0.5/20)*13.3*Fs*sin(Theta_max);
%...End roll control parameters from pg 797

%...Parameters from pg 805 of the book
omega_max=0.05*pi/180; %rad
T_max=(0.5/20)*13.3*Fs*sin(Theta_max); %Nm
%...End parameters from pg 805

% Other values
mr=m*(M+m)/(M+2*m); %kg %Reduced mass
n=6.311e-5; %rad/s %Orbital rate for super-synchronus transfer orbit
 (SSTO)
                  %Value for this mission taken from pg 762
P=4.563e-6; %N/m^2 %Nominal solar-radiation-pressure constant at 1 AU
 from
                  %the sun (pg. 793)
```

Selcting the gains and initial conditions

```
%=====================================================================
% Final Gains
%=====================================================================
K_1=[0.000000110870580
   0.000032486945066
   0.000000000003033
   0.001003578091345
   0.480749420434239
   0.000000147802548
   0.000434177085802
   1.190067849066839
  -0.000000001205386]*10^6;
K_2=[0.000000600000000
   0.000345600000000
   0.000000000260417
   0.003000000000000
   4.320000000000000
   0.000000520833333
   0.003000000000000
   7.560000000000000
   0.000000297619048]*10^4;

% x0=[5 -5 -90 0 0 0]'*pi/180;
% xR=[0 0 -55 0 0 0]'*pi/180;
K_PSO=[0.000000956294894
   0.000259318792317
   0.000000000075565
   0.011655112126712
   3.839189494542947
   0.000000639451501
   0.004456965998984
   5.517868067201267
  -0.000000195023569]*10^5 % Better than above

gains=sort([K_1 K_2]')'
gain=gains(:,1);
KpTx=K_PSO(1);
KdTx=K_PSO(2);
KiTx=K_PSO(3);
Kpz=K_PSO(4);
Kdz=K_PSO(5);
Kiz=K_PSO(6);
Kpy=K_PSO(7);
Kdy=K_PSO(8);
Kiy=K_PSO(9);
%=========================================================================

% Initial state values
x0=[-3 4 -55 0 0 0]'*pi/180;

% Desired state values
xR=[-1 1 -90 0 0 0]'*pi/180;
```

```
% Simulation duration
T=3600*4; %sec
Tsim=0;

y=y_ss;
z=z_ss;
```

Calling the PID model and plotting the results

```
%-------------------------------------------------------------------------
% Parameter changes for robustness test
%-------------------------------------------------------------------------
% Only plot the ROBUSTNESS PLOTTING code
% Plot for i=1,2,3,4
rob=[0.8 1.6 2.4 3.2;
     0.4 0.8 1.2 1.6;
     0.3 0.6 0.9 1.2];
i=4;
Ix=rob(1,i)*4340; %kg-m^2
Iy=rob(2,i)*2171; %kg-m^2
Iz=rob(3,i)*2171; %kg-m^2
%-------------------------------------------------------------------------
sim('SPID_Control_yz')
sim('PSO_PID_Control_yz')
```

Plotting

```
disp('Plotting')
% Transposing time and output data
time=time';
x=x'*180/pi;
u=u';
time_s=time_s';
x_s=x_s'*180/pi;
u_s=u_s';
```

Closed Loop Plotting

```
figure
% Euler angles
subplot(3,2,1)
plot(time,x(1,:),'--',time_s,x_s(1,:))
ylabel('Roll, deg')
grid on
subplot(3,2,3)
plot(time,x(2,:),'--',time_s,x_s(2,:))
ylabel('Pitch, deg')
grid on
subplot(3,2,5)
plot(time,x(3,:),'--',time_s,x_s(3,:))
ylabel('Yaw, deg')
xlabel('Time, seconds')
legend('PSO-PID','FLS-PID')
grid on
```

```
% Euler angle derivatives
subplot(3,2,2)
plot(time,x(4,:),'--',time_s,x_s(4,:))
ylabel('Roll Rate, deg/sec')
grid on
subplot(3,2,4)
plot(time,x(5,:),'--',time_s,x_s(5,:))
ylabel('Pitch Rate, deg/sec')
grid on
subplot(3,2,6)
plot(time,x(6,:),'--',time_s,x_s(6,:))
ylabel('Yaw Rate, deg/sec')
xlabel('Time, seconds')
% legend('PSO-PID','FLS-PID')
grid on

figure
% Inputs
subplot(3,1,1)
plot(time,u(1,:),'--',time_s,u_s(1,:))
ylabel('Roll Torque, N-m')
grid on
subplot(3,1,2)
plot(time,u(2,:),'--',time_s,u_s(2,:))
ylabel('Pitch Trim Mass z, m')
grid on
subplot(3,1,3)
plot(time,u(3,:),'--',time_s,u_s(3,:))
ylabel('Yaw Trim Mass y, m')
xlabel('Time, seconds')
legend('PSO-PID','FLS-PID')
grid on
```

Variations in Inertia

```
figure(1)
% Euler angles
subplot(3,2,1)
plot(time_s,x_s(1,:)*180/pi,'LineWidth',2)
ylabel('Roll, deg')
grid on
hold on
subplot(3,2,3)
plot(time_s,x_s(2,:)*180/pi,'LineWidth',2)
ylabel('Pitch, deg')
grid on
hold on
subplot(3,2,5)
plot(time_s,x_s(3,:)*180/pi,'LineWidth',2)
ylabel('Yaw, deg')
xlabel('Time, seconds')
grid on
hold on
```

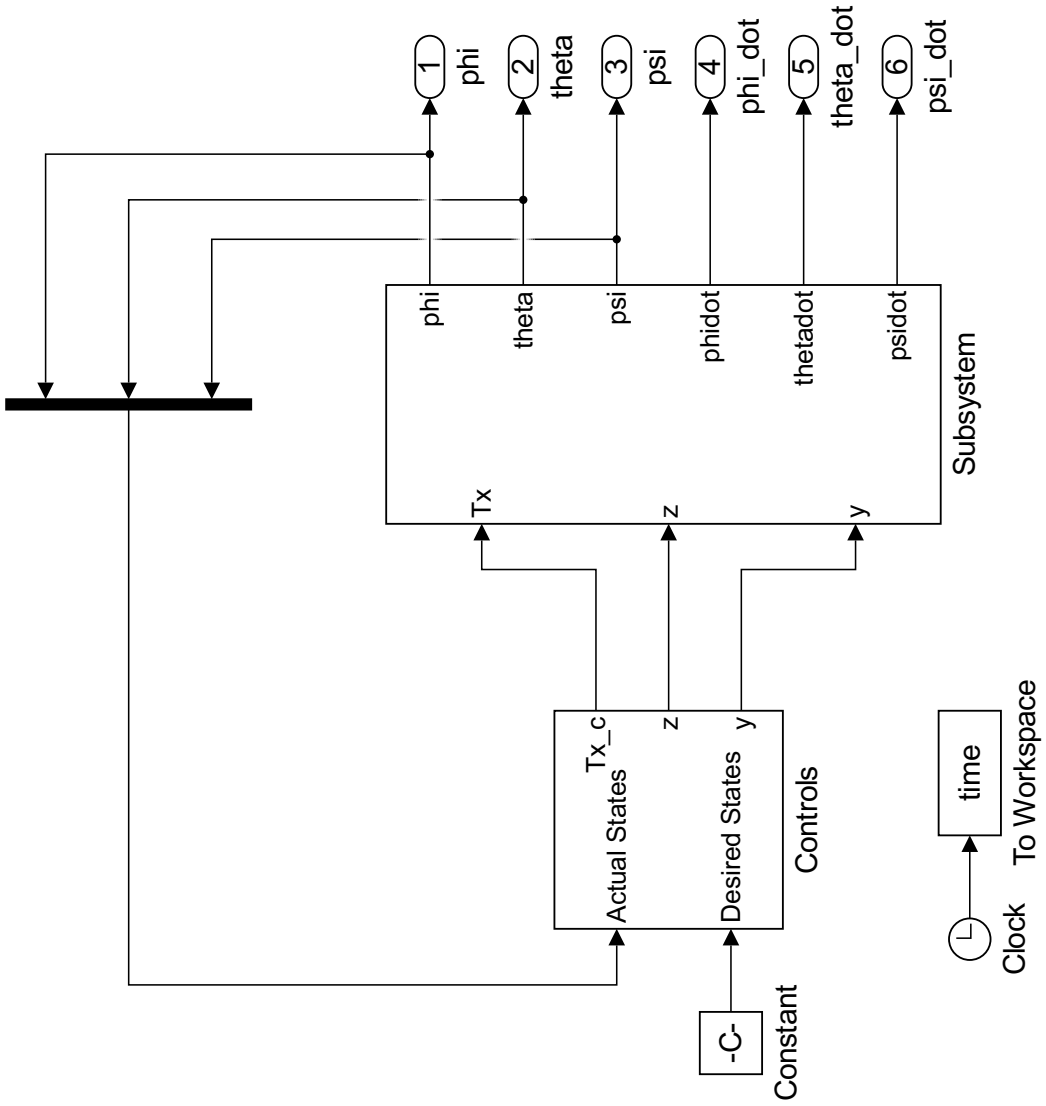
```
% Euler angle derivatives
subplot(3,2,2)
plot(time_s,x_s(4,:)*180/pi,'LineWidth',2)
ylabel('Roll Rate, deg/sec')
grid on
hold on
subplot(3,2,4)
plot(time_s,x_s(5,:)*180/pi,'LineWidth',2)
ylabel('Pitch Rate, deg/sec')
grid on
hold on
subplot(3,2,6)
plot(time_s,x_s(6,:)*180/pi,'LineWidth',2)
ylabel('Yaw Rate, deg/sec')
xlabel('Time, seconds')
grid on
hold on
clear time_s x_s

legend('Class I','Class II', 'Class III', 'Class IV')
```

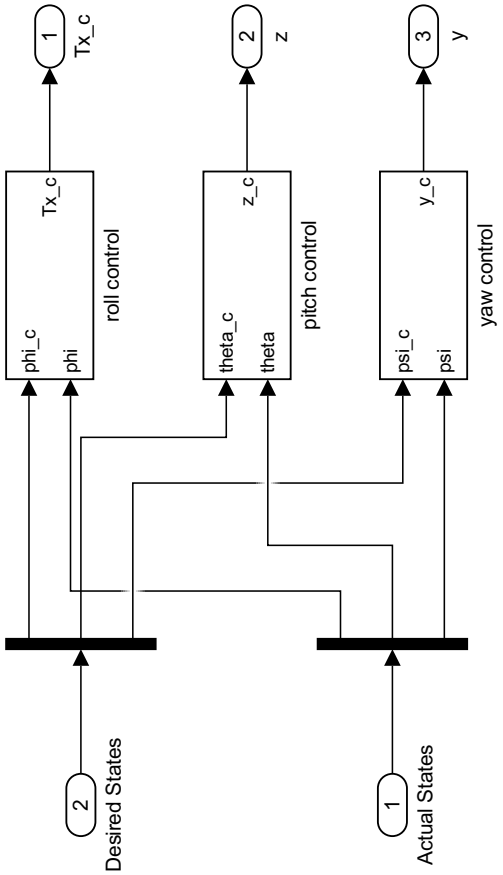
Gains Plot

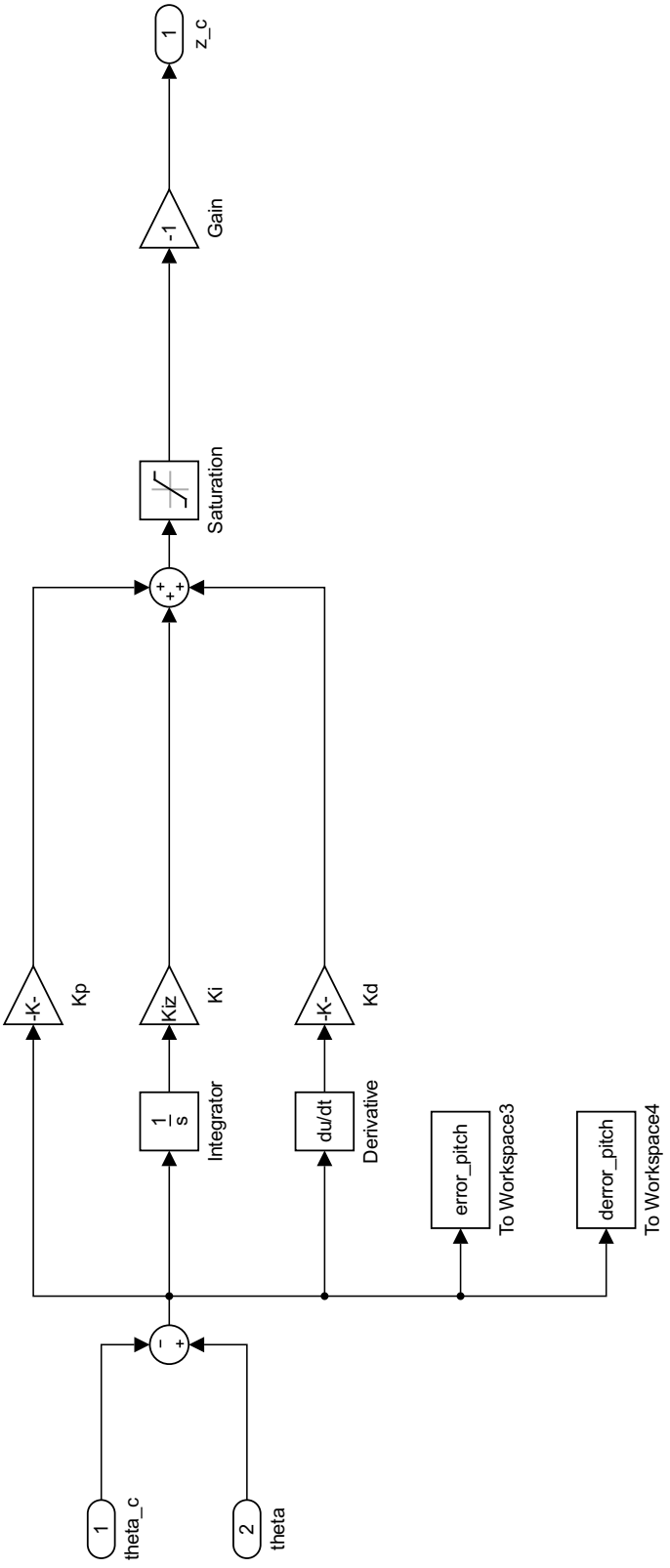
```
KTx=KTx';
Kz=Kz';
Ky=Ky';
figure
subplot(3,1,1)
plot(time_s,KTx(1,:),time_s,Kz(1,:),time_s,Ky(1,:))
ylabel('Kp')
subplot(3,1,2)
plot(time_s,KTx(2,:),time_s,Kz(2,:),time_s,Ky(2,:))
ylabel('Ki')
subplot(3,1,3)
plot(time_s,KTx(3,:),time_s,Kz(3,:),time_s,Ky(3,:))
ylabel('Kd')
legend('Tx','z','y')
```

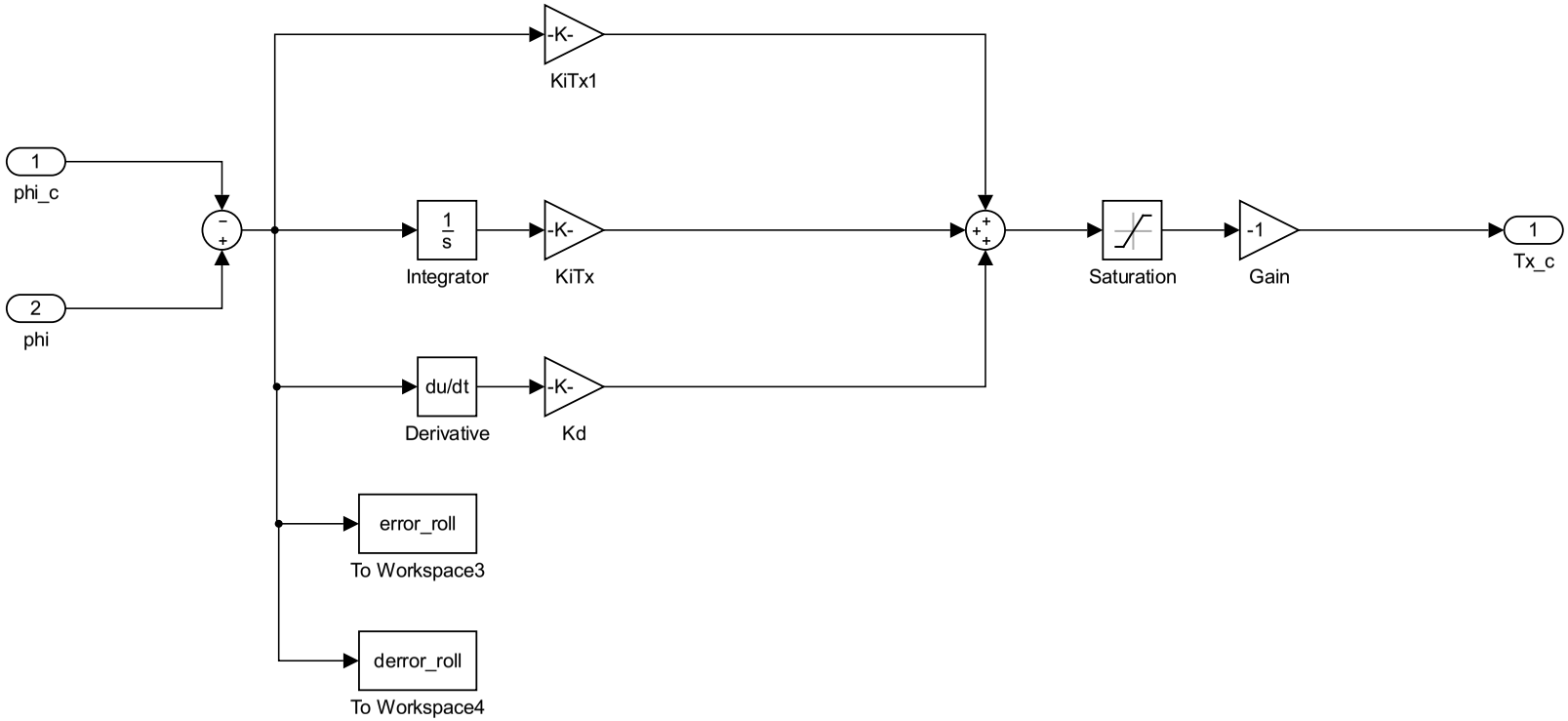
Published with MATLAB® R2015b

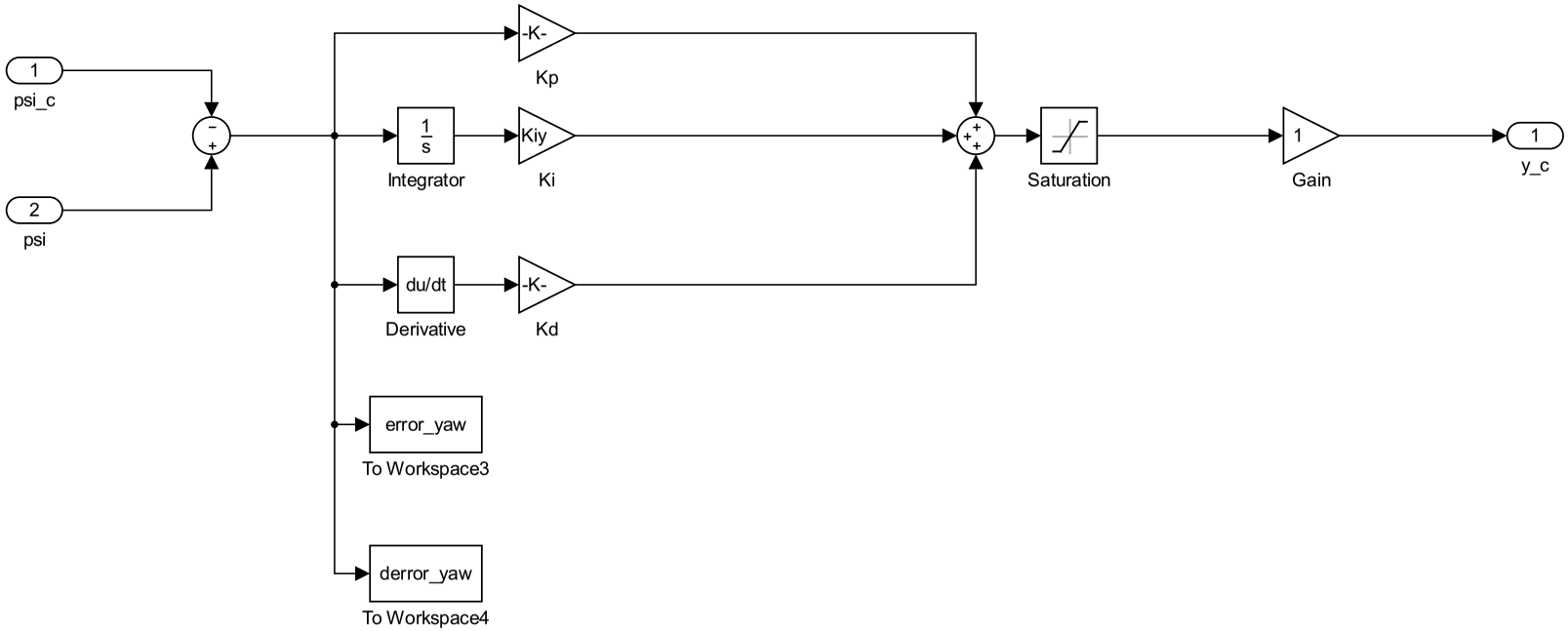
B.4 Closed Loop PID Controller

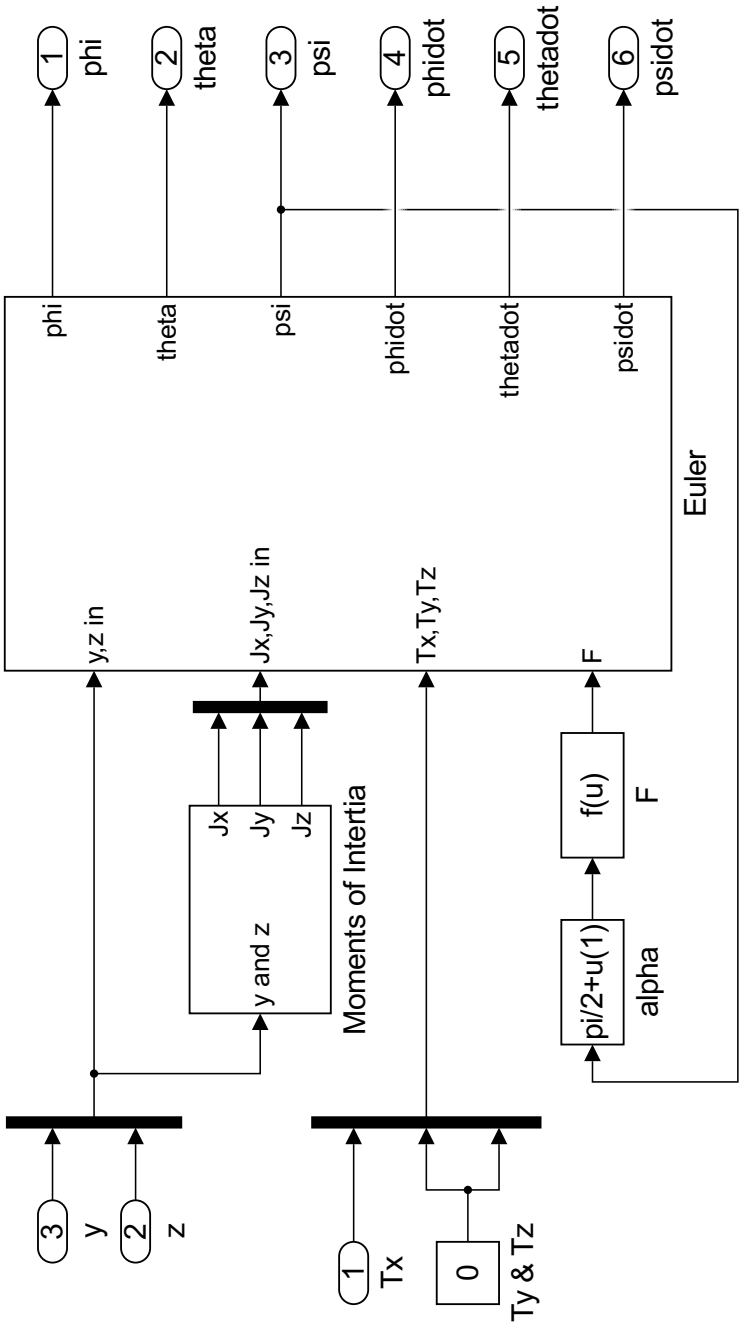


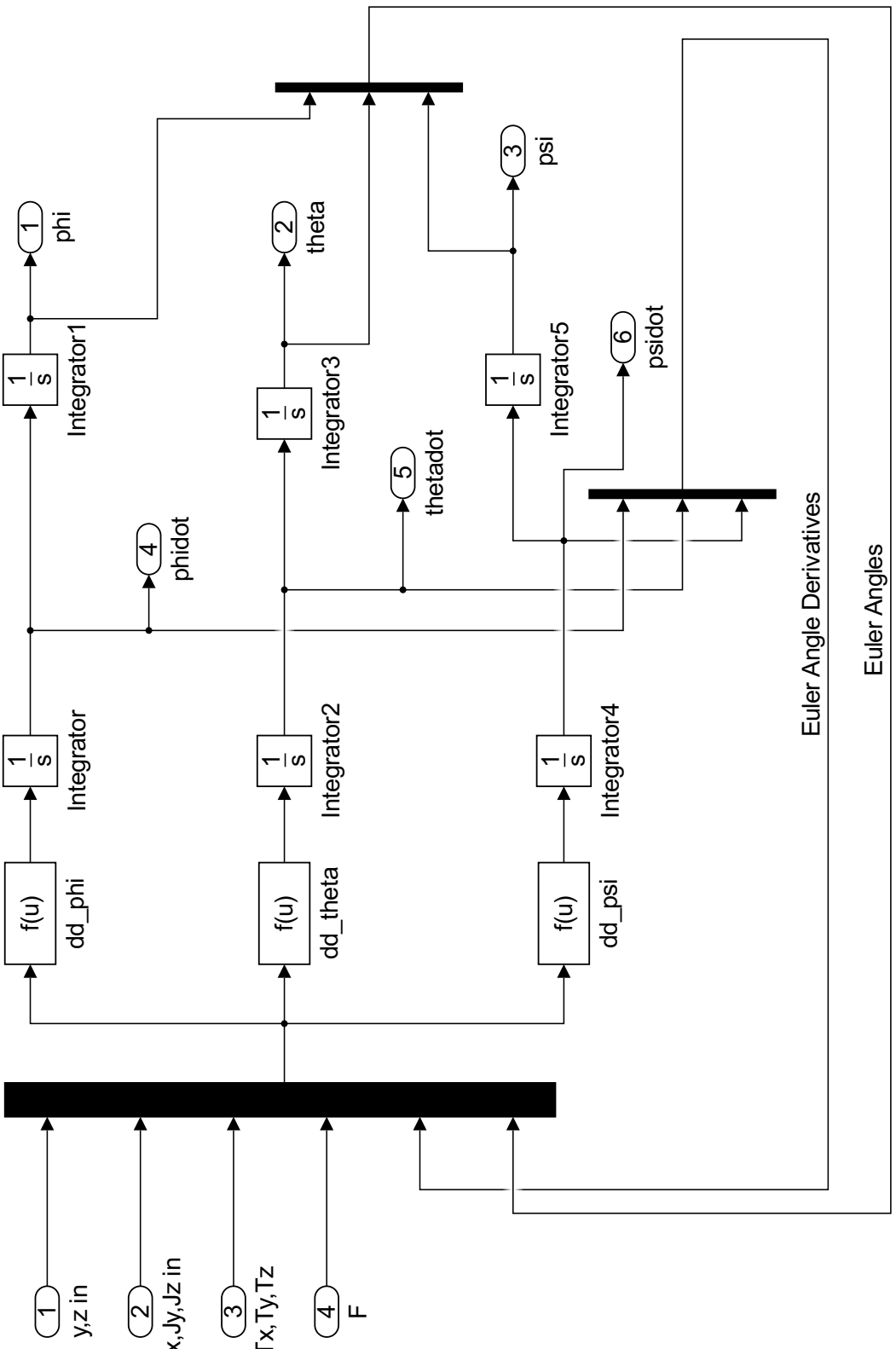

1
Tx_c
2
z
3
y
Tx_c
z_c
y_c
phi_c
phi
roll control
theta_c
theta
pitch control
psi_c
psi
yaw control
2
Desired States
1
Actual States

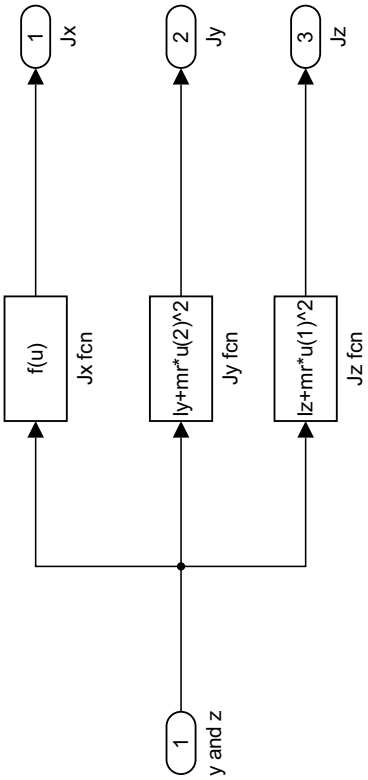

1
theta_c
2
theta
+
-
-K-
Kp
Kiz
Ki
-K-
Kd
1
s
Integrator
du/dt
Derivative
error_pitch
To Workspace3
derror_pitch
To Workspace4
+
+
+
Saturation
-1
Gain
1
z_c


1
phi_c
2
phi
-K-
KiTx1
1
s
Integrator
-K-
KiTx
du/dt
Derivative
-K-
Kd
error_roll
To Workspace3
derror_roll
To Workspace4
Saturation
-1
Gain
1
Tx_c

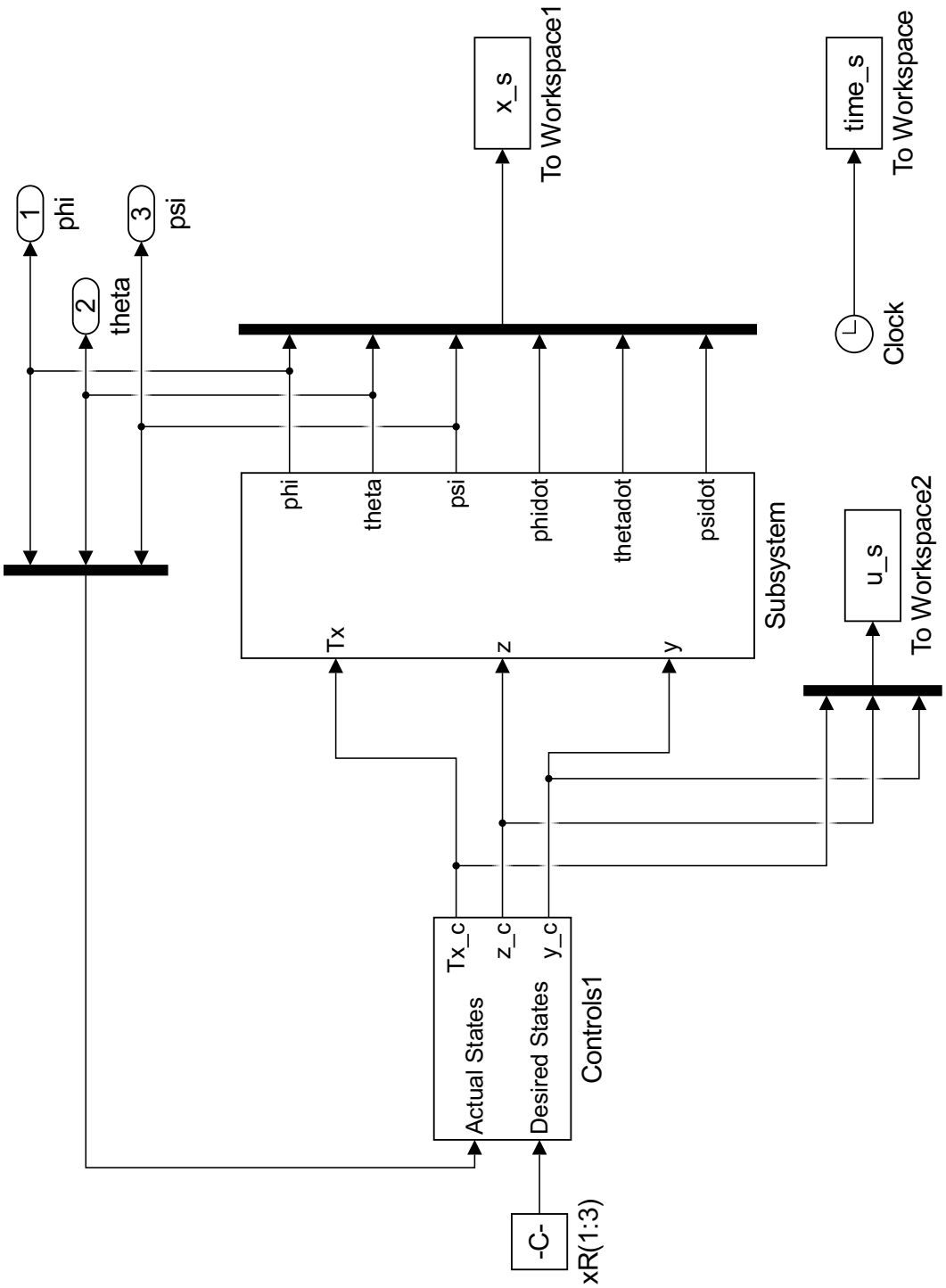

1
psi_c
2
psi
-K-
Kp
1
s
Integrator
Kiy
Ki
du/dt
Derivative
-K-
Kd
error_yaw
To Workspace3
derror_yaw
To Workspace4
Saturation
1
Gain
1
y_c

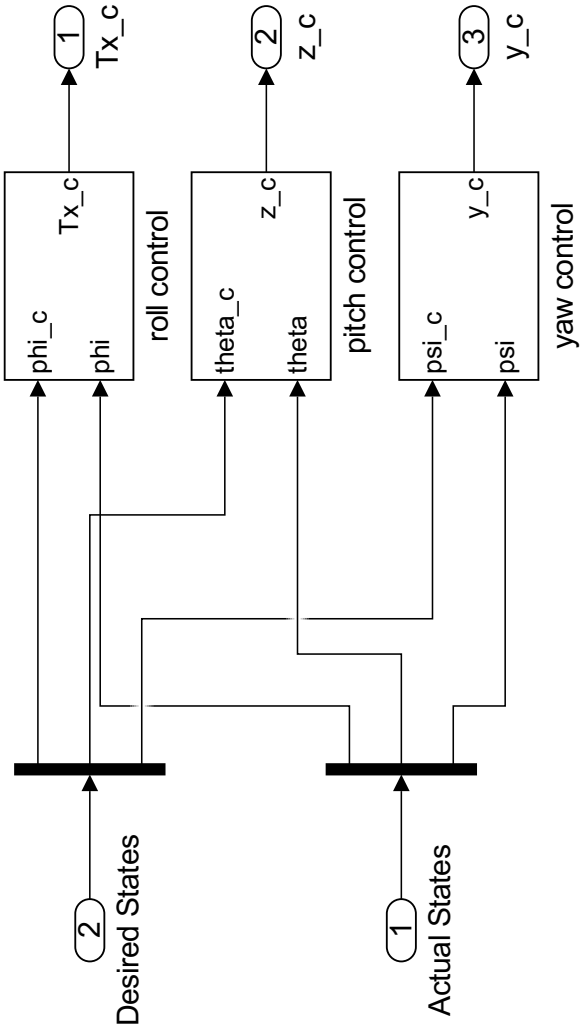

1
phi
2
theta
3
psi
4
phidot
5
thetadot
6
psidot
phi
theta
psi
phidot
thetadot
psidot
Euler
y,z in
Jx,Jy,Jz in
Tx,Ty,Tz
F
Jx
Jy
Jz
y and z
Moments of Intertia
f(u)
F
pi/2+u(1)
alpha
3
y
2
z
1
Tx
0
Ty & Tz

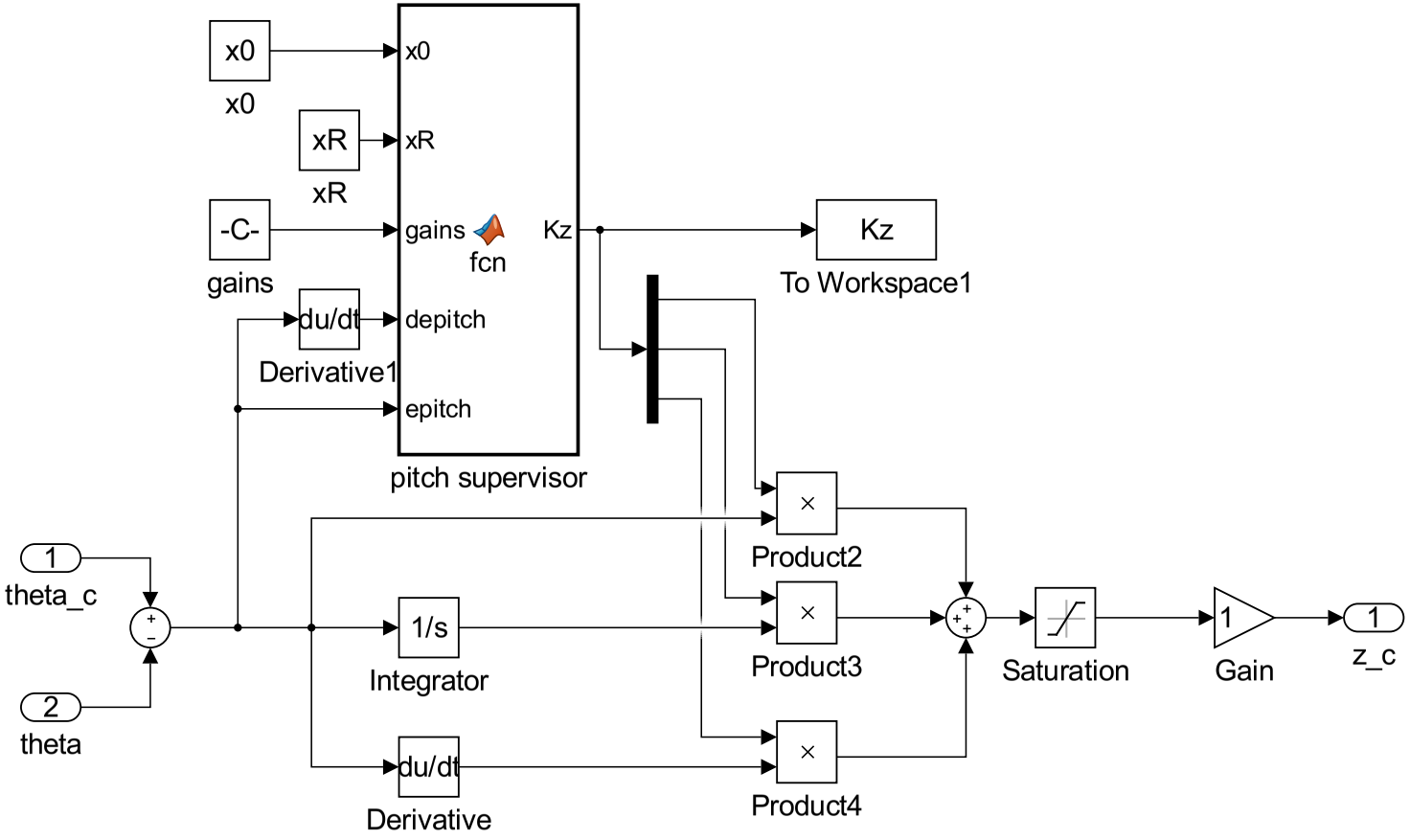

1
y,z in
2
Jx,Jy,Jz in
3
Tx,Ty,Tz
4
F
f(u)
dd_phi
f(u)
dd_theta
f(u)
dd_psi
1/s
Integrator
1/s
Integrator2
1/s
Integrator4
1/s
Integrator1
1/s
Integrator3
1/s
Integrator5
1
phi
2
theta
3
psi
4
phidot
5
thetadot
6
psidot
Euler Angle Derivatives
Euler Angles


1
y and z
f(u)
Jx fcn
Iy+mr*u(2)^2
Jy fcn
Iz+mr*u(1)^2
Jz fcn
1
Jx
2
Jy
3
Jz

B.5 Closed Loop FLS-PID Controller




1
Tx_c
2
z_c
3
y_c
Tx_c
phi_c
phi
roll control
z_c
theta_c
theta
pitch control
y_c
psi_c
psi
yaw control
2
Desired States
1
Actual States


x0
x0
xR
xR
-C-
gains
gains
fcn
Kz
Kz
To Workspace1
du/dt
Derivative1
depitch
epitch
pitch supervisor
1
theta_c
2
theta
1/s
Integrator
du/dt
Derivative
Product2
Product3
Product4
Saturation
1
Gain
1
z_c

This code calculates the pitch supervisory gains in the "pitch supervisor" block in the SPID_Control_yz.slx file

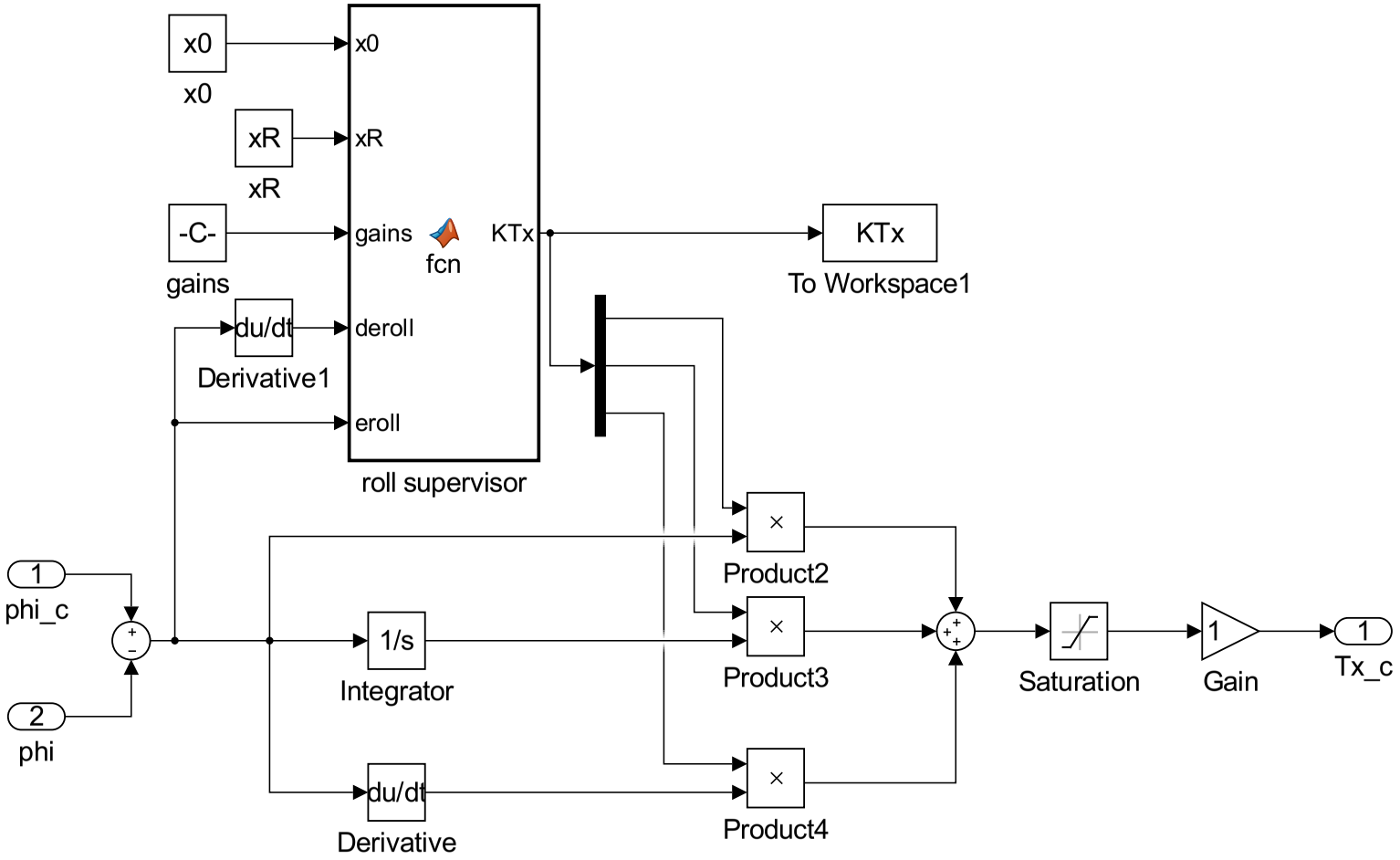
```
%  Written by: Joshua Baculi
%  Last Edited: August 15, 2016
```

pitch supervisor

```
function Kz    = fcn(x0,xR,gains,depitch,epitch)
%=========================================================================
% Membership functions
%=========================================================================
% Fuzzy pitch error
repitch=[-abs(xR(2)-x0(2)) 0 abs(xR(2)-x0(2))];
rdepitch=[-0.05 0 0.05]*pi/180;
mu1=zeros(size(repitch));
x11=repitch(1);
x12=repitch(2);
x13=repitch(3);

if epitch>=x11 & epitch<=x12
    mu1(1)=(epitch-x12)/(x11-x12);
    mu1(2)=(epitch-x11)/(x12-x11);
    mu1(3)=0;
elseif epitch>x12 & epitch<=x13
    mu1(1)=0;
    mu1(2)=(epitch-x13)/(x12-x13);
    mu1(3)=(epitch-x12)/(x13-x12);
elseif epitch>x13
    mu1(1)=0;
    mu1(2)=0;
    mu1(3)=1;
elseif epitch<x11
    mu1(1)=1;
    mu1(2)=0;
    mu1(3)=0;
end

% Fuzzy pitch error rate
mu2=zeros(size(rdepitch));
x21=rdepitch(1);
x22=rdepitch(2);
x23=rdepitch(3);

if depitch>=x21 & depitch<=x22
    mu2(1)=(depitch-x22)/(x21-x22);
    mu2(2)=(depitch-x21)/(x22-x21);
    mu2(3)=0;
elseif depitch>x22 & depitch<=x23
    mu2(1)=0;
    mu2(2)=(depitch-x23)/(x22-x23);
    mu2(3)=(depitch-x22)/(x23-x22);
elseif depitch>x23
```

```
    mu2(1)=0;
    mu2(2)=0;
    mu2(3)=1;
elseif depitch<x21
    mu2(1)=1;
    mu2(2)=0;
    mu2(3)=0;
end

%============================
% CHECK MEMBERSHIP FUNCTIONS %
%============================
% Combining the fuzzy membership functions
Mu={[mu1], [mu2]}; % input data: cell array of vectors
nvec=numel(Mu); %number of vectors
w=cell(1,nvec); %pre-define to generate comma-separated list
[w{2:-1:1}]=ndgrid(Mu{end:-1:1}); %the reverse order in these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
w = cat(nvec+1, w{:}); %concat the n n-dim arrays along dimension n+1
w = reshape(w,[],nvec); %reshape to obtain desired matrix
w=prod(w')';
h=w./sum(w);
%=========================================================================

%=========================================================================
% Gains
%=========================================================================
Kpr=gains(4,:);
Kdr=gains(5,:);
Kir=gains(6,:);

Kpi=[Kpr(2) Kpr(2) Kpr(2) Kpr(1) Kpr(2) Kpr(1) Kpr(2) Kpr(2) Kpr(2)]';
Kii=[Kir(1) Kir(1) Kir(1) Kir(2) Kir(2) Kir(2) Kir(2) Kir(2) Kir(1)]';
Kdi=[Kdr(1) Kdr(1) Kdr(1) Kdr(2) Kdr(2) Kdr(2) Kdr(1) Kdr(1) Kdr(1)]';
%=========================================================================

%=========================================================================
% Defuzzification
%=========================================================================
Kpz=sum(h.*Kpi);
Kiz=sum(h.*Kii);
Kdz=sum(h.*Kdi);
Kz=[Kpz Kiz Kdz]';

end
```

Published with MATLAB® R2015b


x0
x0
xR
xR
-C-
gains
du/dt
Derivative1
x0
xR
gains
fcn
KTx
deroll
eroll
roll supervisor
KTx
To Workspace1
1
phi_c
2
phi
1/s
Integrator
du/dt
Derivative
Product2
Product3
Product4
Saturation
1
Gain
1
Tx_c

This code calculates the roll supervisory gains the "roll supervisor" block in the SPID_Control_yz.slx file

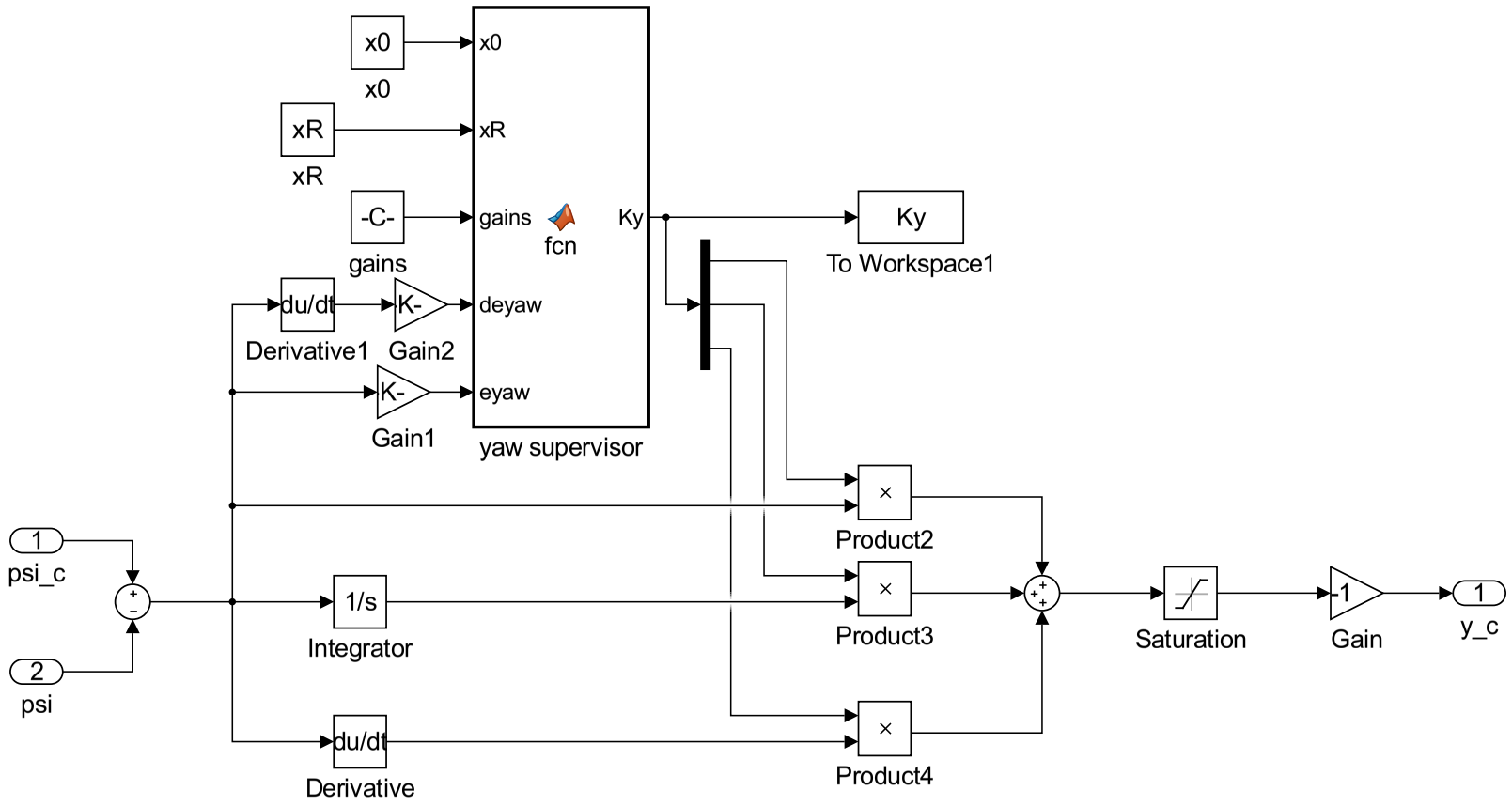
```
%  Written by: Joshua Baculi
%  Last Edited: August 15, 2016
```

roll supervisor

```
function KTx   = fcn(x0,xR,gains,deroll,eroll)
%=========================================================================
% Membership functions
%=========================================================================
% Fuzzy roll error
reroll=[-abs(xR(1)-x0(1)) 0 abs(xR(1)-x0(1))];
rderoll=[-0.05 0 0.05]*pi/180;
mu1=zeros(size(reroll));
x11=reroll(1);
x12=reroll(2);
x13=reroll(3);

if eroll>=x11 & eroll<=x12
    mu1(1)=(eroll-x12)/(x11-x12);
    mu1(2)=(eroll-x11)/(x12-x11);
    mu1(3)=0;
elseif eroll>x12 & eroll<=x13
    mu1(1)=0;
    mu1(2)=(eroll-x13)/(x12-x13);
    mu1(3)=(eroll-x12)/(x13-x12);
elseif eroll>x13
    mu1(1)=0;
    mu1(2)=0;
    mu1(3)=1;
elseif eroll<x11
    mu1(1)=1;
    mu1(2)=0;
    mu1(3)=0;
end

% Fuzzy roll error rate
mu2=zeros(size(rderoll));
x21=rderoll(1);
x22=rderoll(2);
x23=rderoll(3);

if deroll>=x21 & deroll<=x22
    mu2(1)=(deroll-x22)/(x21-x22);
    mu2(2)=(deroll-x21)/(x22-x21);
    mu2(3)=0;
elseif deroll>x22 & deroll<=x23
    mu2(1)=0;
    mu2(2)=(deroll-x23)/(x22-x23);
    mu2(3)=(deroll-x22)/(x23-x22);
elseif deroll>x23
    mu2(1)=0;
```

```
    mu2(2)=0;
    mu2(3)=1;
elseif deroll<x21
    mu2(1)=1;
    mu2(2)=0;
    mu2(3)=0;
end

%============================
% CHECK MEMBERSHIP FUNCTIONS %
%============================
% Combining the fuzzy membership functions
Mu={[mu1], [mu2]}; % input data: cell array of vectors
nvec=numel(Mu); %number of vectors
w=cell(1,nvec); %pre-define to generate comma-separated list
[w{2:-1:1}]=ndgrid(Mu{end:-1:1}); %the reverse order in these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
w = cat(nvec+1, w{:}); %concat the n n-dim arrays along dimension n+1
w = reshape(w,[],nvec); %reshape to obtain desired matrix
w=prod(w')';
h=w./sum(w);
%=========================================================================

%=========================================================================
% Gains
%=========================================================================
Kpr=gains(1,:);
Kdr=gains(2,:);
Kir=gains(3,:);

Kpi=[Kpr(2) Kpr(2) Kpr(2) Kpr(1) Kpr(2) Kpr(1) Kpr(2) Kpr(2) Kpr(2)]';
Kii=[Kir(1) Kir(1) Kir(1) Kir(2) Kir(2) Kir(2) Kir(2) Kir(2) Kir(1)]';
Kdi=[Kdr(1) Kdr(1) Kdr(1) Kdr(2) Kdr(2) Kdr(2) Kdr(1) Kdr(1) Kdr(1)]';
%=========================================================================

%=========================================================================
% Defuzzification
%=========================================================================
KpTx=sum(h.*Kpi);
KiTx=sum(h.*Kii);
KdTx=sum(h.*Kdi);
KTx=[KpTx KiTx KdTx]';

end
```

Published with MATLAB® R2015b


x0
x0
xR
xR
-C-
gains
x0
xR
gains
fcn
Ky
yaw supervisor
Ky
To Workspace1
du/dt
Derivative1
K-
Gain2
K-
Gain1
deyaw
eyaw
1
psi_c
2
psi
1/s
Integrator
du/dt
Derivative
Product2
Product3
Product4
Saturation
-1
Gain
1
y_c

This code calculates the yaw supervisory gains in the "yaw supervisor" block in the SPID_Control_yz.slx file

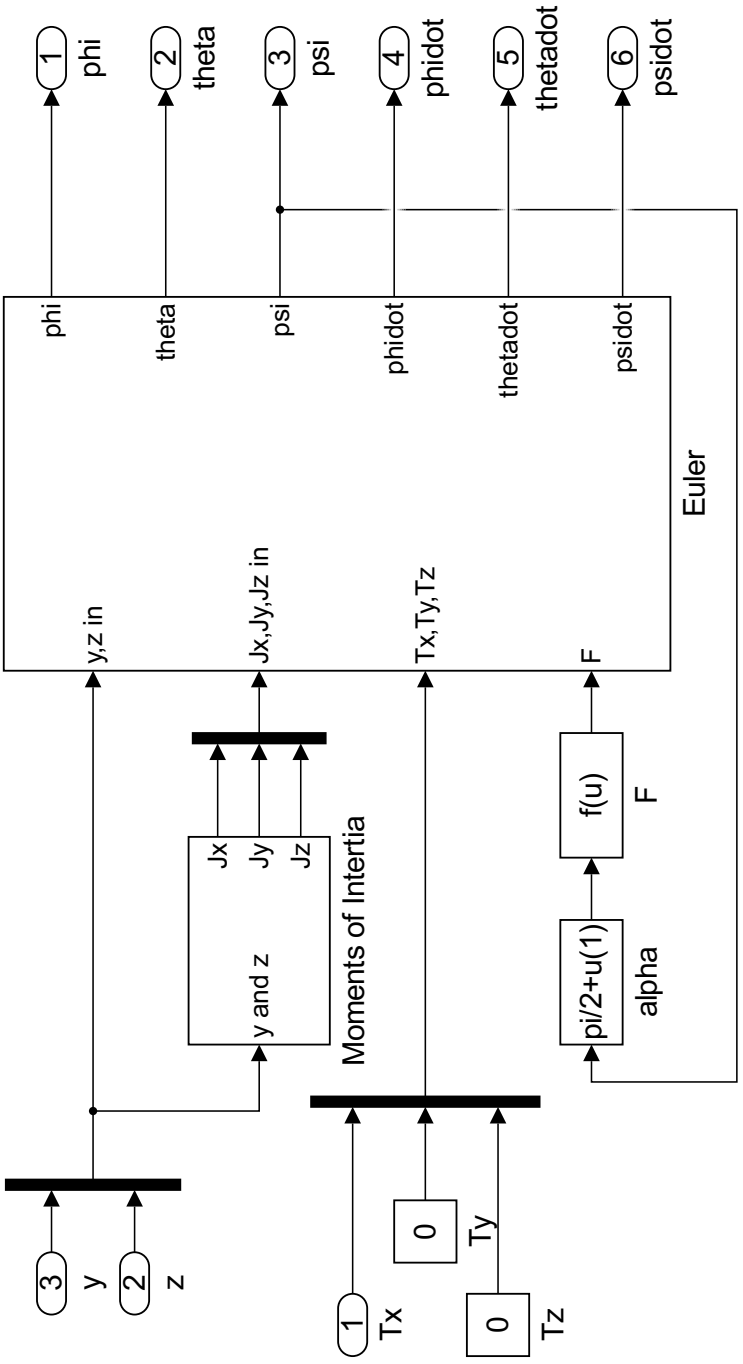
```
%  Written by: Joshua Baculi
%  Last Edited: August 15, 2016
```

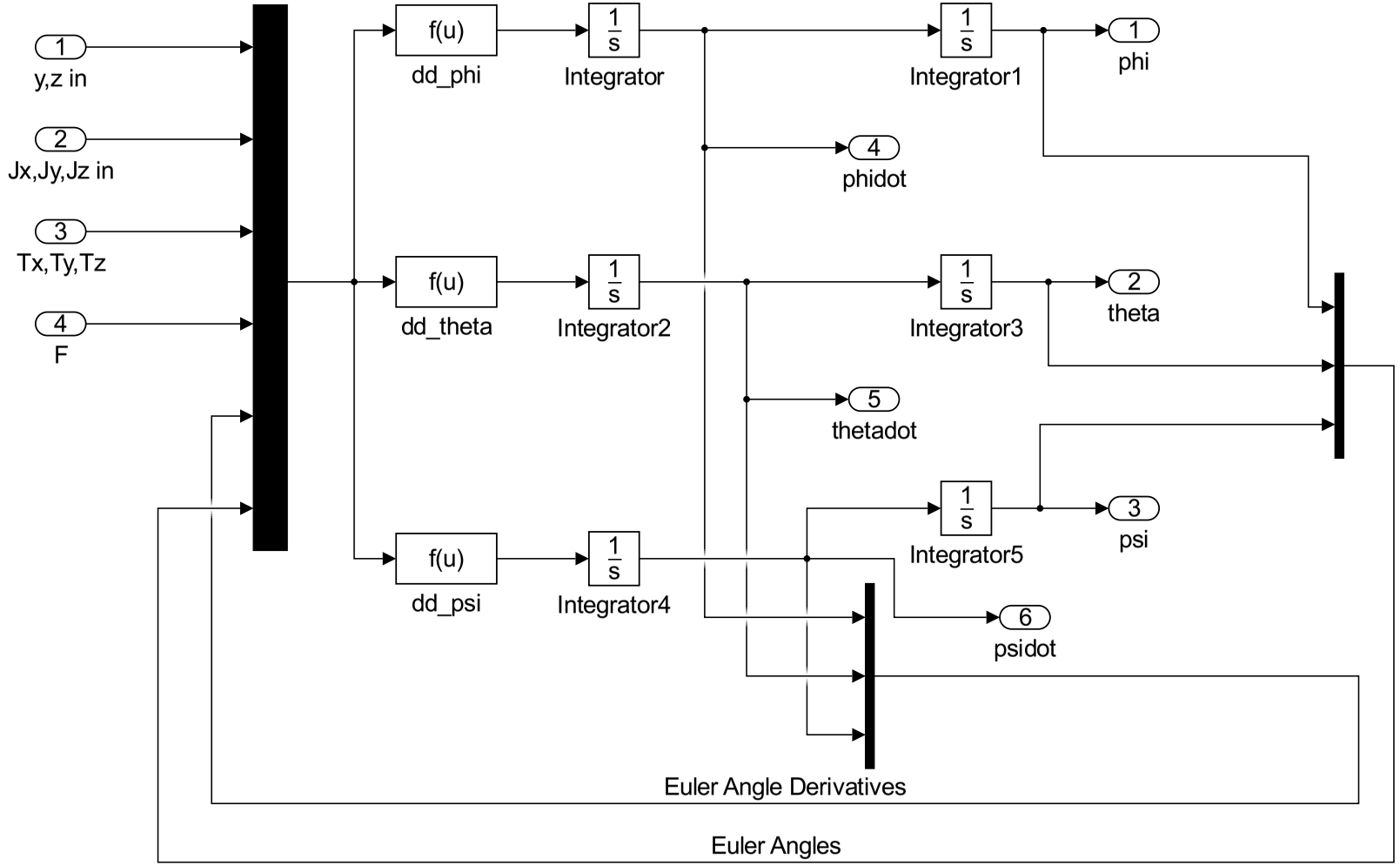
yaw supervisor

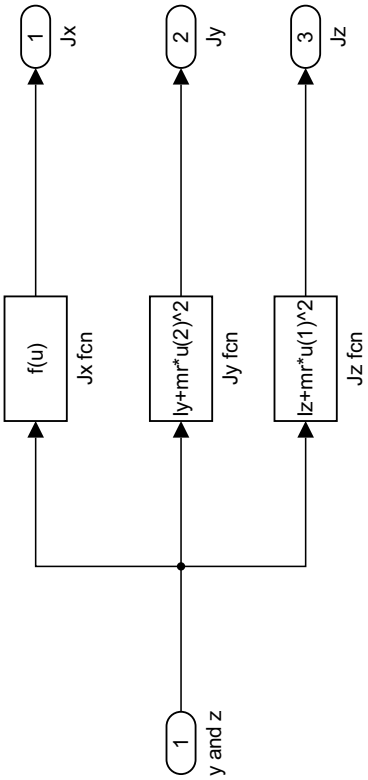
```
function Ky   = fcn(x0,xR,gains,deyaw,eyaw)
%=========================================================================
% Membership functions
%=========================================================================
% Fuzzy yaw error
reyaw=[-abs(xR(3)-x0(3)) 0 abs(xR(3)-x0(3))];
rdeyaw=[-0.05 0 0.05]*pi/180;
mu1=zeros(size(reyaw));
x11=reyaw(1);
x12=reyaw(2);
x13=reyaw(3);

if eyaw>=x11 & eyaw<=x12
    mu1(1)=(eyaw-x12)/(x11-x12);
    mu1(2)=(eyaw-x11)/(x12-x11);
    mu1(3)=0;
elseif eyaw>x12 & eyaw<=x13
    mu1(1)=0;
    mu1(2)=(eyaw-x13)/(x12-x13);
    mu1(3)=(eyaw-x12)/(x13-x12);
elseif eyaw>x13
    mu1(1)=0;
    mu1(2)=0;
    mu1(3)=1;
elseif eyaw<x11
    mu1(1)=1;
    mu1(2)=0;
    mu1(3)=0;
end

% Fuzzy yaw error rate
mu2=zeros(size(rdeyaw));
x21=rdeyaw(1);
x22=rdeyaw(2);
x23=rdeyaw(3);

if deyaw>=x21 & deyaw<=x22
    mu2(1)=(deyaw-x22)/(x21-x22);
    mu2(2)=(deyaw-x21)/(x22-x21);
    mu2(3)=0;
elseif deyaw>x22 & deyaw<=x23
    mu2(1)=0;
    mu2(2)=(deyaw-x23)/(x22-x23);
    mu2(3)=(deyaw-x22)/(x23-x22);
elseif deyaw>x23
```

```
    mu2(1)=0;
    mu2(2)=0;
    mu2(3)=1;
elseif deyaw<x21
    mu2(1)=1;
    mu2(2)=0;
    mu2(3)=0;
end

%============================
% CHECK MEMBERSHIP FUNCTIONS %
%============================
% Combining the fuzzy membership functions
Mu={[mu1], [mu2]}; % input data: cell array of vectors
nvec=numel(Mu); %number of vectors
w=cell(1,nvec); %pre-define to generate comma-separated list
[w{2:-1:1}]=ndgrid(Mu{end:-1:1}); %the reverse order in these two
%comma-separated lists is needed to produce the rows of the result
 matrix in
%lexicographical order
w = cat(nvec+1, w{:}); %concat the n n-dim arrays along dimension n+1
w = reshape(w,[],nvec); %reshape to obtain desired matrix
w=prod(w')';
h=w./sum(w);
%=========================================================================

%=========================================================================
% Gains
%=========================================================================
Kpr=gains(7,:);
Kdr=gains(8,:);
Kir=gains(9,:);

Kpi=[Kpr(2) Kpr(2) Kpr(2) Kpr(1) Kpr(2) Kpr(1) Kpr(2) Kpr(2) Kpr(2)]';
Kii=[Kir(1) Kir(1) Kir(1) Kir(2) Kir(2) Kir(2) Kir(2) Kir(2) Kir(1)]';
Kdi=[Kdr(1) Kdr(1) Kdr(1) Kdr(2) Kdr(2) Kdr(2) Kdr(1) Kdr(1) Kdr(1)]';
%=========================================================================

%=========================================================================
% Defuzzification
%=========================================================================
Kpy=sum(h.*Kpi);
Kiy=sum(h.*Kii);
Kdy=sum(h.*Kdi);
Ky=[Kpy Kiy Kdy]';

end
```

Published with MATLAB® R2015b


1
phi
2
theta
3
psi
4
phidot
5
thetadot
6
psidot
phi
theta
psi
phidot
thetadot
psidot
Euler
y,z in
Jx,Jy,Jz in
Tx,Ty,Tz
F
Jx
Jy
Jz
y and z
Moments of Intertia
f(u)
F
pi/2+u(1)
alpha
3
y
2
z
1
Tx
0
Ty
0
Tz


1
y,z in
2
Jx,Jy,Jz in
3
Tx,Ty,Tz
4
F
f(u)
dd_phi
1/s
Integrator
1/s
Integrator1
1
phi
4
phidot
f(u)
dd_theta
1/s
Integrator2
1/s
Integrator3
2
theta
5
thetadot
f(u)
dd_psi
1/s
Integrator4
1/s
Integrator5
3
psi
6
psidot
Euler Angle Derivatives
Euler Angles


1
y and z
f(u)
Jx fcn
Iy+mr*u(2)^2
Jy fcn
Iz+mr*u(1)^2
Jz fcn
1
Jx
2
Jy
3
Jz

Vita

In 2015, Joshua Baculi obtained his Bachelor's Degree from Santa Clara University in Mechanical Engineering with a minor in Mathematics. He continued his studies in mechanical engineering through the Master's program at Santa Clara. His graduate studies follow the Dynamics and Controls concentration with classes more oriented to aerospace studies. He is particularly interested in the attitude and orbit control of spacecrafts as well as astrodynamics.

In 2014, he was awarded the NASA Undergraduate Aeronautics Scholarship. The award consisted of a two year scholarship that he used to fund his graduate studies as well as a summer internship at NASA Ames Research Center.