

6-15-2017

Hybrid Fuzzy Logic and Extremum Seeking Attitude Control of Solar Sail Spacecraft

Nikolai Kalnin

Santa Clara University, nkalnin@scu.edu

Follow this and additional works at: http://scholarcommons.scu.edu/mech_mstr



Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Kalnin, Nikolai, "Hybrid Fuzzy Logic and Extremum Seeking Attitude Control of Solar Sail Spacecraft" (2017). *Mechanical Engineering Master's Theses*. 12.

http://scholarcommons.scu.edu/mech_mstr/12

This Thesis is brought to you for free and open access by the Engineering Master's Theses at Scholar Commons. It has been accepted for inclusion in Mechanical Engineering Master's Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.

Santa Clara University
DEPARTMENT OF MECHANICAL ENGINEERING

DATE: JUNE 12, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY


Nikolai Kalnin

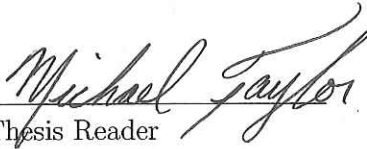
ENTITLED

**Hybrid Fuzzy Logic and Extremum Seeking Attitude Control of Solar Sail
Spacecraft**

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

MASTER OF SCIENCE IN MECHANICAL ENGINEERING


Thesis Advisor
Dr. Mohammad A. Ayoubi


Thesis Reader
Dr. Michael Taylor


Chairman of Department
Dr. Drazen Fabris

Hybrid Fuzzy Logic and Extremum Seeking Attitude Control of Solar Sail Spacecraft

By

Nikolai Kalnin

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, 2017

Santa Clara, California

Acknowledgments

I would like to thank Dr. Mohammad A. Ayoubi, for all of your help and direction that has made this research into reality. I truly appreciate all the guidance and I have had a great time working with you!

I would also like to offer a sincere thank you to the Thesis Reviewer, Professor Michael Taylor. I appreciate the time you have taken to look at this work and feedback you have provided.

Next, I would like to thank Joshua Baculi, whose work this thesis is built upon. He graciously provided his code as a starting point for the work in this Thesis.

Lastly, I would like to acknowledge my parents Nikolai and Valentina, for their unwavering support and all the help over the years. I cannot express enough gratitude for all of the things you have done for me in life, this is just the most recent.

Hybrid Fuzzy Logic and Extremum Seeking Attitude Control of Solar Sail Spacecraft

Nikolai Kalnin

Department of Mechanical Engineering
Santa Clara University
Santa Clara, California
2017

ABSTRACT

This thesis explores four controllers applied to the attitude control of a solar sail craft with control masses with the goal of showing benefits over more standard control schemes. The controllers examined in this paper are: 1) a PID controller that incorporates a discrete extremum seeking algorithm, 2) a type-1 fuzzy logic controller that incorporates a discrete extremum seeking algorithm, 3) a type-1 fuzzy logic controller, and 4) a type-2 fuzzy logic controller. The first two controllers are examined for their ability to quickly converge to a set of optimal gains over time. The latter two controllers are evaluated for their ability to maintain stability with respect to model uncertainty and sensor noise. The four controllers discussed in this paper are compared against other control techniques that have already been shown in previously published literature to provide good control performance when applied to this system.

Table of Contents

1	Introduction	3
2	Dynamics of a Solar Sail with Translating Masses	7
2.1	Model Description	7
2.2	Equations of Motion	11
3	Extremum Seeking	16
3.1	Introduction	16
3.2	Theory	17
3.3	Hybrid PID-ES Controller	18
3.4	Hybrid Fuzzy-ES Controller	20
3.5	PSO-PID Controller	20
3.6	Simulation Results	23
3.6.1	PID-ES vs Fuzzy-ES	23
3.6.2	ES Convergence Improvement	33

4 Fuzzy Controller (Type I and Type II)	42
4.1 Introduction and Theory	42
4.2 Simulation Results	48
5 Conclusion	56
6 Future Work	58
References	59
Appendix: Computer Code	63
A PID-ES Controller	63
B Fuzzy-ES Controller	78
C Type-1 vs Type-2 Fuzzy PID Driving Script	92
D Type-1 Fuzzy PID Controller	119
E Type-2 Fuzzy PID Controller	129
F PSO-PID Controller	139

List of Figures

2.1	Square solar sail spacecraft with translating masses.	8
2.2	Full-thrust and zero-thrust attitudes.	10
2.3	Solar-sail spacecraft reference frames in orbit.	13
3.1	Schematic of discrete extremum-seeking algorithm.	18
3.2	PID-ES controller schematic.	19
3.3	Fuzzy-ES controller schematic.	21
3.4	Simulation of PID-ES controller, time response.	25
3.5	Simulation of PID-ES controller, gain evolutions.	26
3.6	Simulation of PID-ES controller, cost function evolution.	27
3.7	Simulation of Fuzzy-ES controller, time response.	28
3.8	Simulation of Fuzzy-ES controller, gain evolutions.	29
3.9	Simulation of Fuzzy-ES controller, cost function evolution.	30
3.10	Simulation of PSO-PID controller, time response.	31
3.11	Simulation of PSO-PID controller, control action time response.	32
3.12	1-Dimensional ES example of convergence rate vs. gradient	35

3.13	Simulation of improved ES convergence algorithm (2000 iterations), time response.	36
3.14	Simulation of improved ES convergence algorithm (2000 iterations), gain evolutions.	37
3.15	Simulation of improved ES convergence algorithm (2000 iterations), cost function evolution.	38
3.16	Simulation of improved ES convergence algorithm (200 iterations), time response.	39
3.17	Simulation of improved ES convergence algorithm (200 iterations), gain evolutions.	40
3.18	Simulation of improved ES convergence algorithm (200 iterations), cost function evolution.	41
4.1	Block diagram of a general type-1 fuzzy logic controller.	43
4.2	Example of membership functions for a water heater.	44
4.3	Fuzzy logic decision making.	45
4.4	A comparison of type-1 and type-2 fuzzy logic membership functions. . .	46
4.5	Type-2 fuzzy logic controller overview.	46
4.6	Type-1 and Type-2 Input Membership Functions	49
4.7	Simulation overlay of type-1 vs type-2 fuzzy controller time response, baseline case.	51
4.8	Simulation overlay of type-1 vs type-2 fuzzy controller control action and error, baseline case.	52

4.9	Simulation overlay of type-1 vs type-2 fuzzy controller time response, model mismatch case.	53
4.10	Simulation overlay of type-1 vs type-2 fuzzy controller control action and error, model mismatch case.	54
4.11	Simulation overlay of type-1 vs type-2 fuzzy controller time response, sensor noise case.	55

List of Tables

2.1 Solar Sail Parameters. 9

4.1 Example type-1 fuzzy rules. 43

Nomenclature

Symbol	Description
cm	Center of mass
cp	Center of [solar] pressure
$\{\vec{i}, \vec{j}, \vec{k}\}$	Basis vectors, body reference frame
$\{\vec{x}, \vec{y}, \vec{z}\}$	Basis vectors, LVLH reference frame
$\{\phi, \theta, \psi\}$	Euler angles
t	Time
M	Mass of solar sail craft
m	Mass of trim masses
m_r	Reduced mass
y, z	Position of control masses
ω_i	Angular rate about i-th axis
T_i	Control torques about i-th axis
I_x	Moment of inertia about x-axis
I_y	Moment of inertia about y-axis
I_z	Moment of inertia about z-axis
ϵ	cm - cp offset

Symbol	Description
G_i	Gravity gradient torque about i-th axis
μ_E	Earth's gravitational parameter
k	Discrete ES algorithm iteration
$\Theta(k)$	Vector of parameters (controller gains)
$\hat{\Theta}(k)$	Vector of parameter estimates (controller gains)
$J(\Theta(k))$	Cost function
e	Error
$\zeta(k)$	ES algorithm scalar
γ	Vector of ES algorithm adaptation gains
α	Vector of ES algorithm perturbation frequencies
ω	Vector of ES algorithm perturbation frequencies
$\{\mathbf{N}, \mathbf{Z}, \mathbf{P}\}$	Negative, zero, positive input membership functions, respectively
$\mu(x)$	Degree to which membership function describes input (x)
N_{PSO}	Number of particles in particle swarm

1. Introduction

The idea of a spacecraft propelled by reflecting the light emitted by the Sun was first seriously proposed by Tsiolkovsky [1] and the mathematics behind the approach were then worked out in 1926 by Tsander [2]. Additional theoretical work on the subject trickled in until 1976, when the Jet Propulsion Laboratory began the first formal design effort for a prototype solar sail spacecraft. Numerous sail configurations were explored with most falling into two categories: static-sail and rotating-blade designs. In a static-sail design, the sail was strung between rigid booms, much like a sailboat. In contrast, the rotating-blade designs used centrifugal force caused by the rotation of the spacecraft to stretch out the sails. Advantages of the rotating-blade designs include simpler manufacturing of the sails and a simpler mechanism for initially unfurling the sail, but these advantages come at the cost of making the craft inherently more difficult to control because the system is rotating.

With the first successful launch of the solar sail craft IKAROS (Interplanetary Kitecraft Accelerated by Radiation of the Sun) by the Japan Aerospace Exploration Agency (JAXA) in 2010, many missions have been proposed that are uniquely suited to solar sail craft. The same year also saw the launch of Nanosail-D2, a proof-of-concept solar sail nanosatellite. The miniature solar sail craft does not have an active attitude control system, instead relying on geometry and magnets to maintain its orientation during its mission [3]. In 2016, Breakthrough Initiatives, an organization searching for extrater-

restrial intelligence, announced its new research and engineering project Breakthrough Starshot. The project aims to develop a fleet of tiny solar sail craft that will be propelled toward Alpha Centauri, the closest star system to our Solar System [4]. NASA's Near-Earth Asteroid Scout (NEA Scout) is scheduled for launch in 2018 with the mission of searching out near-Earth asteroids and identifying possible risks and rewards for future manned and unmanned missions [5]. There will be even more such announcements as mankind seeks to explore deeper into space.

The major advantage of solar sail craft is their ability to use solar radiation pressure for thrust. The craft's large mirror-like sail reflects photons coming from the sun, and momentum transfer to the craft propels it away from the source of the light. This scheme greatly reduces the amount of fuel necessary for long missions deep into space[6]. In addition to providing thrust for the craft, several schemes have been proposed for using solar radiation for attitude control. This can be done by either moving the center of pressure (cp) relative to a fixed center of mass (cm), or by moving the cm relative to a fixed cp . In the former scheme, changes in cp can be affected by changing the geometry, size, or incident angle of different portions of the solar sail in a fashion similar to how sailboats adjust fore and aft sails to change heading without the use of a rudder. The resulting force imbalance about the cm of the craft generates a torque which can be used for attitude control. In the latter scheme, a control boom or sliding masses can be utilized to adjust the cm of the spacecraft relative to a fixed-geometry solar sail. These methods control architectures and others are examined in [7, 8, 9].

Many types of linear controllers have been applied for the attitude control of solar sail craft, each with different strengths and weaknesses. Wie [6] applied a Proportional-Integral-Derivative (PID) controller for single-axis Euler angle control of a solar sail craft in orbit around the Earth. The plots given for this controller showed that, while this type of controller could yield reasonable results, it did not take into account any possible

vibrational modes aggressive control could excite in a large, lightweight structure like a rigid solar sail. Several others [10, 11] have applied PD controllers to linearized models of the solar sail craft. Eldad's [10] work describes a two-part primary ACS that incorporates a standard PD controller in conjunction with a bias elimination scheme that periodically recalculates trim angles for a solar sail with rotating tip-mounted control vanes. The algorithm uses a built-in table for a first guess at the required trim angles and then relies on a learning algorithm to periodically adjust the actual trim angles to the correct values to compensate for any model mismatch or disturbance torques. The advantage is that this scheme lets the PD controller operate about an equilibrium position, thus eliminating the need for an integral term. The controller achieves quite reasonable performance, but only corrects the trim angle once every 15 hours, thus resulting in steady-state error from the primary Attitude Control System (ACS) for at least some time that the PD controller cannot fully eliminate. In Choi's [11] work, the PD controller moved adjustable vanes located at the tips of the rigid solar sail booms instead of moving masses along the booms. His work also went to considerable lengths to attempt to model membrane dynamics such as wrinkling and shading. These factors affect the thrust magnitude and direction generated by each segment of the solar sail, and thus can contribute significantly to disturbance torques affecting the craft's attitude during a mission. More recently, Takagi-Sugeno fuzzy control has been shown to be an attractive option for remedying the deficiencies of standard linear controllers [12, 13, 14, 15].

We investigate the application of the Extremum Seeking adaptive control algorithm applied to the attitude control system of a solar sail spacecraft. The motivation is to reduce the amount of effort that is currently required to build a dynamical model of the craft that is accurate enough to perform offline controller tuning. The ideal outcome would involve finding an algorithm that can start from a poorly tuned, but stable,

controller and quickly perform online gain tuning to improve performance to a level similar to that produced by current offline tuning algorithms such as Particle Swarm Optimization.

We also investigate the possible advantages of the Type-2 fuzzy controller over Type-1 fuzzy control. The goal of this comparison is to better understand practical advantages and disadvantages between the two nonlinear controllers and to validate claims about possible benefits ([16]) of Type-2 over Type-1 fuzzy controllers, especially with regards to sensor noise rejection.

We first present our work on both the PID-ES and Fuzzy-ES controllers, comparing them to a particle-swarm-optimized-PID (PSO-PID) controller similar to the one in [12]. Next we present a comparison of type-1 and type-2 fuzzy controllers looking at robustness against model inaccuracy, sensor noise, and ability to minimize error under standard operation. The Appendix contains the computer code used to generate the results.

2. Dynamics of a Solar Sail with Translating Masses

2.1 Model Description

Of the many shapes, sizes, and control schemes proposed over the years for solar sail craft, we consider a simple square solar sail with control masses moving along the perpendicular booms as shown in Figure 2.1. Due to the small magnitude of forces generated by the solar sail, the large size of the craft itself, and the stringent weight requirements for spacecraft, roll rates are assumed to be limited to a few degrees/hour. While the mathematical model itself may be accommodate much more aggressive angular rates, in reality the geometry and weight restrictions on these types of craft all but guarantee a multitude of vibrational modes in the structure itself. These vibrational modes are both difficult to model and control, but can easily be avoided by reducing the rotational speeds and accelerations of the craft.

The assumption about slow angular rates also enables us to further assume that because the dynamics of this craft are so much slower than the actuator dynamics of the control masses that any contributions will be negligible. As such, we will not consider them.

We continue the work done by Baculi and Ayoubi[12] used a combination of a PID

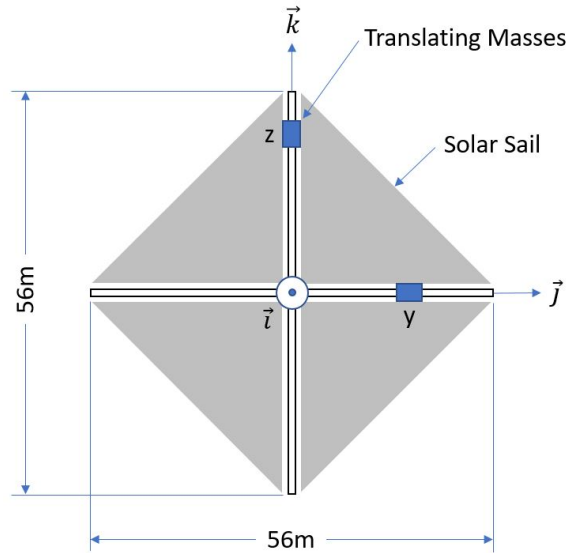


Fig. 2.1: Square solar sail spacecraft with translating masses.

controller with a Fuzzy-Logic Supervisor (FLS) for attitude control. We build on their results by presenting controllers that tackle a more complex set of equations of motion for the solar sail spacecraft through several different control techniques.

The solar sail configuration investigated here incorporates the translating or sliding masses configuration shown in Figure 2.1. As the masses slide within the booms of the sail, they pull the center of mass (cm) of the craft away from the center of pressure (cp) of the sail. This, in turn creates a torque which can be used to control the craft's pitch and yaw in the body frame. However, since the body frame roll axis is perpendicular to the sail, this type of craft must rely on alternate methods of attitude control along that axis. We assume that the roll axis attitude is controlled via a tip-mounted microthruster system as described in section 3.9.3 of [6]. The reference frames used herein, body and Local Vertical Local Horizontal (LVLH), are depicted in Figure 2.3.

Solar-sail craft employ secondary attitude control systems to deal with cases where the sail is perpendicular to the sun (and the control masses have no effect) or when the craft may be passing through the shadow of a celestial body [6]. For our purposes, we only

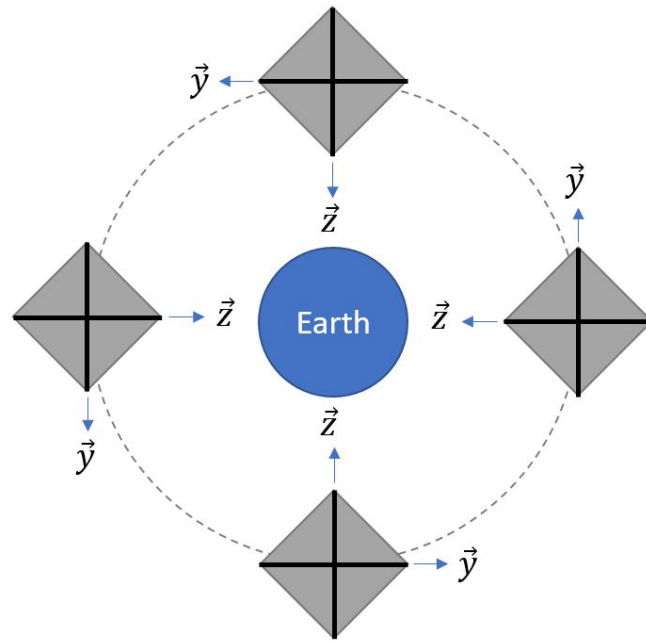
Variable	Value	Unit	Description
M	148	kg	Mass of solar sail craft
m	1	kg	Mass of trim masses
y, z	-	m	Position of control masses
ω_i	-	<i>deg/sec</i>	Angular rate about i-th axis
T_i	-	N-m	Control torques about i-th axis
I_x	4340	kg-m ²	Moment of inertia about x-axis
I_y	2171	kg-m ²	Moment of inertia about y-axis
I_z	2171	kg-m ²	Moment of inertia about z-axis
ϵ	0.1	m	<i>cm-cp</i> offset
G_i	-	N-m	Gravity gradient torque about i-th axis
μ_E	398600	km ³ /s ²	Earth's gravitational parameter

Table 2.1: Solar Sail Parameters.

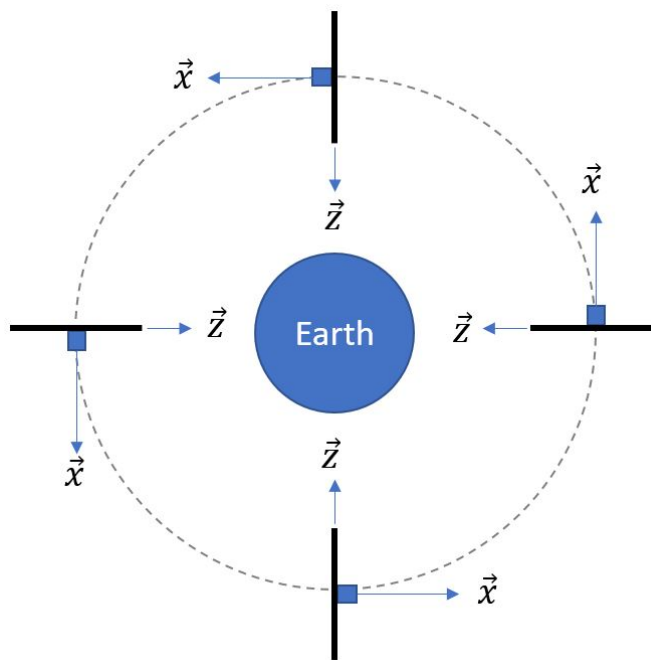
consider tip-mounted microthrusters system for the roll axis and the sliding masses for the pitch and yaw axes. All other control inputs are set to zero.

The body reference frame is shown in Figure 2.1. Pitch and yaw axes are aligned with the booms of the sail, while the roll axis points out of the page.

We consider a solar sail craft performing maneuvers in a dawn-dusk sun-synchronous (DDSS) orbit. The craft orbits Earth such that the craft z-axis is always pointed toward the Earth and the craft moves in a perpendicular fashion to the planet's rotation about the Sun. This enables the craft to achieve both zero-thrust and full-thrust orientations with minimal effect from gravity gradient torque as shown in Figures 2.2a and 2.2b, respectively.



(a)



(b)

Fig. 2.2: (a) Full-Thrust Attitude ($\psi = 0^\circ$), craft is facing the Sun. (b) Solar sail full-thrust attitude ($\psi = -90^\circ$), craft is edge-on with respect to the Sun.

2.2 Equations of Motion

Starting with the attitude equations of motion in the body reference frame derived by Wie in [6]

$$\begin{aligned}
 J_x \dot{\omega}_x &= (J_y - J_z) \omega_y \omega_z - \frac{m}{M + 2m} (y F_z - z F_y) + G_x, \\
 J_y \dot{\omega}_y &= (J_z - J_x) \omega_z \omega_x - \frac{m}{M + 2m} z F_x + G_y, \\
 J_z \dot{\omega}_z &= (J_x - J_y) \omega_x \omega_y - \frac{m}{M + 2m} y F_x + G_z,
 \end{aligned} \tag{2.1}$$

where ω_x , ω_y , ω_z are the angular velocities, F_x , F_y , F_z are the solar pressure forces acting on the craft, G_x , G_y , G_z are the gravity gradient torques.

$$\begin{aligned}
 J_x &\triangleq I_x + m_r (y^2 + z^2), \\
 J_y &\triangleq I_y + m_r z^2, \\
 J_z &\triangleq I_z + m_r y^2,
 \end{aligned} \tag{2.2}$$

are the total moments of inertia which include the effect of the sliding control masses, and m_r , the reduced mass, is defined as

$$m_r \triangleq \frac{m(M + m)}{M + 2m}. \tag{2.3}$$

Gravity gradient torques are

$$\begin{aligned}
 G_x &= -3n^2 (J_y - J_z) R_y R_z, \\
 G_y &= -3n^2 (J_z - J_x) R_z R_x, \\
 G_z &= -3n^2 (J_x - J_y) R_x R_y,
 \end{aligned} \tag{2.4}$$

with

$$\begin{aligned}
R_x &= -\sin \theta, \\
R_y &= \sin \phi \cos \theta, \\
R_z &= \cos \phi \cos \theta.
\end{aligned} \tag{2.5}$$

The solar radiation pressure, \vec{F} , is given by

$$\vec{F} = -F\hat{i} = -F_s \sin^2 \psi \hat{i}. \tag{2.6}$$

Next we consider the equations of motion in an Earth-centered circular orbit in the local vertical local horizontal (LVLH) reference frame. This reference frame has its origin at the spacecraft cm and has a set of orthonormal basis vectors $\{\hat{x}, \hat{y}, \hat{z}\}$. \hat{z} points toward the Earth, \hat{x} toward the locally horizontal direction, and \hat{y} is perpendicular to the orbit plane. By choosing the Euler angle sequence $C_1(\phi) \leftarrow C_2(\theta) \leftarrow C_3(\psi)$, the kinematic differential equation yields

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + n \begin{bmatrix} c_\theta c_\psi \\ s_\phi s_\theta s_\psi + c_\phi c_\psi \\ c_\phi s_\theta s_\psi - s_\phi c_\psi \end{bmatrix} \tag{2.7}$$

The $n = \sqrt{\mu_E/a^3}$ is the orbital rate and ϕ, θ, ψ are the roll, pitch, and yaw angles in the LVLH reference frame, respectively. From these equations, with no additional assumptions, we are able to write the first nonlinear equations of motion in a state-space formulation

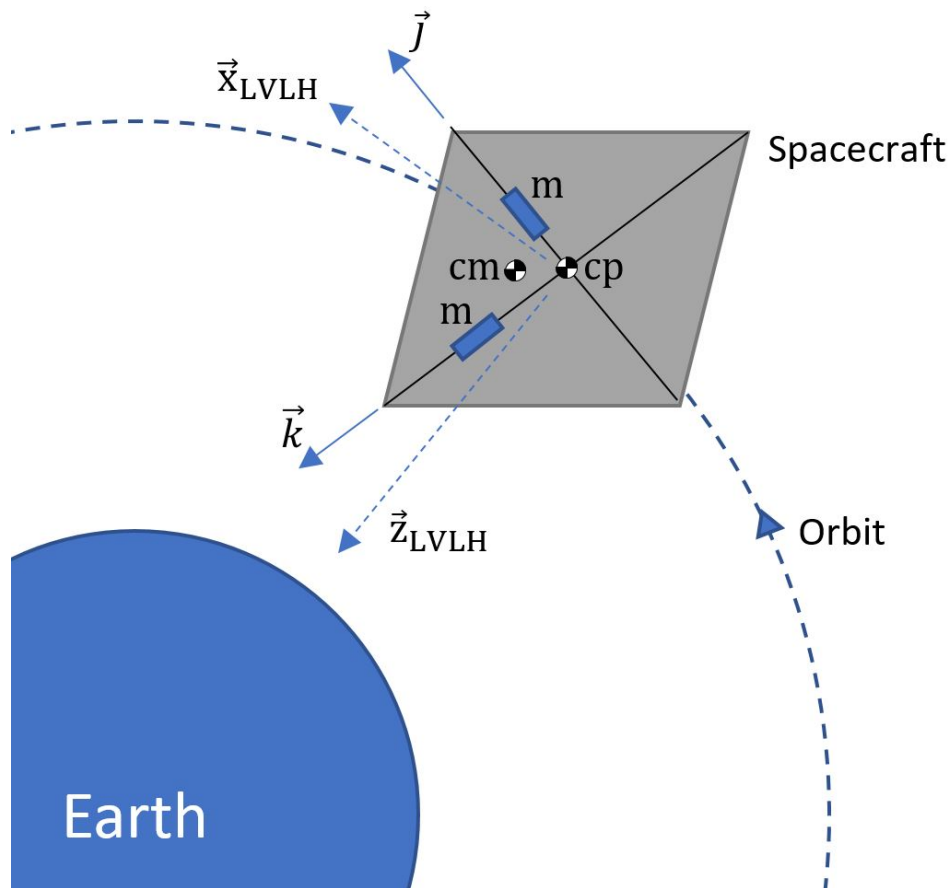


Fig. 2.3: Solar-sail spacecraft reference frames in orbit.

$$\begin{aligned}
\dot{\phi} &= (c_\theta \omega_x + s_\phi s_\theta \omega_y + c_\phi s_\theta \omega_z + n s_\psi) / c_\theta, \\
\dot{\theta} &= (c_\phi c_\theta \omega_y - s_\phi s_\theta \omega_z + n c_\theta c_\psi) / c_\theta, \\
\dot{\psi} &= (s_\phi \omega_y + c_\phi \omega_z + n s_\theta s_\psi) / c_\theta, \\
\dot{\omega}_x &= [(J_y - J_z) \omega_y \omega_z - \frac{m}{M + 2m} (y F_z - z F_y) + G_x] / J_x, \\
\dot{\omega}_y &= [(J_z - J_x) \omega_z \omega_x - \frac{m}{M + 2m} z F_x + G_y] / J_y, \\
\dot{\omega}_z &= [(J_x - J_y) \omega_x \omega_y - \frac{m}{M + 2m} y F_x + G_z] / J_z.
\end{aligned} \tag{2.8}$$

Next, if we assume small roll and pitch angles, we can approximate equations 2.7 as

$$\begin{aligned}
\omega_x &\approx \dot{\phi} - n \sin \psi, \\
\omega_y &\approx \dot{\theta} - n \cos \psi, \\
\omega_z &\approx \dot{\psi} - n(\theta \sin \psi - \phi \cos \psi).
\end{aligned} \tag{2.9}$$

If we further assume that $M \gg m$, we can substitute $m_r \approx m$. Using this result with Equations 2.1, 2.4, and 2.7, we get the linearized equations of motion

$$\begin{aligned}
\ddot{\phi} &= [-n^2(J_y - J_z)(3 + \cos^2 \psi)\phi + n^2(J_y - J_z)(\cos \psi \sin \psi)\theta \\
&\quad + n(J_x - J_y + J_z)(\cos \psi)\dot{\psi} + 0.5\epsilon F + T_x] / J_x, \\
\ddot{\theta} &= [-n^2(J_x - J_z)(3 + \sin^2 \psi)\theta + n^2(J_x - J_z)(\cos \psi \sin \psi)\phi \\
&\quad + n(J_x - J_y - J_z)(\sin \psi)\dot{\psi} + \epsilon F + T_y + \frac{m}{M + 2m} z F] / J_y, \\
\ddot{\psi} &= [-n^2(J_y - J_x) \sin \psi \cos \psi - n(J_x - J_y + J_z)(\cos \psi)\dot{\phi} \\
&\quad - n(J_x - J_y - J_z)(\sin \psi)\dot{\theta} + \epsilon F + T_z - \frac{m}{M + 2m} y F] / J_z.
\end{aligned} \tag{2.10}$$

Equations 2.1 can be easily manipulated into state-space formulation and will be herein

referred to as the linearized equations of motion. Similarly, equations 2.10 will be referred to as the nonlinear equations of motion.

3. Extremum Seeking

3.1 Introduction

In addition to applying a known hybrid ES-PID control system to the solar sail craft, we explore the novel application of ES to Fuzzy control optimization. The combination of these techniques should enable the creation of both model-based and model-free controllers that can accommodate highly nonlinear systems with minimal understanding of the internal dynamics of the plant. The additional advantage of online tuning can be used to maximize performance of some time-varying systems, providing the time-scale of the plant drift is much longer than the time-scale of the ES algorithm[17].

Several applications of the extremum seeking algorithm are described in [18]. They include the maximization of friction in a vehicle antilock breaking system (ABS), the maximization of production of micro-organisms in a continuously stirred bioreactor, minimization of fuel use in aircraft by flying in formations similar to that of migratory birds, minimization of emissions in gas turbines by reduction of pressure oscillations in the system, and instability avoidance by performance maximization in axial compressors.

3.2 Theory

The ES algorithm implemented here can be described in general terms as a mechanism to adjust controller parameters along the cost function gradient toward a local extremum in the cost function. In more detail, the cost function maps the control parameters in $\Theta(k)$ to the desired closed-loop system performance. It is important to note that ES can only achieve a local extremum in the cost function, and there is no guarantee of globally optimal performance.

A time-domain response experiment is run iteratively (shown in the dashed box of Figure 3.1) to generate the input for the cost function at different Θ . This algorithm achieves the local extremum by perturbing the input parameters $\Theta(k)$ and estimating the gradient $J(\Theta(k))$. The algorithm then filters and updates the parameter estimate $\hat{\Theta}(k)$, moving closer to the cost function extremum with each iteration. Equations 3.1, 3.2, and 3.3 show the discrete-time ES algorithm as we implemented them. Initial conditions were set to zero-gradient.

$$\zeta(k) = -h\zeta(k-1) + J(\Theta(k-1)), \quad (3.1)$$

$$\hat{\Theta}_i(k+1) = \hat{\Theta}_i(k) - \gamma_i \alpha_i \cos(\omega_i k) [J(\Theta(k)) - (1+h)\zeta(k)], \quad (3.2)$$

$$\Theta_i(k+1) = \hat{\Theta}_i(k+1) + \alpha_i \cos(\omega_i(k+1)), \quad (3.3)$$

With regards to the cost function, we have found empirically that the integral of time-weighted absolute error (ITAE) works better than the others we have considered, namely integrated square error (ISE), integrated absolute error (IAE), and the integrated time-weighted square error (ITSE). For the simulations, t_f was set to the simulation duration

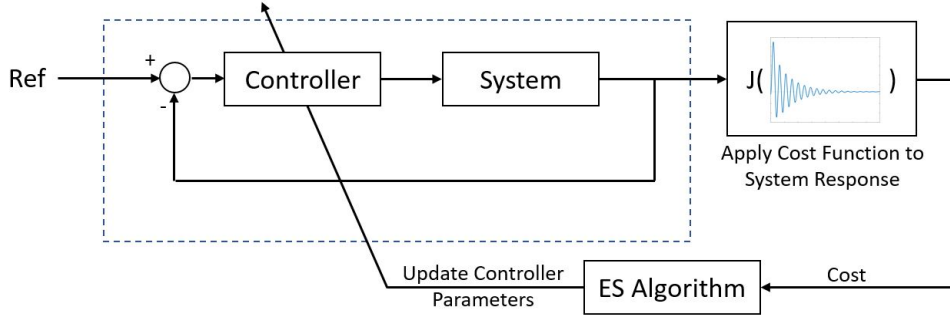


Fig. 3.1: Schematic of discrete extremum-seeking algorithm.

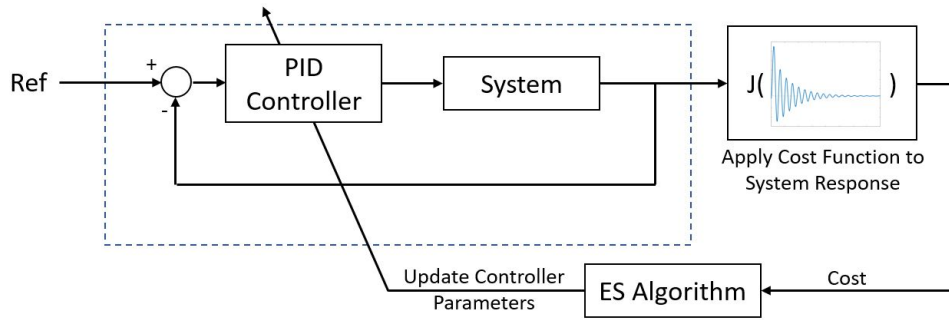
and t_0 was set to 0.

$$J(\Theta) = \frac{1}{t_f - t_0} \int_{t_0}^{t_f} t |e(t, \Theta)| dt \quad (3.4)$$

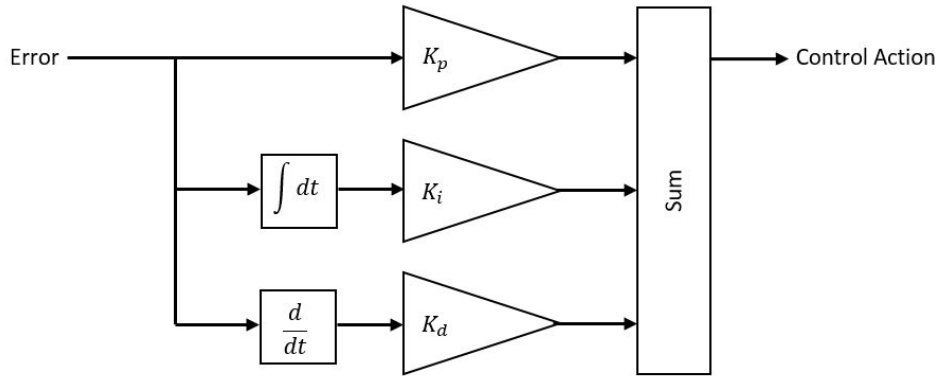
where $\Theta_i(k)$ is a vector of the controller parameters being tuned, $J(\Theta_i(k))$ is the cost function, $\zeta(k)$ is a scalar, γ_i is the adaptation gain, α_i is the perturbation amplitude, ω_i is the modulation frequency, and the subscript i denotes the i -th entry of a vector. During each iteration, the algorithm runs a step-response experiment on the system in the dashed box of Figure 3.1. The system response is then fed into the cost function to distill the performance of the controller into a single number. The ES algorithm then adjusts Θ in order to either minimize the cost function. A detailed explanation of this technique can be found in [19] and [20].

3.3 Hybrid PID-ES Controller

The next logical step is to pair the ES algorithm with a robust controller architecture and let the algorithm optimize the response based on the cost function. As a baseline, we extended the results in [19] and [20] to the Multiple-Input Multiple-Output solar sail system. Three independent PID controllers were used, one for each axis of rotation con-



(a)



(b)

Fig. 3.2: (a) Overview schematic PID-ES controller and (b) detailed schematic of PID controller.

trolled, per Figure 3.2b. Outputs of the linear controllers were capped to the maximum torque output of the primary ACS considered here.

The purpose of this controller was to capitalize on the promising results reported in [19] and [20], as well as to compare whether this controller structure was better-suited to fast convergence when combined with the ES algorithm. The computer code we used for this control scheme can be found in Appendix A.

3.4 Hybrid Fuzzy-ES Controller

In general, Fuzzy control involves applying membership functions to the quantity being controlled in order to determine how well the degree to which the membership function describes the current state of the variable to be controlled. The result of this “fuzzification” process is then evaluated to determine the correct action to be taken. For example, if the current temperature has a high degree of membership within the “it’s too cold” membership function and a low degree of membership in the “it’s too warm” membership function, then the control system would give more weight to the “increase the temperature” control action instead of the “decrease the temperature” control action. For our purposes, we have chosen to implement the standard Mamdani Fuzzy PD + I controller [21]. This controller uses the scaled difference between opposing high and low membership functions to create proportional and derivative control actions. An integration block adds steady-state tracking and disturbance rejection to the control system.

The parameters to be tuned via ES in this controller are the gains K_p , K_i , and K_d . While the membership functions could also be tuned to adjust the performance, that is beyond the scope of this work. The computer code we used for this control scheme can be found in Appendix B.

3.5 PSO-PID Controller

The Particle Swarm Optimization (PSO) algorithm was originally proposed in 1995 [22] and has become a very popular technique for global optimization problems. In our case, the PSO algorithm enables us to tune the controller gains to optimize the cost function

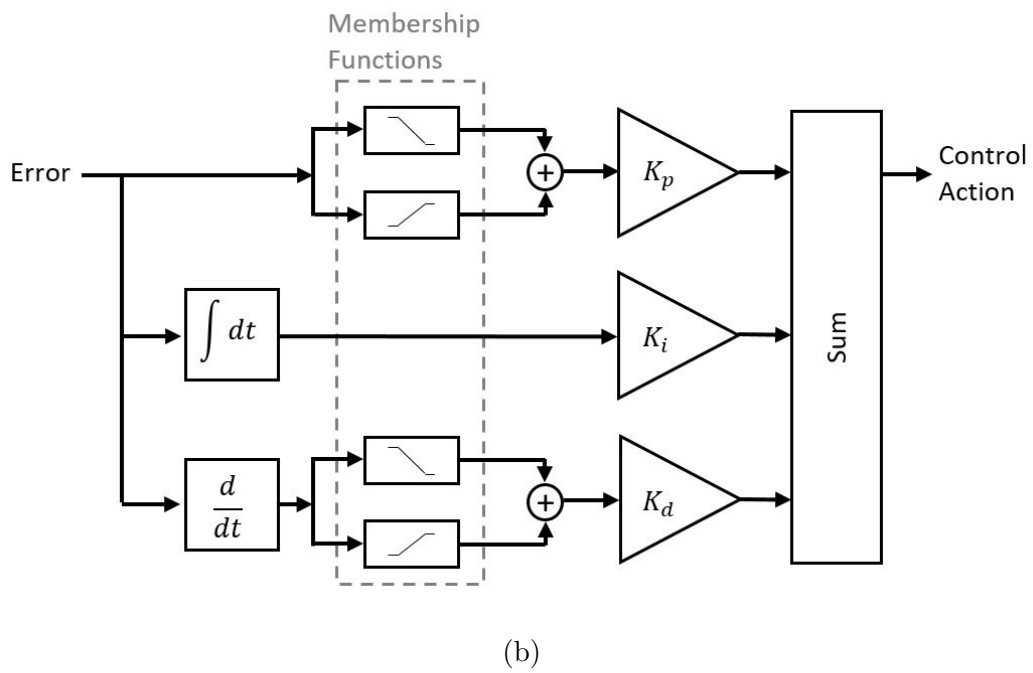
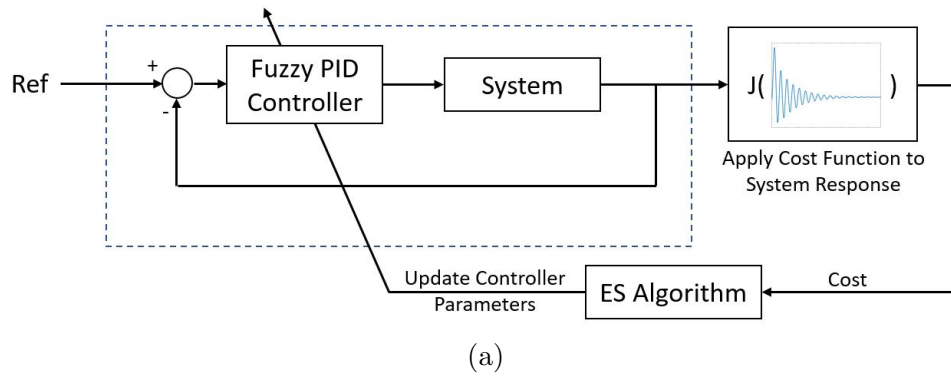


Fig. 3.3: (a) Overview schematic of Fuzzy-ES controller and (b) detailed schematic of fuzzy PD+I controller.

offline in the same fashion as [12] and create a performance baseline with a control technique that yields known good results. This performance baseline will then be used to judge the how well the ES algorithm can tune the response of the solar sail ACS.

The original PSO algorithm itself is fairly straightforward, though numerous improvements and tweaks have been proposed since its introduction. The algorithm begins by distributing a specified number of particles, N_{PSO} , throughout the operating space which is defined by the input variables that to be optimized. The operating space may have minimum and maximum bounds for any or all of the variables being optimized. Including such bounds can help to significantly reduce the amount of iterations necessary to converge to the global minimum (or maximum) of the associated cost function J . During each iteration, the algorithm runs through the following steps:

1. Calculate the cost function at the current location of each particle
2. Update the velocity of each particle based on:
 - (a) The previous velocity of the particle
 - (b) The best cost found by all particles in the swarm
 - (c) The best cost found by that particular particle so far
 - (d) The best cost found by neighboring particles
3. Update particle positions
4. If any particles leave bounds, move particles back to boundary edge
5. If end condition is met, terminate algorithm
 - (a) Desired number of iterations has been reached
 - (b) Desired run time has been achieved

- (c) Change in cost function over last X iterations $\epsilon < \epsilon_{min}$

The latest and greatest PSO implementations incorporate additional tweaks to improve convergence speed and accuracy, but the core of the algorithm remains the same. The code used to implement this control scheme can be found in Appendix F.

3.6 Simulation Results

To generate the following results, the ES algorithm was initiated with Zeigler-Nichols gains obtained from [12]. These gains represent a stable, but poorly tuned controller to showcase how ES can improve the controller performance. Other starting gains were also considered, but the goal of these simulations was to eliminate entirely, or at least minimize, the efforts involved in modeling the dynamical system well enough to use offline tuning methods.

3.6.1 PID-ES vs Fuzzy-ES

First, comparing Figures 3.4 and 3.7, we notice that both the PID-ES and Fuzzy-ES control schemes achieved very similar results within the 2000 iterations they were run for, though the Fuzzy PD+I controller cost function in Figure 3.9 ended up just a little bit below that of the PID controller in Figure 3.6. It is also worth noting from Figures 3.5 and 3.8 that neither controller achieved its cost function minimum at the last iteration (2000). This is an inherent downside of ES - the algorithm never achieves the true optimum, but instead oscillates around it.

Next, we see from Figures 3.4, 3.7, and 3.10 that while the performance of the primary ACS did improve on all 3 axes, the performance gains were not uniform and still could

not match the results of a PSO-PID scheme discussed in [12]. We replicate those results in Figures 3.10, and they clearly show that significantly better performance is possible with a second-order linear controller. To create this controller, we utilized MathWorks' Matlab[®] Global Optimization Toolbox PSO implementation. Simulations were run on a computer with an Intel i7 quad-core processor, 16GB of RAM, and Windows 10[®] x64. We consider the gain evolutions as the ES algorithm steps through iterations in Figure 3.8. The rate of change in the gains is neither uniform, nor is it constant with iterations. The majority of the change occurs in the first 500 or so iterations. This is confirmed by the cost function plot in Figure 3.9. The rate of decay in the cost function quickly drops and bang for your buck on subsequent iterations drops off significantly. This brings us to one of the main problems with the ES algorithm: choosing gains. As explained in more detail in [23], without a model of the system it is impossible to determine each of the gains a priori. Choosing gains that are too aggressive will cause the algorithm to choose unstable gains, which both breaks the algorithm and can damage the craft. The consequence of this is that each variable converges to the local minimum at a different rate and, in a more practical sense, prevents the system from converging in a reasonable amount of time. Consider this as an extreme case where, since each iteration takes roughly 30,000 seconds (8hours, 20 mins), 2000 iterations running nonstop would take almost 2 years to complete. This is also ignoring the reality that the spacecraft would run out of fuel long before then.

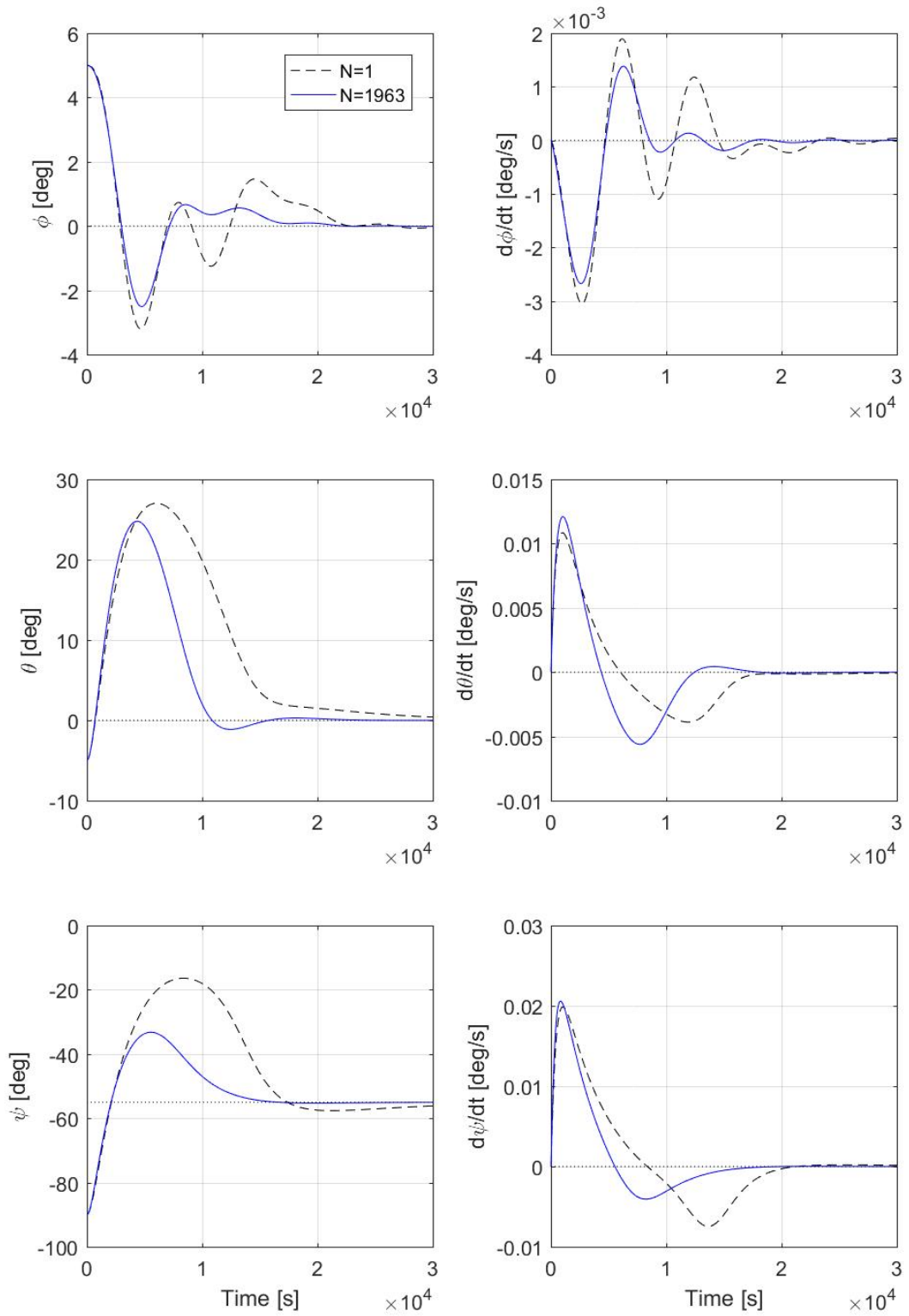


Fig. 3.4: Simulation of PID-ES controller, time response. Euler angles (left) and Euler rates (right) in LVLH frame.

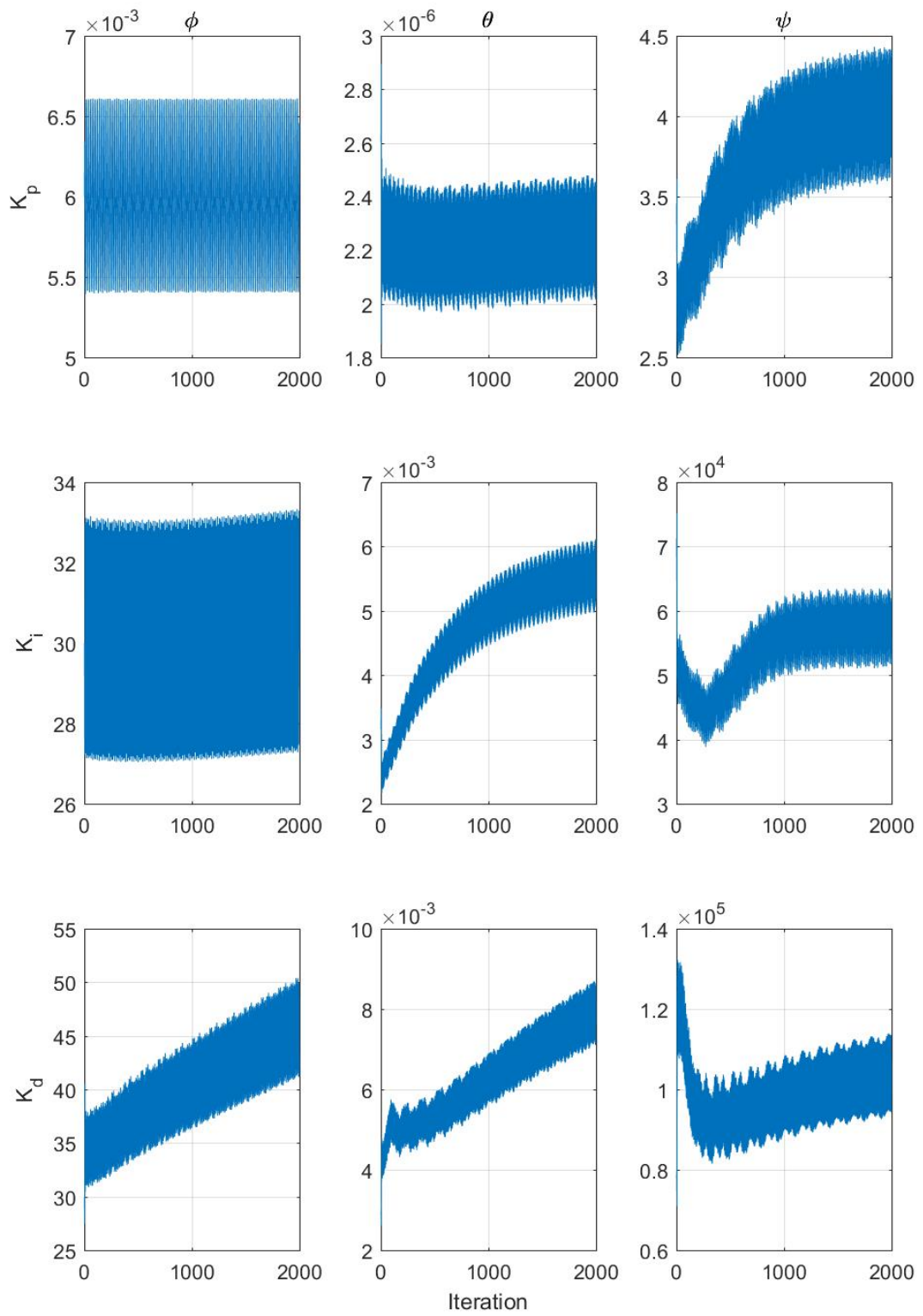


Fig. 3.5: Simulation of PID-ES controller, gain evolutions. Gain evolutions for PID controller by controlled axis of rotation.

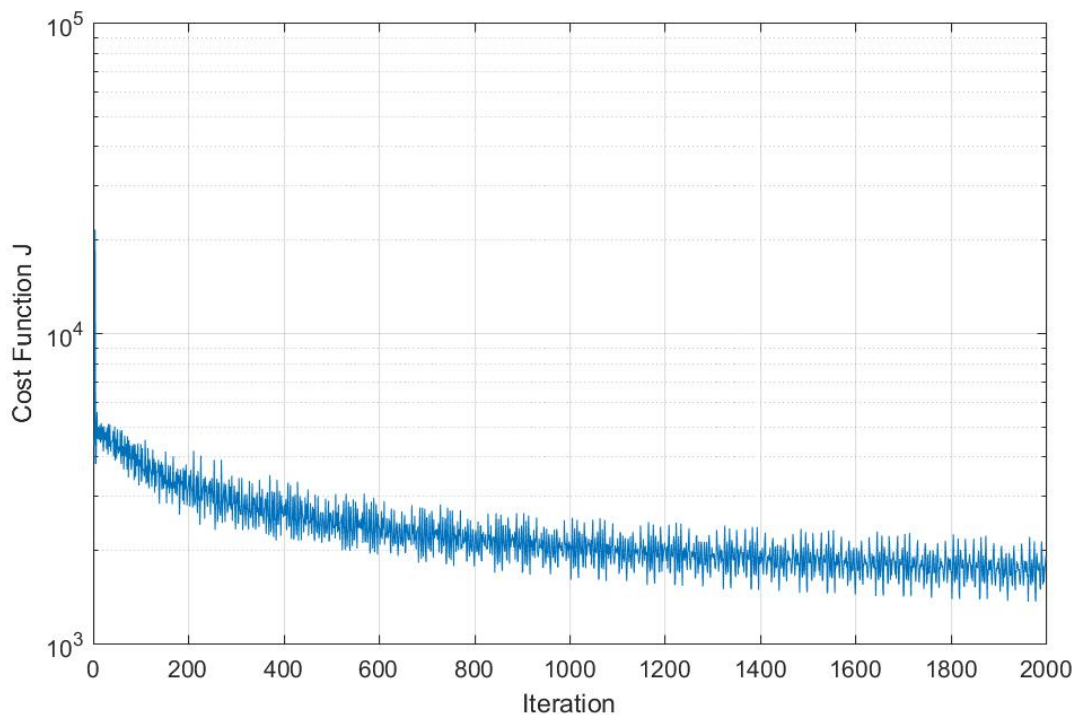


Fig. 3.6: Simulation of PID-ES controller, cost function evolution.

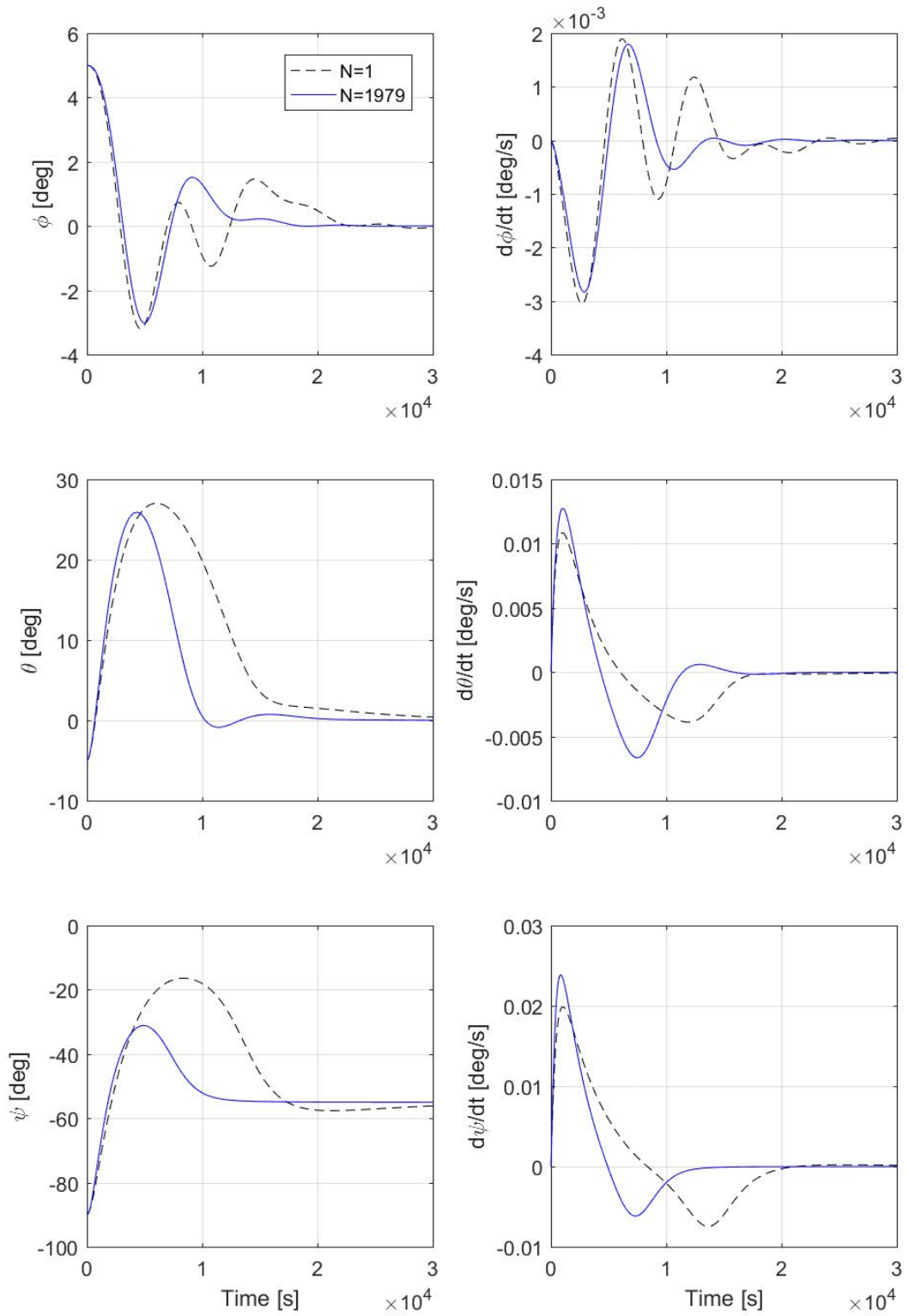


Fig. 3.7: Simulation of Fuzzy-ES controller, time response. Euler angles (left) and Euler rates (right) in LVLH frame.

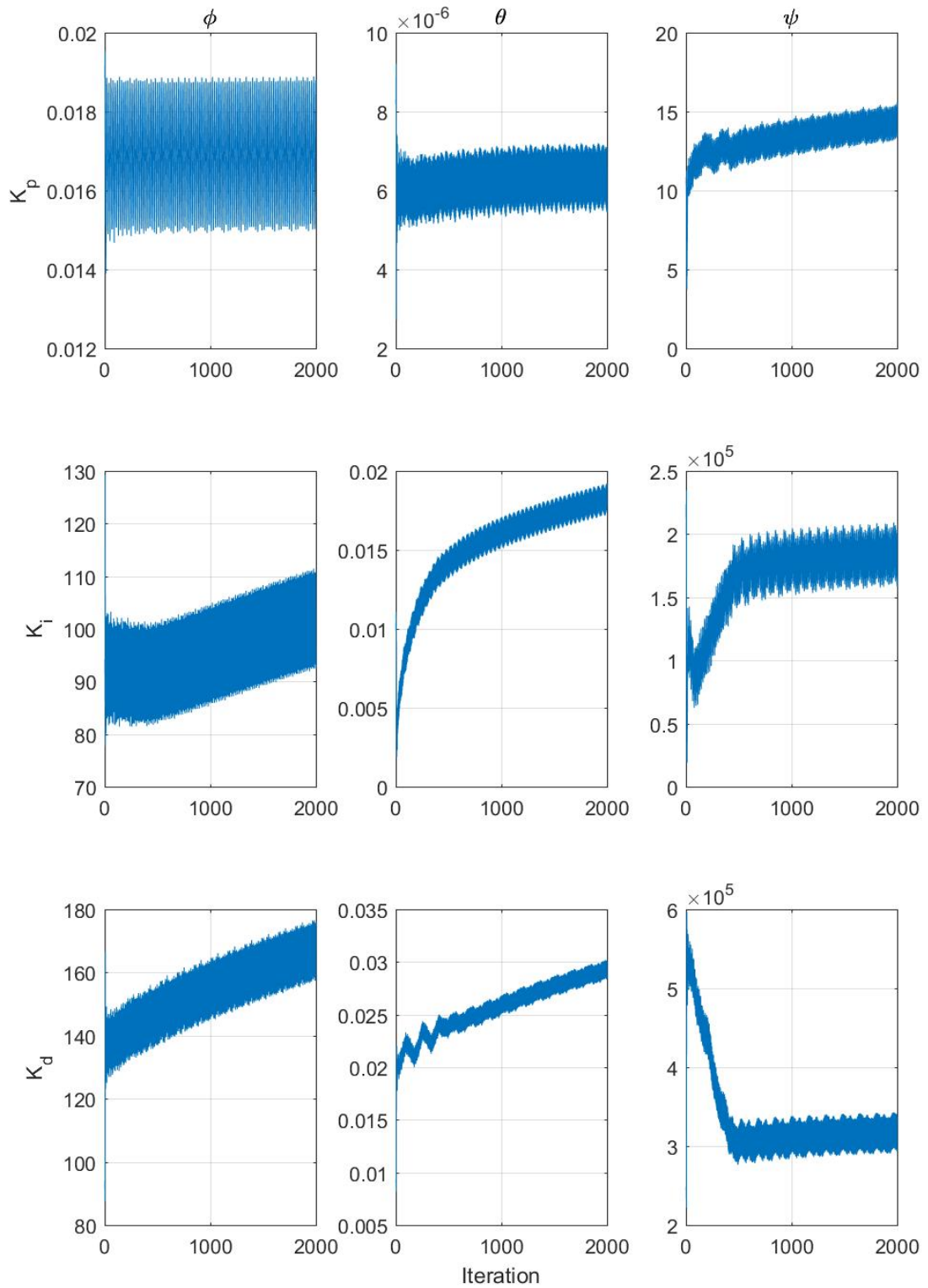


Fig. 3.8: Simulation of Fuzzy-ES controller, gain evolutions. Gain evolutions for fuzzy-ES controller by controlled axis of rotation.

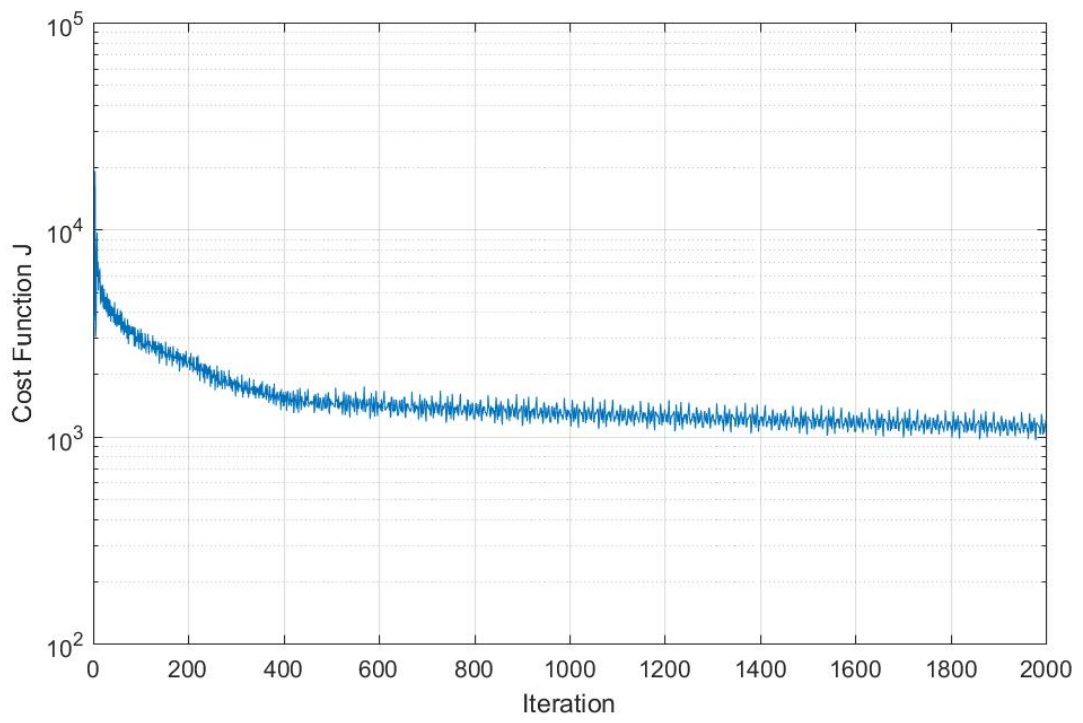


Fig. 3.9: Simulation of Fuzzy-ES controller, cost function evolution.

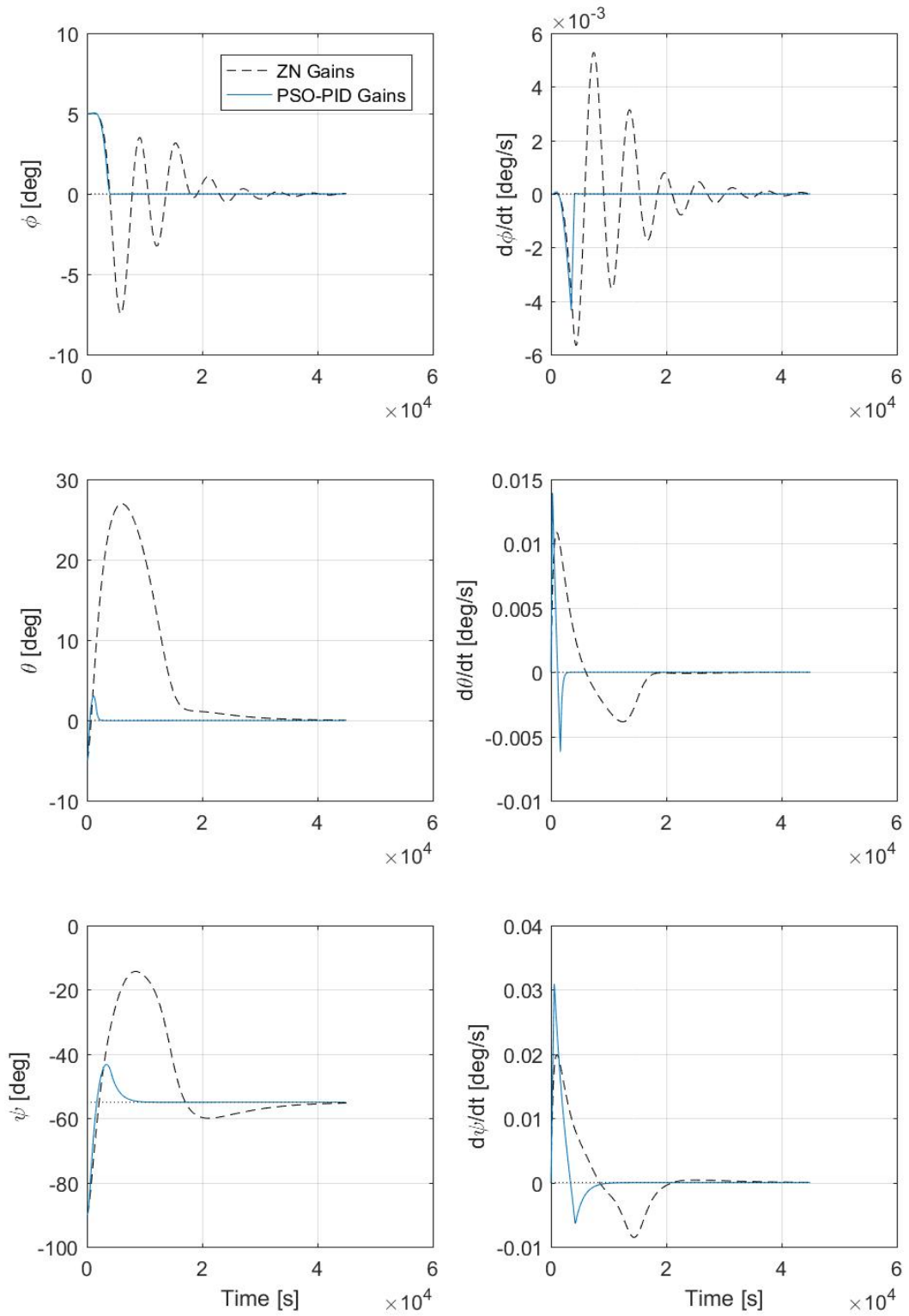


Fig. 3.10: Simulation of PSO-PID controller, time response. Euler angles (left) and Euler rates (right) in LVLH frame.

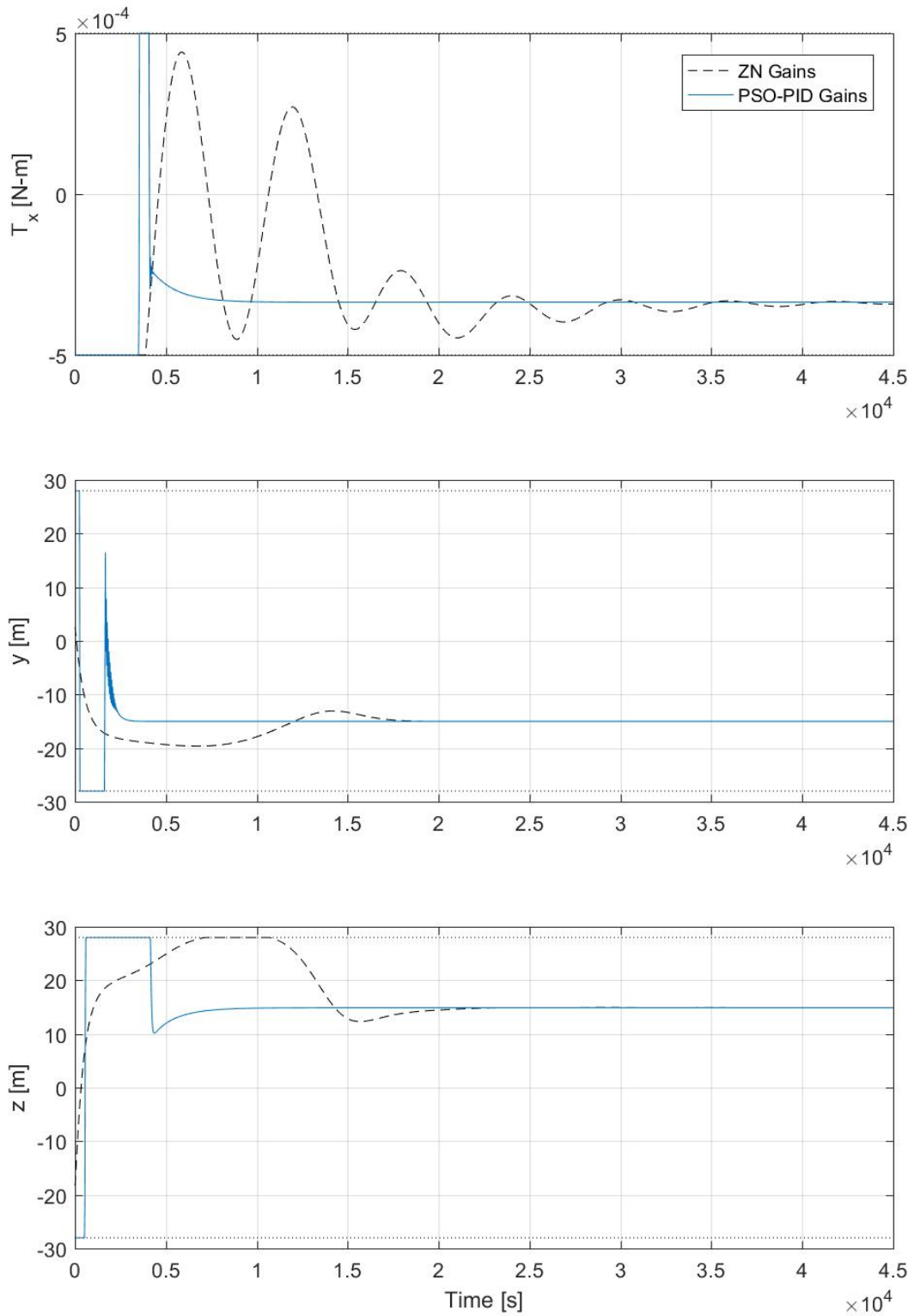


Fig. 3.11: Simulation of PSO-PID controller, control action time response. Control action (left) and tracking error (right). Control limits are $-5 \times 10^{-4} \leq T_x \leq 5 \times 10^{-4}$, $-28 \leq y \leq 28$, and $-28 \leq z \leq 28$.

3.6.2 ES Convergence Improvement

We attempted to mitigate the drop-off in the rate of convergence by modifying the way the cost function is calculated. Because the ES algorithm is a gradient-based algorithm, the step size, or rate of convergence, is necessarily a function of the gradient. Let's consider the simple one-dimensional discrete ES example shown in Figures 3.12a and 3.12b. The axes on the left show the simple cost function $J = |x - 1|$, with the cost on the horizontal axis and the input variable Θ on the vertical axis. The axes on the right show the convergence of the ES algorithm to the local minimum over iterations. The dashed white line represents the minimum of the cost function.

Another way to look at this plot is that the ES algorithm is moving on a 3-dimensional "V". The right plot represents the "V" as seen from the top view, while the left graph shows the same "V" from the left right side. For those more familiar with reading engineering drawings, this graph is in first angle projection. The color map behind the ES trajectory line (red) on the right represents a topographic map of the cost function as seen from "above".

All ES parameters are kept the same between the two figures, the only difference between the two figures is that in the latter the cost function is scaled by a factor of 0.25. The effect on the rate of convergence is quite pronounced in this case, with the former case converging in less than 15 iterations and the latter finally converging in around 50. In the case of this simple example, we could compensate the reduced convergence rate by simply increasing the gains (γ).

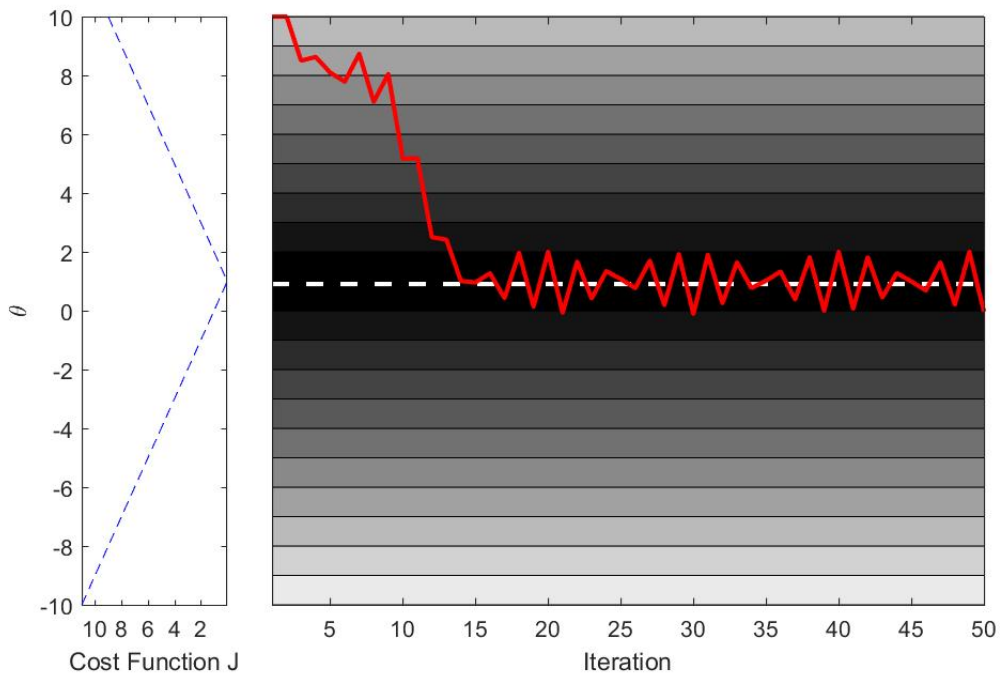
Next, let us again consider the case of the solar sail spacecraft. The cost function is only defined so long as we maintain stability, so if the ES algorithm happens to choose gains that are not stable, not only will the ES algorithm not converge, but the result may also damage or destroy the craft being controlled. Additionally, we also see from the

convergence of the ES algorithm that the gradient of the cost function is initially quite steep, forcing us to choose somewhat conservative gains to maintain stability. However, as the algorithm continues to run, the gradient of the cost function begins to taper off and the convergence rate drops. If it was possible to either increase the gains of the ES algorithm or increase the gradient of the cost function, we would be able to increase the rate of convergence.

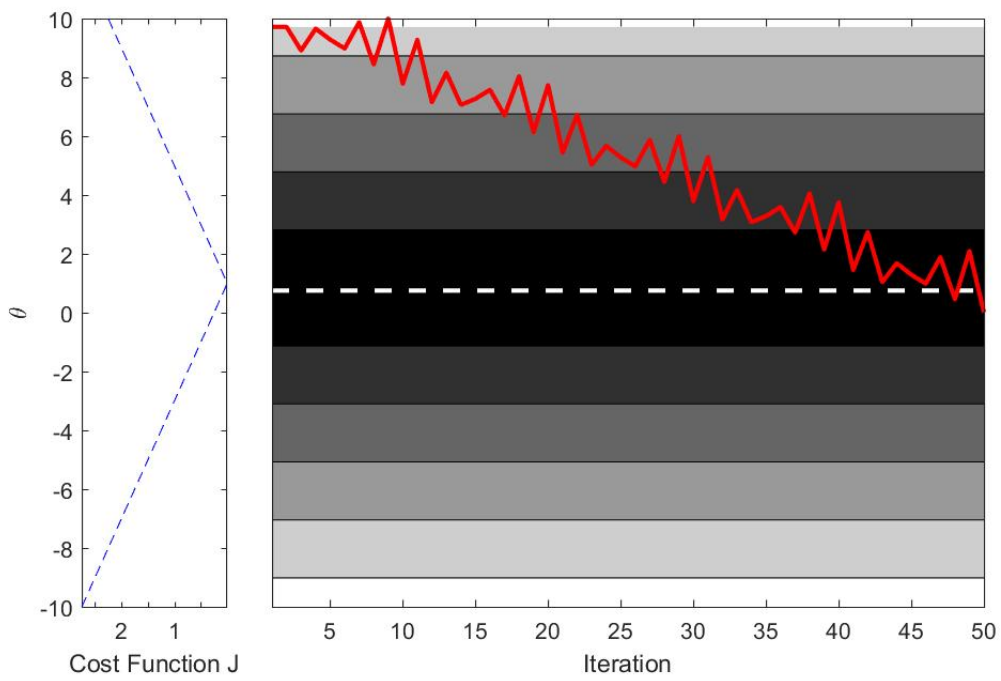
The approach we examined was to increase the gradient of the cost function. This approach was chosen mainly to avoid the difficulties inherent in choosing the starting gains for the ES algorithm. There is neither a simple nor systematic way of determining how much to push the gains before loss of stability other than trial and error, thus attempting to increase them on-the-fly would always run a high risk of failure.

The approach we took was to periodically reduce the length of the simulation over which the ITAE cost function was applied so that the system almost stabilized within the time limit. In this way, the cost function would be most sensitive to the endpoint of the simulation, and controller gains which reduced settling time would yield a larger impact on the overall cost function. However, by doing this we are asking the ES algorithm to prioritize settling time over steady-state error and overshoot, so results may vary.

A comparison of the efficacy of this cost function length reset scheme can be found in Figures 3.13, 3.14, and 3.15. To summarize, it offers modest gains, but also does not require any other modification of the ES algorithm. For this case we ran the ES algorithm for quite a long time ($N = 2000$). If we were to stop earlier, we would notice that the ES algorithm with the length reset enabled prioritizes settling time as predicted (see Figures 3.16, 3.17, and 3.18).



(a)



(b)

Fig. 3.12: 1-dimensional ES example, (a) fast convergence and (b) slow convergence rates are achieved by only changing cost function gradient. Θ is the independent variable.

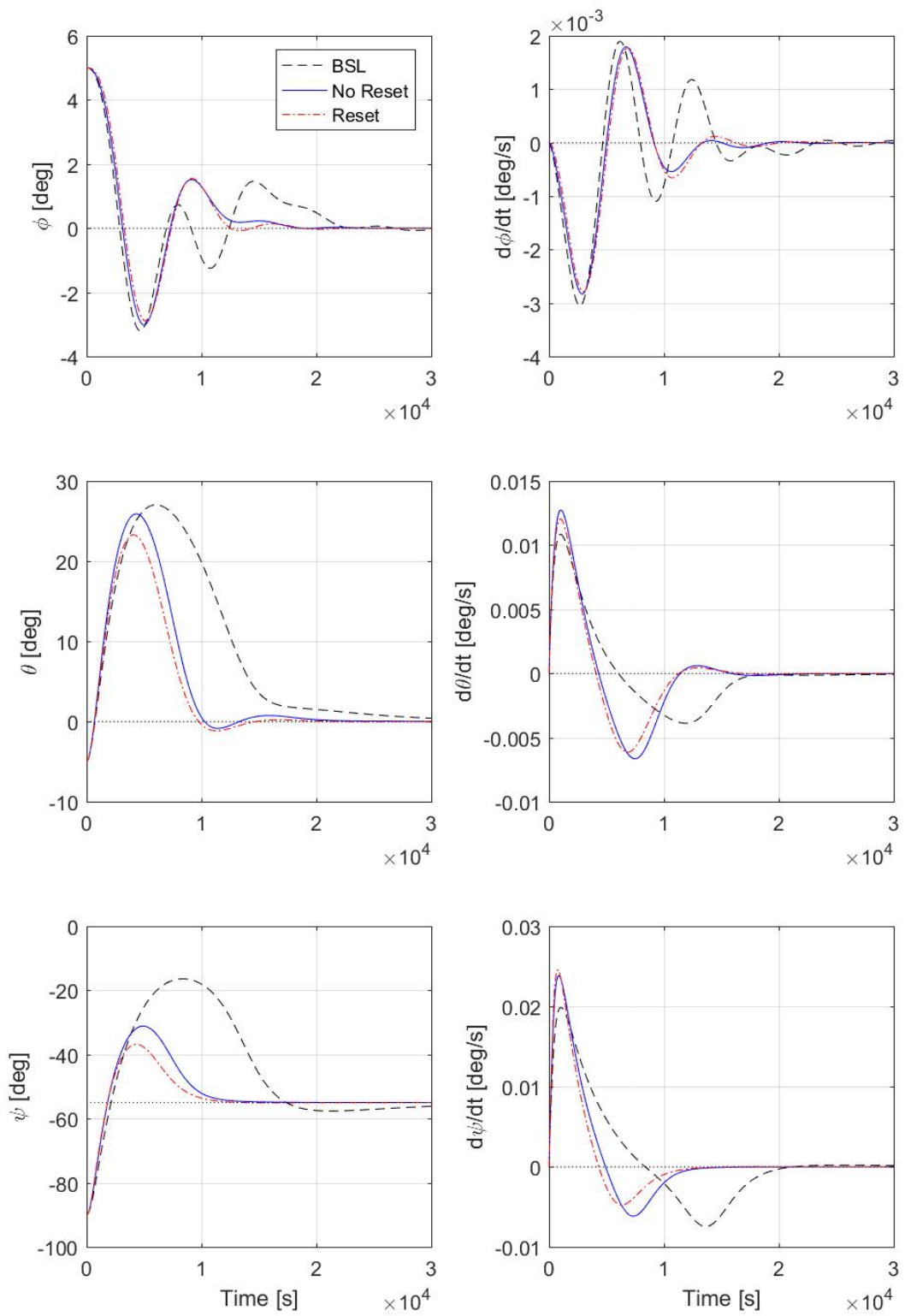


Fig. 3.13: Simulation of improved ES convergence algorithm (2000 iterations), time response for fuzzy-ES controller. Euler angles (left) and Euler rates (right) in LVLH frame.

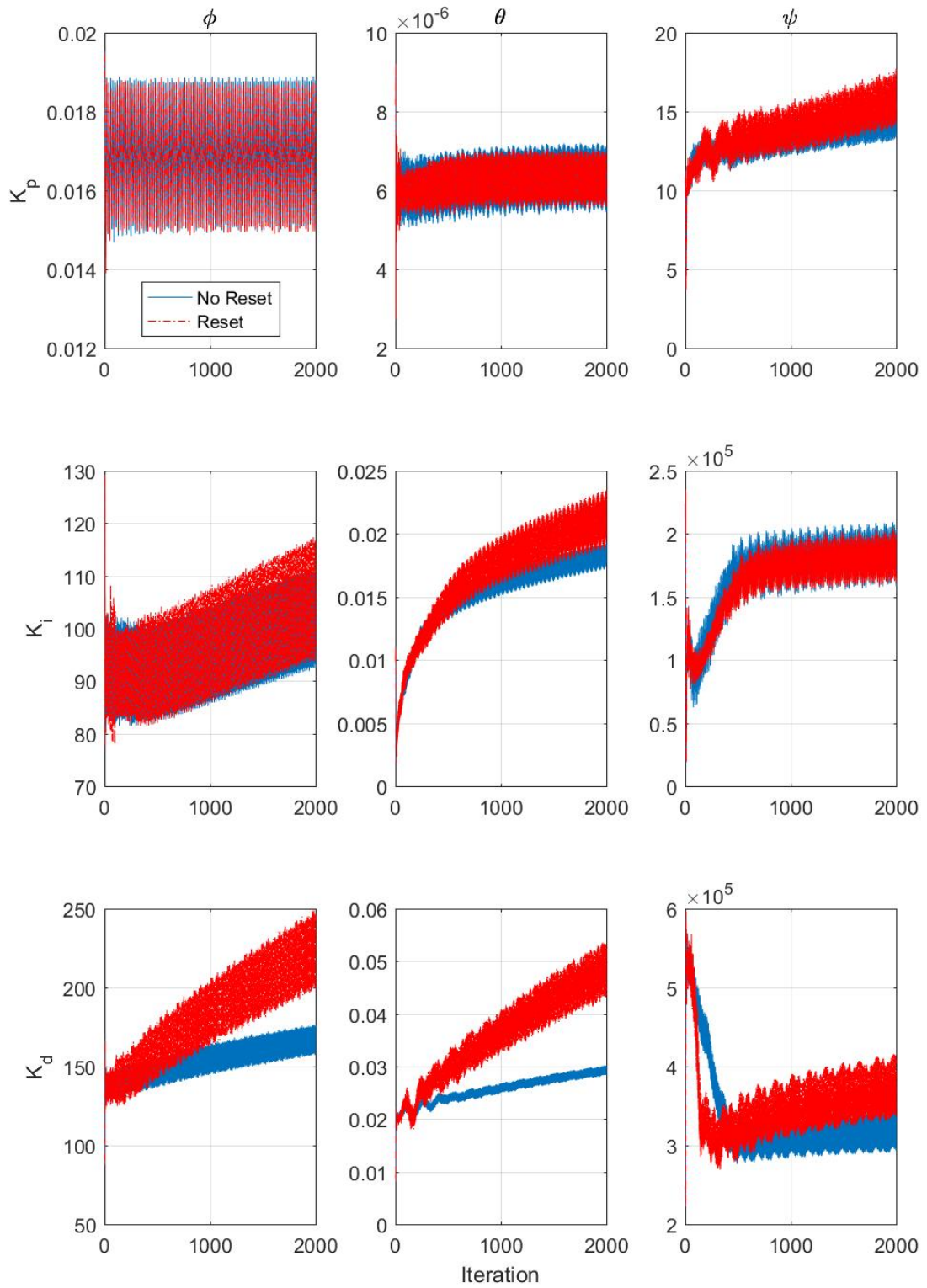


Fig. 3.14: Simulation of improved ES convergence algorithm (2000 iterations), gain evolutions for fuzzy-ES controller by controlled axis of rotation.

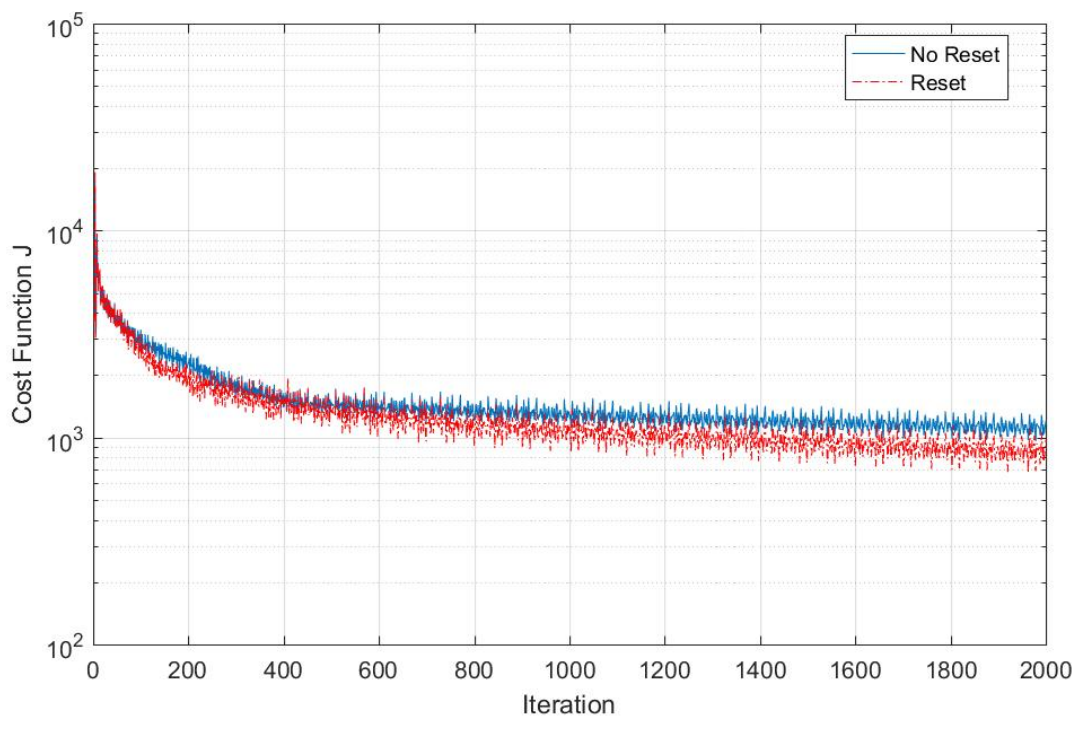


Fig. 3.15: Simulation of improved ES convergence algorithm (2000 iterations), cost function evolution for fuzzy-ES controller.

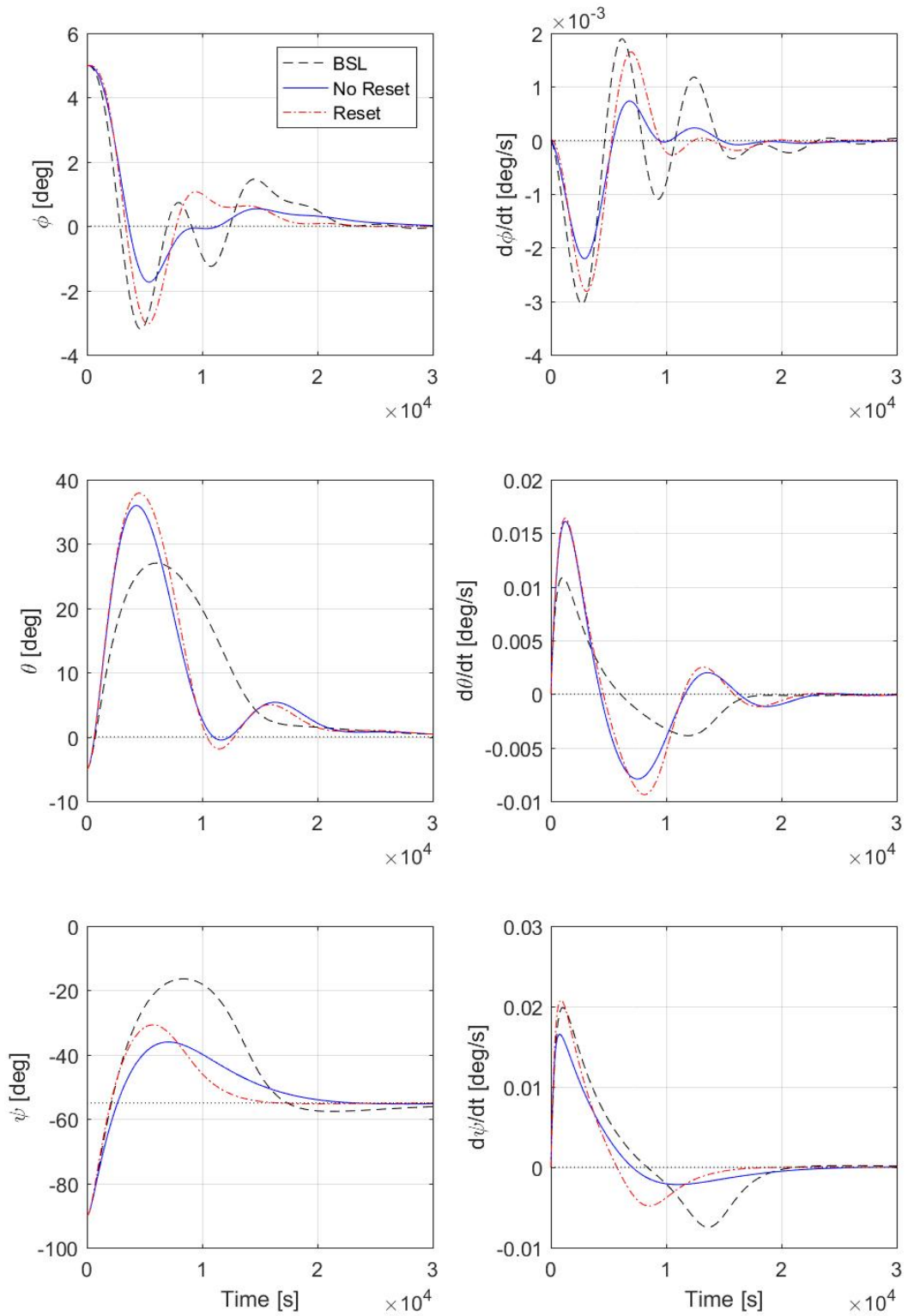


Fig. 3.16: Simulation of improved ES convergence algorithm (200 iterations), time response for fuzzy-ES controller. Euler angles (left) and Euler rates (right) in LVLH frame.

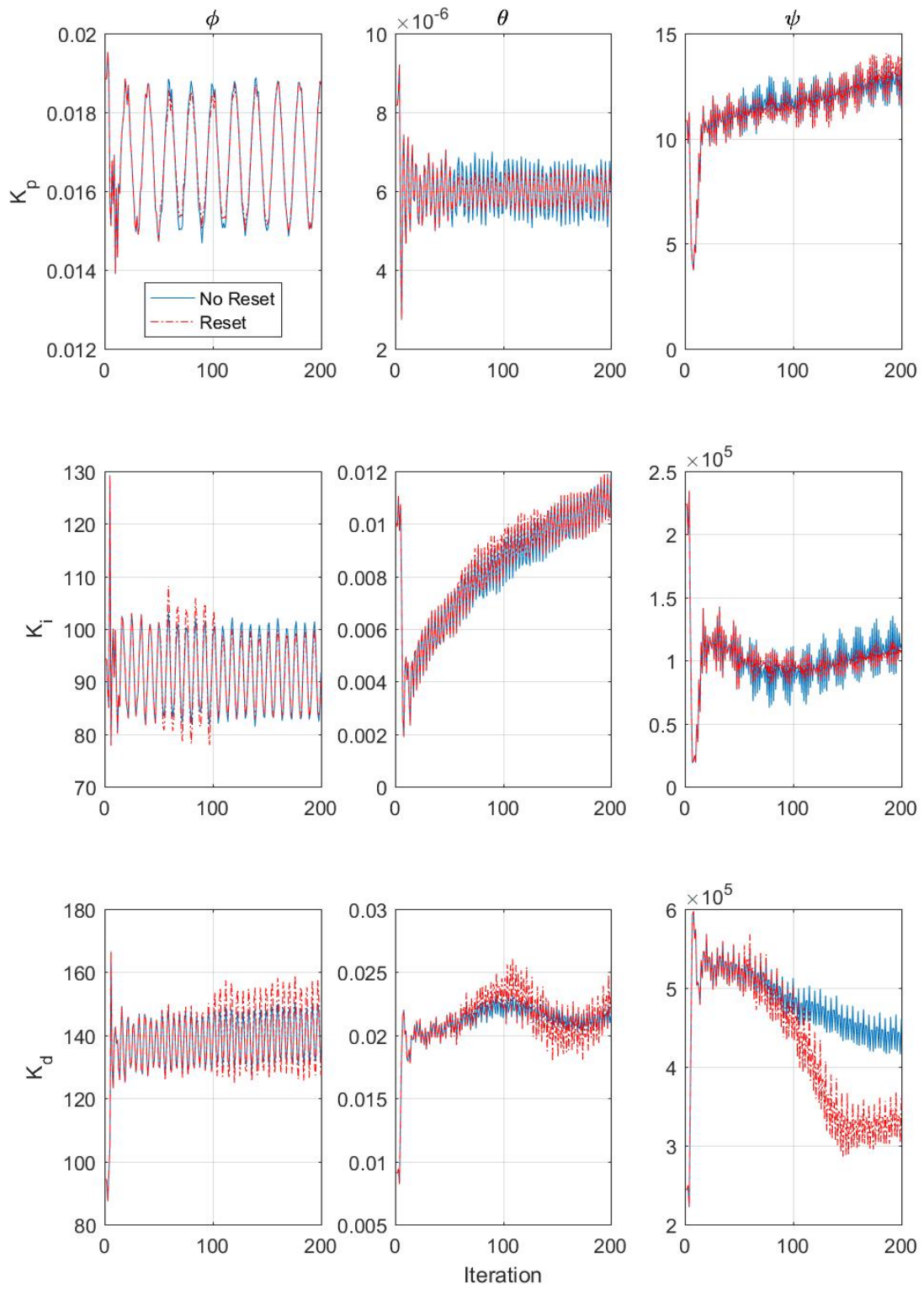


Fig. 3.17: Simulation of improved ES convergence algorithm (200 iterations), gain evolutions for fuzzy-ES controller by controlled axis of rotation.

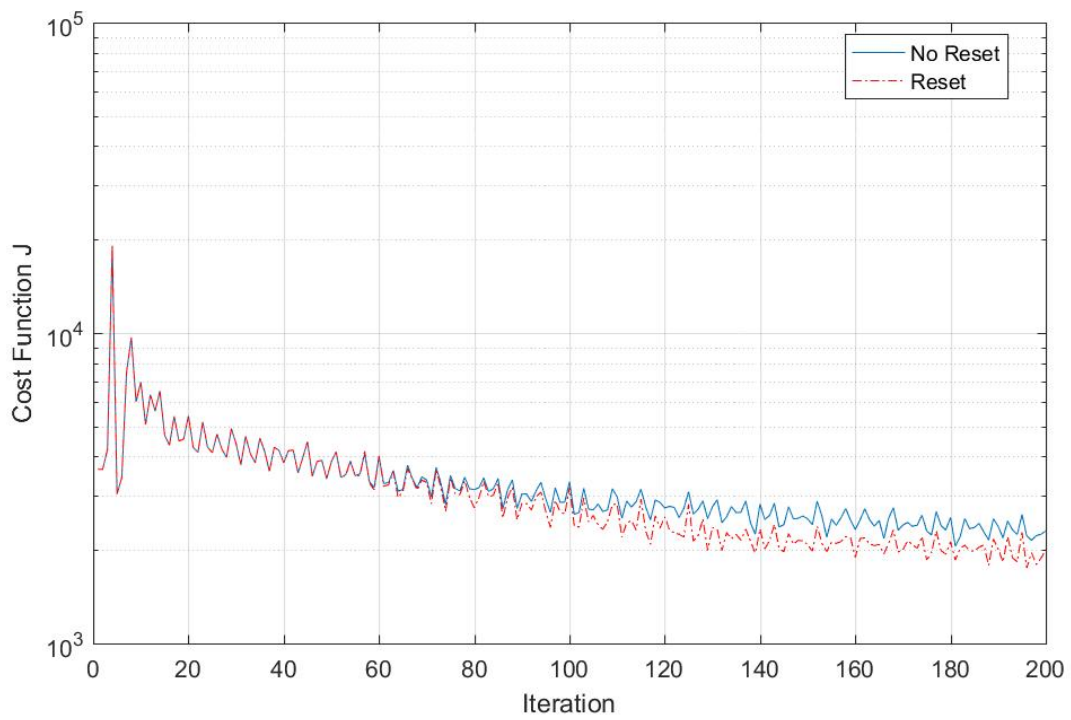


Fig. 3.18: Simulation of improved ES convergence algorithm (200 iterations), cost function evolution for fuzzy-ES controller.

4. Fuzzy Controller (Type I and Type II)

Controllers based on fuzzy logic have been a popular choice for taming nonlinear systems since the late 1980's due to their intuitive structure and ease of tuning [24]. Basic descriptions of how a human operator would respond to a given set of inputs can be directly converted into a set of rules. Once the desired behavior is set, the controller can then be tuned with parameters that are much easier to understand than alternative nonlinear control approaches.

4.1 Introduction and Theory

To begin, let us consider the four basic parts of a simple type-1 fuzzy logic controller as shown in Figure 4.1: the so-called fuzzifier, inference engine, rule set, and de-fuzzifier. To help illustrate how a fuzzy logic controller operates, we will consider a simple water heater. The input of the controller is the difference in the current water temperature relative to a fixed reference (error) and the output of the controller is the duty cycle of an electric heating element.

While it may seem counterintuitive at the moment, let us consider the rules of the fuzzy logic controller first. These take the form of if-then statements. The “if” portion is

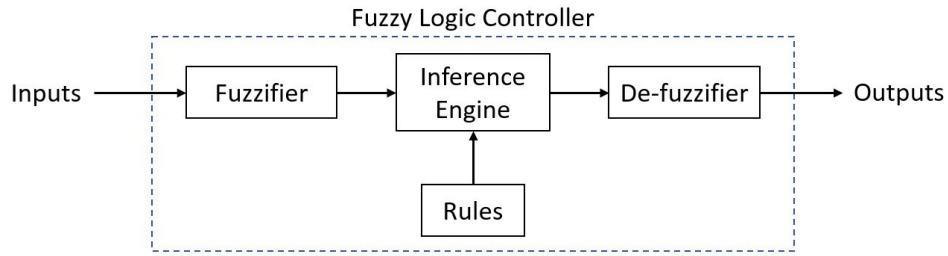


Fig. 4.1: Block diagram of a general type-1 fuzzy logic controller.

If	Then
Water too cold	Set heater output to 100%
Water too hot	Set heater output to 0%
Water just right	Set heater output to 50%

Table 4.1: Example type-1 fuzzy rules.

referred to as the antecedent and the “then” portion is called the consequent. For the case of our water heater, let us write down a set of heuristic rules that describe how we would like the system to behave (see Table 4.1).

Now that we see the rules, we see that we need a way to mathematically describe how well the antecedent applies to the current state of the system. In other words, what is “too hot” or “too cold”? This is precisely the job of the Fuzzifier - it converts the inputs of the controller into a fuzzy set that describes how well each description matches the current state of the system. The values of the fuzzy set range from 0 (does not describe at all) to 1 (describes perfectly). There is one member in the fuzzy set for each description, or membership function as they are commonly known. As a simple example, consider the membership functions for the water heater shown in Figure 4.2.

Now that we have sorted out how to apply the inputs to the rules, we actually need to do so. This process happens in the inference engine - input membership functions are converted to output membership functions. The output membership functions can be defined similarly to the input membership function (Mamdani-Type Inference [25]),

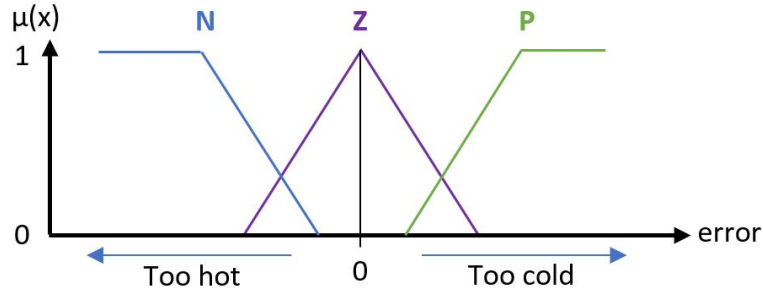


Fig. 4.2: Example of membership functions for a water heater.

crisp values (zero-order Sugeno-Type Inference [26]), or linear functions of the inputs (n-order Sugeno-Type Inference [26]). For simplicity, let us consider that the output membership functions are crisp values as described in our rules for the water heater.

The second job of the inference engine is to assign the applicability, or firing strength, to the output membership function for the given set of inputs to the controller. For the case of our simple water temperature controller, the firing strength is the value of the relevant input membership function. However, when multiple antecedents are considered for a rule, OR and AND logical operators are used to determine the firing strength of each rule.

The last step of fuzzy logic controller is to aggregate and de-fuzzify the output membership functions into a single crisp control action. For the case of zero-order output membership functions, aggregation and de-fuzzification can be a simple weighted average $y = \frac{\sum_{i=1}^N \alpha_i O_i}{\sum_{i=1}^N \alpha_i}$, where O_i is the output membership function, α_i is the firing strength of the rule, and N is the number of output membership function. All of the steps are summarized in Figure 4.3.

One criticism of the type-1 fuzzy controller is that while the control scheme can accommodate uncertainty in the value of the control inputs, the input and output membership functions must have crisp definitions. The solution to this is to incorporate uncertainty

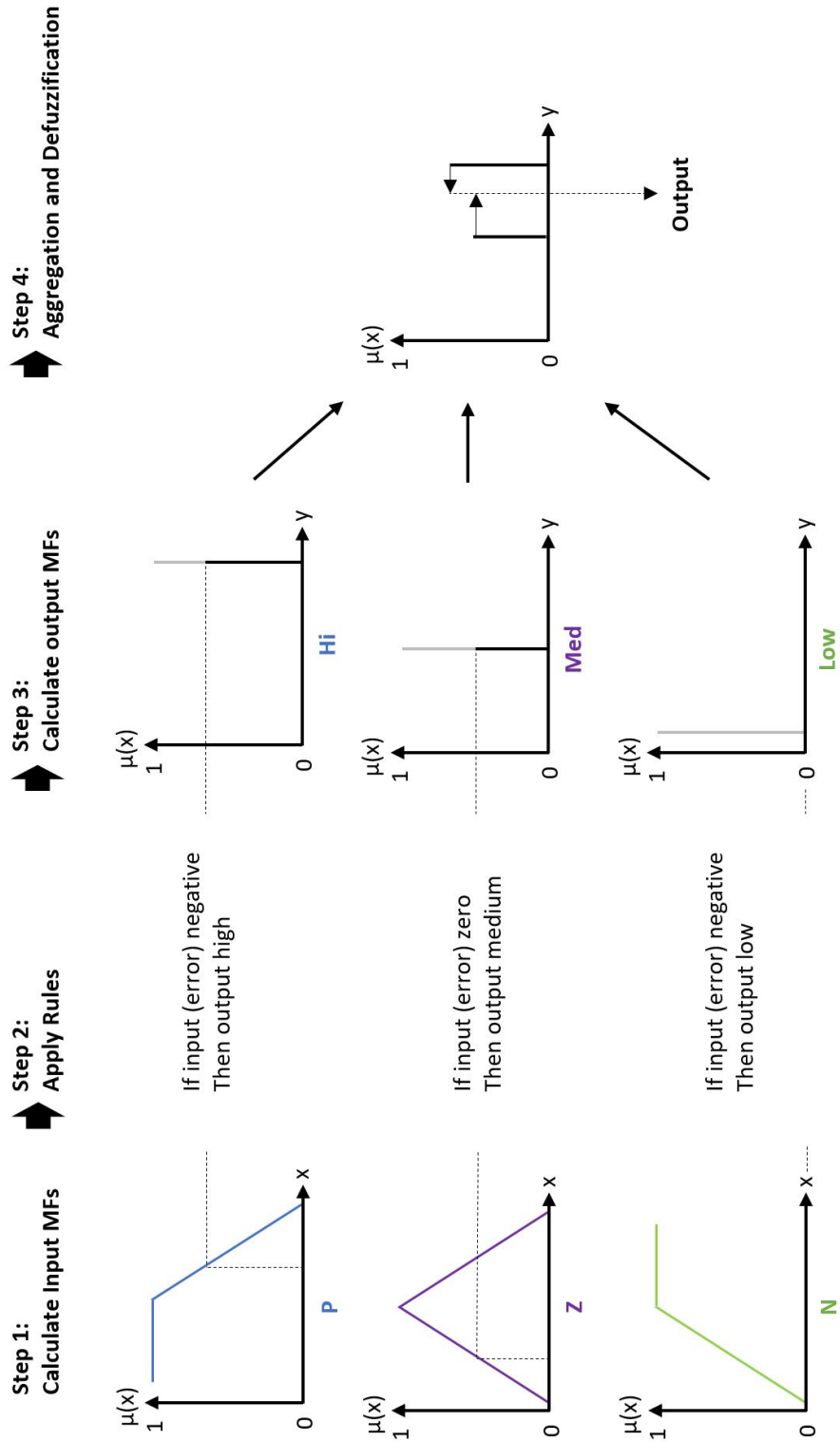


Fig. 4.3: Fuzzy logic decision making.

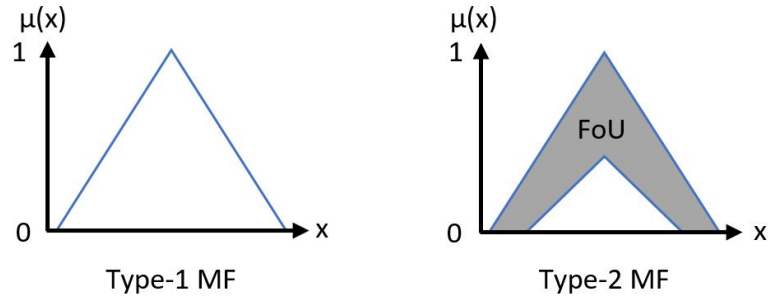


Fig. 4.4: A comparison of type-1 and type-2 fuzzy logic membership functions.

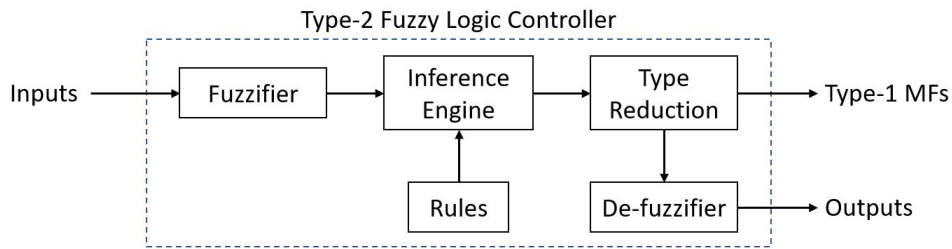


Fig. 4.5: Type-2 fuzzy logic controller overview.

within the membership functions themselves. In Figure 4.4, we see how the Type-2 membership function is no longer a curve, but is instead a region. The shaded area between the upper and lower membership functions is referred to as the Field of Uncertainty (FoU).

While there are several ways of defining the FoU, we chose to use the interval method. This approach specifies the upper and lower bounds of the FoU using type-1 membership functions. Alternate methods include defining a 3-dimensional FoU volume from which “slices” are taken. More information about these alternate approaches can be found in [16].

The second major difference between type-1 and type-2 fuzzy logic controllers is the addition of a type-reduction step shown in Figure 4.5. Because the output of the controller must be a crisp value, this step is needed to find the true type-1 membership

function within the FoU of the type-2 membership function. After type reduction, the crisp control action can be calculated in much the same way as it is for type-1 fuzzy controllers.

In a practical sense, type-2 controllers give the designer additional tuning knobs without the need to increase the number of rules. While this may not be of great benefit in the case of simple systems, if you consider a 3-input 1-output system, we see that 3 membership functions per input requires $3^3 = 27$ rules, whereas 4 membership functions needs $4^3 = 64$ rules. However, the ability to shape the controller response without greatly increasing complexity does come at a price - type reduction is computationally expensive because it is an iterative algorithm. The current gold standard for type reduction is the Enhanced Iterative Algorithm with Stop Condition (EIASC) because it is mathematically well-grounded. The additional computational overhead of this operation can often make it more advantageous to simply add more membership functions to a type-1 fuzzy controller.

To overcome the computational problems posed by type reduction of type-2 membership functions, numerous methods have been proposed [27, 28]. Several, like the Nie-Tan method we used (equation 4.1), combine type reduction and defuzzification into a single step and greatly reduce the computational overhead of calculating the crisp output of the type-2 fuzzy controller. Unfortunately, the increase in speed comes at the cost of lost information about the type-reduced membership functions. In addition, such approaches, while effective in practice, do not stand on quite as firm mathematical footing as the EIASC.

$$y = \frac{\sum_{i=1}^N y_i (\overline{O}_i + \underline{O}_i)}{\sum_{i=1}^N (\overline{O}_i + \underline{O}_i)} \quad (4.1)$$

The computer code we used for these two control schemes can be found in Appendix

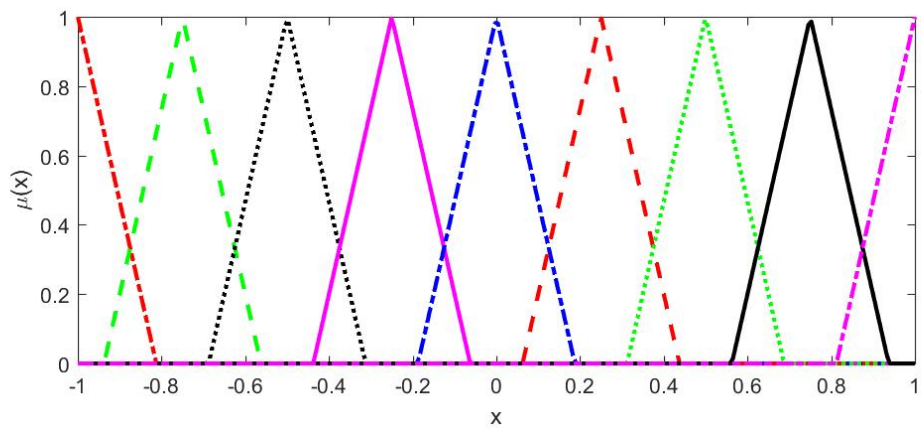
C, D, and E.

4.2 Simulation Results

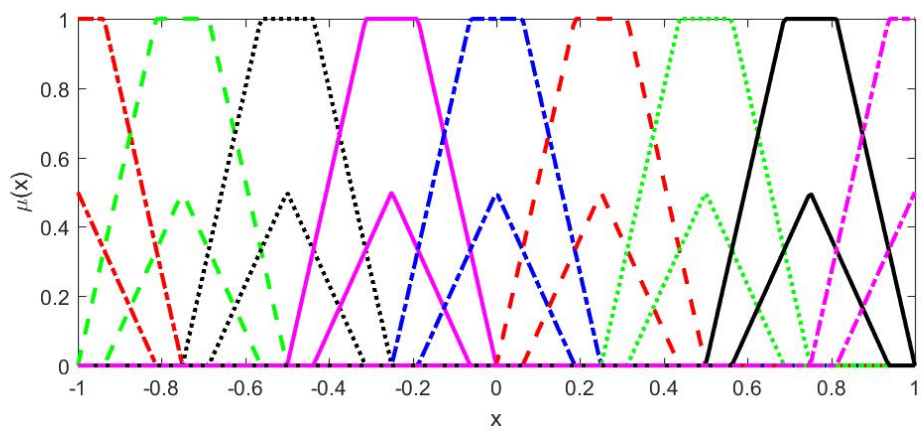
The structure of the type-1 and type-2 fuzzy logic controllers for the solar sail craft was similar to that of a PID controller. Each rotational axis was given an independent fuzzy controller with 3 inputs: e , de/dt , and $\int edt$. Each input was assigned 9 membership functions shown in Figure 4.6b for a total of $9^3 = 729$ rules. Both 7 and 5 membership functions were also examined, as well as different shapes and size of FoU, but no significant differences were observed. Gaussian membership functions were also examined, but again, no significant differences were observed. Type-1 membership functions used for comparison are shown in Figure 4.6a.

The type-1 fuzzy controller was designed first using particle swarm optimization (PSO) with the linearized equations of motion 2.10. Gains were applied by scaling the output membership functions instead of the standard practice of scaling the input signal to effectively “stretch” or “compress” the membership functions relative to the input variable [29]. The downside of the standard approach is that when the scaling factors differ by orders of magnitude, it tends to either immediately saturate the control action or only use a small fraction of the total number of input membership functions. Once optimized gains were obtained, they were plugged directly into the type-2 controller unmodified. The goal of this series of simulations was to find direct benefits of type-2 control over type-1 without introducing additional confounding factors. The following simulations were run using the full nonlinear equations of motion 2.1 to help understand how the controllers handled model mismatch that would be present in any system. Both controllers were originally tuned using PSO using the linearized equations of motion.

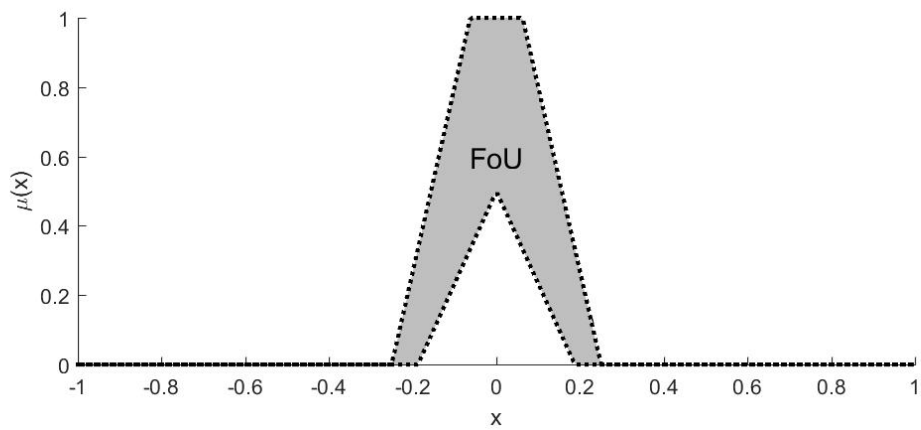
When considering the baseline trajectory in Figures 4.7 and 4.8, both type-1 and type-2



(a)



(b)



(c)

Fig. 4.6: (a) Type-1 MFs and (b) Type-2 MFs used for simulation. (c) Single membership function from (b) shown with FoU. x is the input variable and $\mu(x)$ is the degree of membership for each MF.

fuzzy controllers yield respectable results. One key difference is that the FoU has the property of smoothing sharp edges the control surface and generally making controller gains less aggressive [30]. This is evident here by the slightly larger angular error of the type-2 controller versus the type-1 controller.

The extreme case is shown in Figures 4.9 and 4.10. While neither controller behaves particularly well, it is evident that the type-1 controller maintains stability where the type-2 controller fails to do so. The increase in stability margin is not great, and improved results with the type-2 controller could be possible with additional controller tuning, better choices of Type-2 input/output membership functions, and possibly different type reduction and defuzzification algorithms. We attempted to improve the type-2 controller performance by adjusting the shapes of the membership functions, but did not see any major benefits with regard to this test. That said, the work done here should by no means be considered exhaustive in these respects and additional efforts could very well lead to considerably better gains in stability margin.

As expected, the type-2 fuzzy controller did outperform the type-1 with regards to stability in the presence of feedback signal noise. Figure 4.11 shows that there is definitely some benefit to be gained from type-2 in the presence of noise. While the case shown does represent a significant amount of noise with respect to the true values of the system, we believe that the continued trend toward smaller and less expensive spacecraft over the years will eventually require the use of control systems that are able to tolerate very high levels of sensor noise.

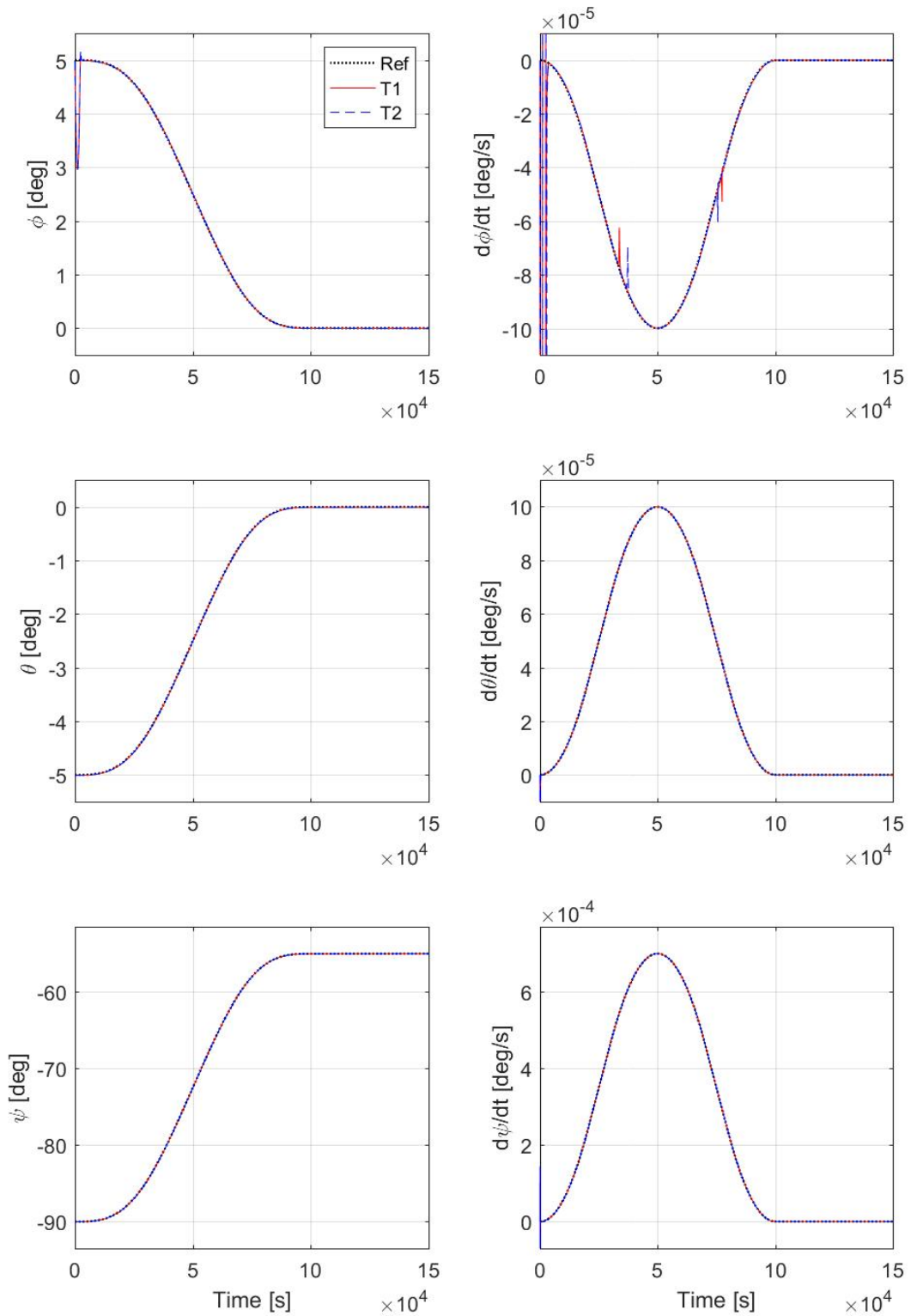


Fig. 4.7: Simulation overlay of type-1 vs type-2 fuzzy controller time response, baseline case. Euler angles (left) and Euler rates (right) in LVLH frame.

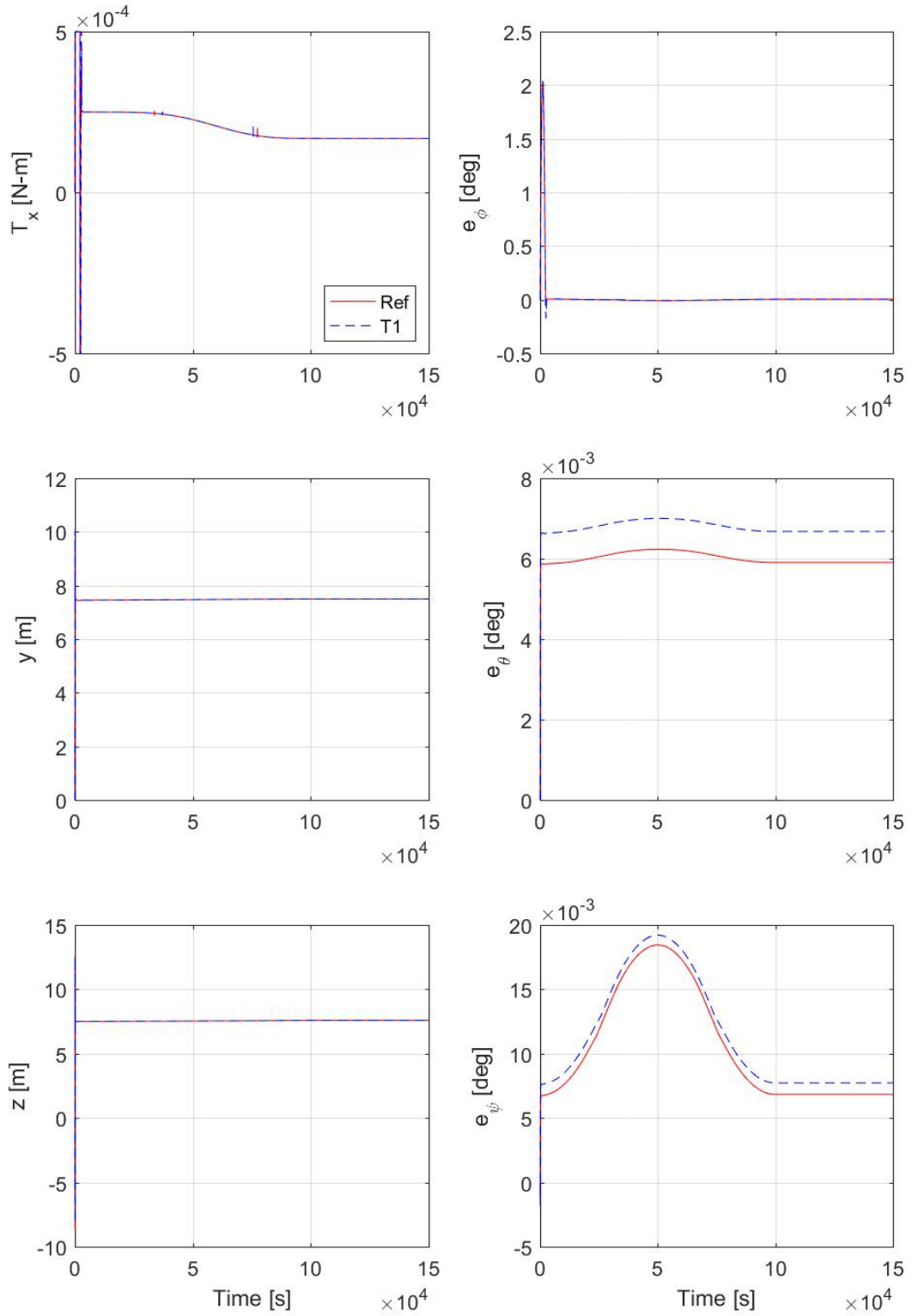


Fig. 4.8: Simulation overlay of type-1 vs type-2 fuzzy controller control action and error, baseline case. Control action (left) and tracking error (right). Control limits are $-5 \times 10^{-4} \leq T_x \leq 5 \times 10^{-4}$, $-28 \leq y \leq 28$, and $-28 \leq z \leq 28$.

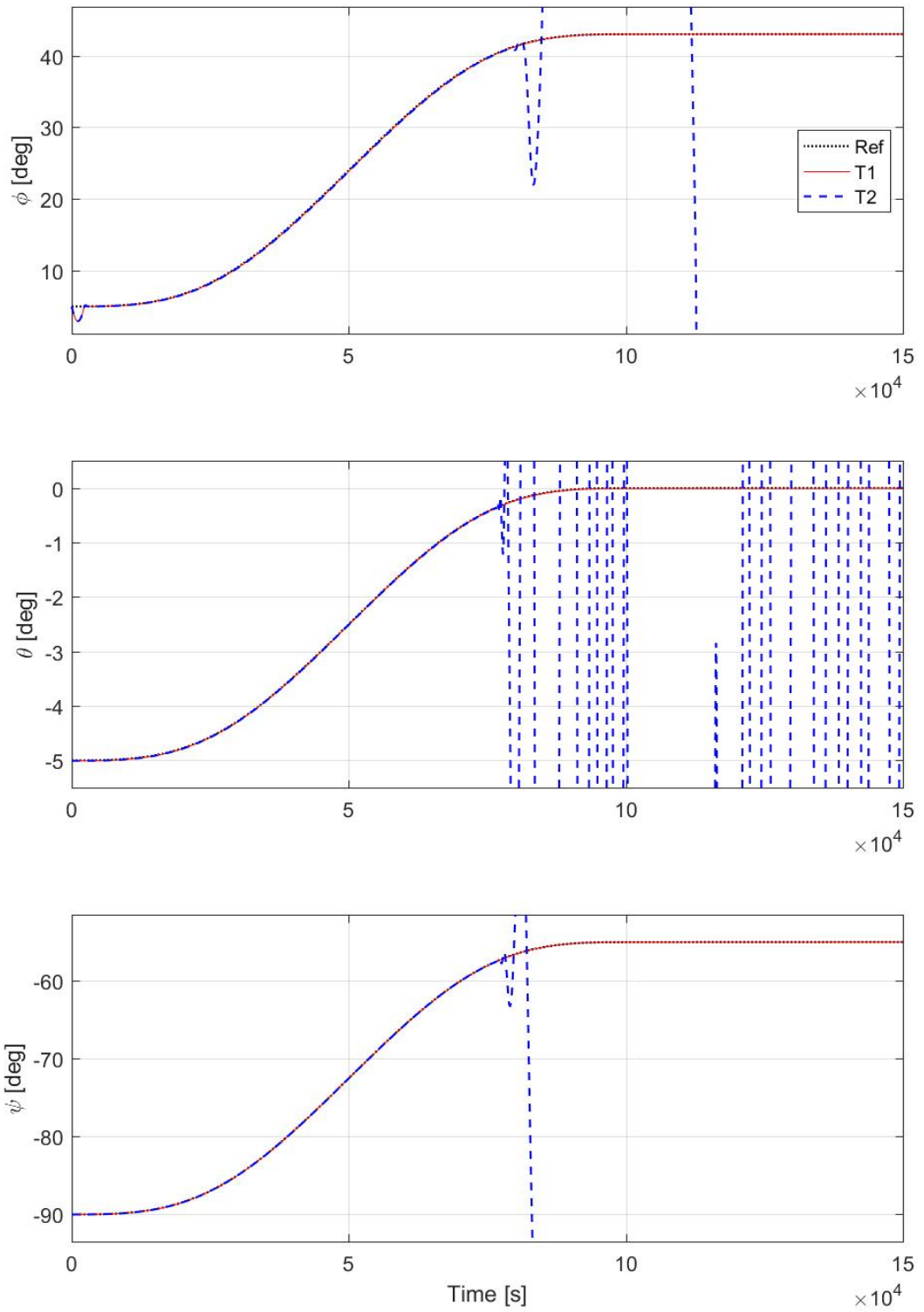


Fig. 4.9: Simulation overlay of type-1 vs type-2 fuzzy controller time response, model mismatch. Euler angles in LVLH frame.

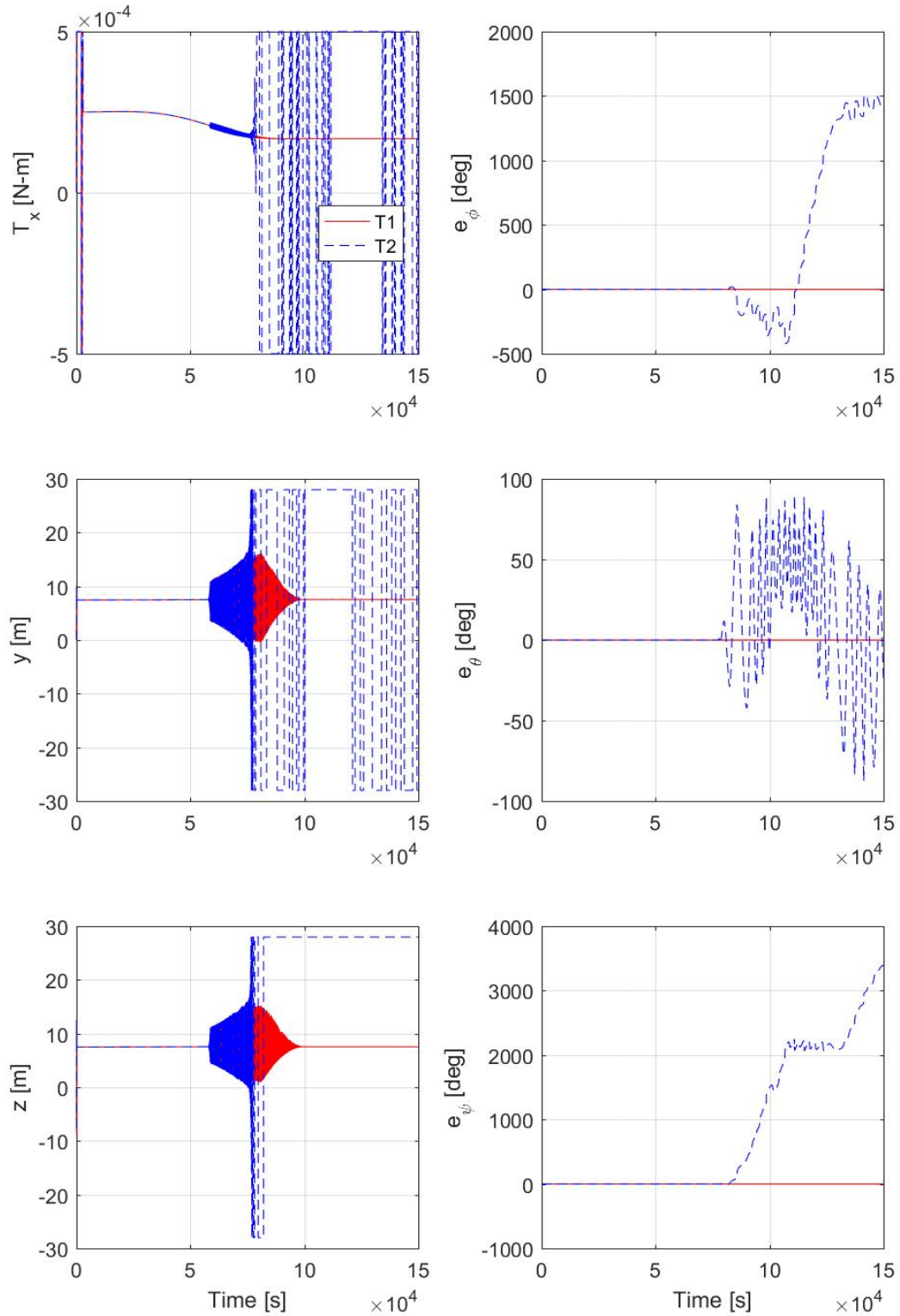


Fig. 4.10: Simulation overlay of type-1 vs type-2 fuzzy controller control action and error, model mismatch case. Control action (left) and tracking error (right). Control limits are $-5 \times 10^{-4} \leq T_x \leq 5 \times 10^{-4}$, $-28 \leq y \leq 28$, and $-28 \leq z \leq 28$.

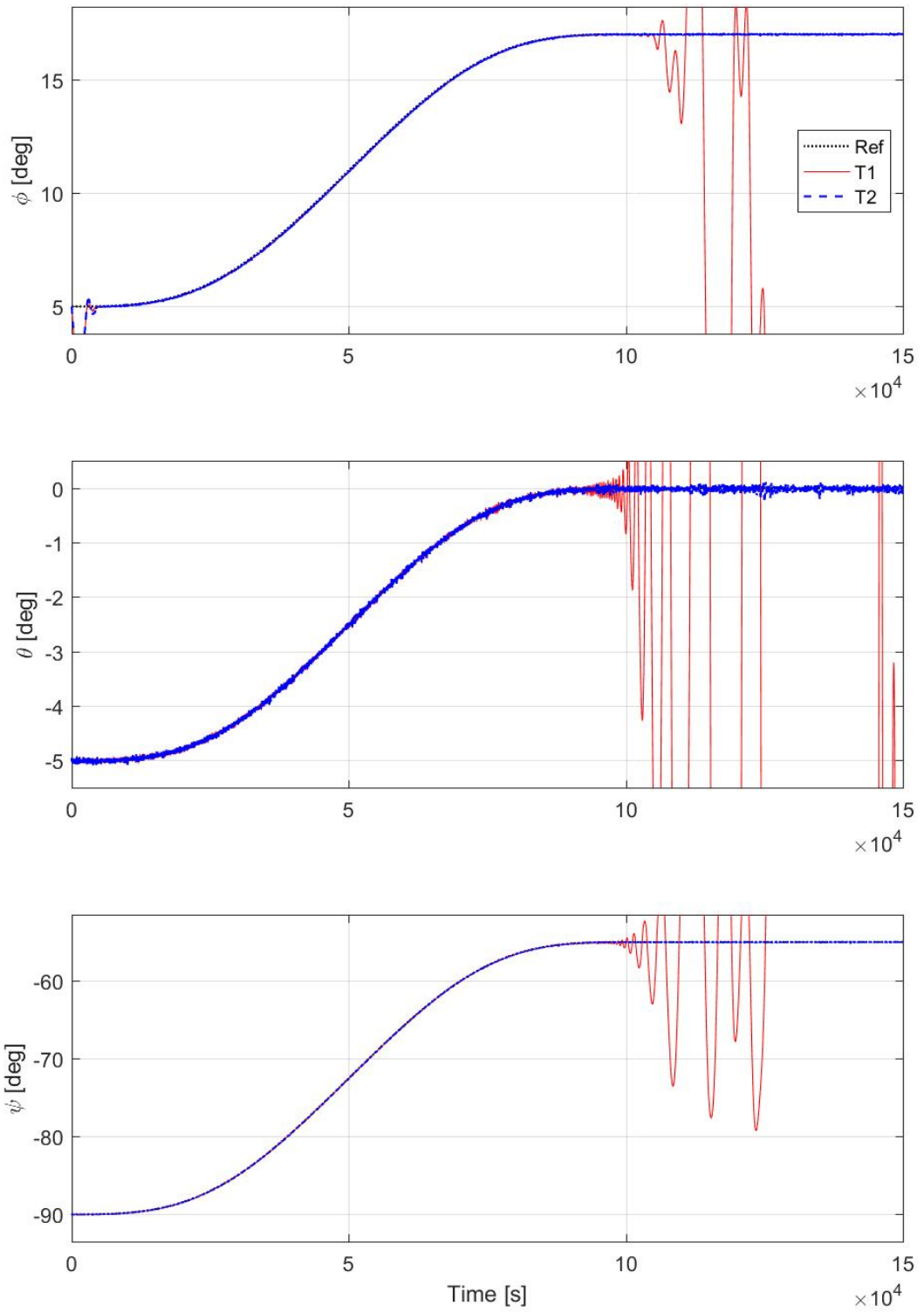


Fig. 4.11: Simulation overlay of type-1 vs type-2 fuzzy controller time response, sensor noise case. Euler angles in LVLH frame.

5. Conclusion

Both PID-ES and fuzzy-ES algorithms show promise as an adaptive control algorithm. However, the algorithm requires a series of experiments that must be run in order to achieve convergence and, in the case of a solar sail spacecraft, the amount of time each experiment takes is simply too great. This algorithm has been used successfully in other applications, but perhaps a spacecraft able to complete the requisite test maneuver much more quickly could make better use of this control technique. Alternatively, we could also consider a different test maneuver to use with the ES algorithm.

In addition, when we compare our results to those in [19] and [20], we note that the rate of convergence is much reduced in the case of the solar sail craft. This leaves the question of whether this is because we are tuning more parameters at once or if it is simply easier to find more aggressive ES gains with a reduced number of parameters.

We also propose a method for improved convergence rate of the fuzzy-ES algorithm by periodically reducing the length of time over which the ITAE cost function is applied. Results show that with absolutely minimal modifications to the ES algorithm and no retuning, the rate of convergence improved over the standard algorithm.

The fuzzy controllers presented in chapter 4 are shown to be completely viable controllers for the solar sail spacecraft. The type-1 fuzzy controller offers improved ability to withstand model mismatch and reduced tracking error, while the type-2 controller

performs better in the presence of sensor noise. Despite these differences, the controllers behave quite similarly.

With regard to performance, the Type-1 fuzzy controller yielded the smallest error when following a prescribed trajectory. When considering robustness in the presence of sensor noise, the Type-2 fuzzy controller performed the best. Lastly, the Type-1 fuzzy controller maintained stability over the widest range of conditions.

The last factor worth considering is the ability of the controller to directly incorporate constraints on control input. Any variation of the fuzzy controller has the ability to directly control the maximum output by adjusting the output membership functions or output scaling factor. The PID-based controllers require some auxiliary output limiting.

In general, the Hybrid ES controllers did not do nearly so well as those tuned offline with PSO. We were unable to find a set of gains that would enable the ES algorithm to converge to a similar level of performance as any of the PSO using the Ziegler-Nichols gains as the starting point.

6. Future Work

Work on either the PID-ES or fuzzy-ES algorithms should focus on systems with much faster response times, unlike the solar sail spacecraft considered here. This will capitalize on the benefits of the ES algorithm despite the need to execute many iterative cycles.

We did not examine the idea of tuning a type-2 fuzzy controller using ES, but there is no reason why this should not be possible. Additionally, instead of tuning gains, the ES algorithm could adjust either type-1 or type-2 membership functions to affect performance. For example, the ES algorithm could tune the size and shape of the FoU to compensate for the current magnitude of sensor noise seen by the controller.

The last direction suggested by this author is to consider the behavior of type-1 and type-2 systems with far fewer membership functions per input. It is possible that the benefits of type-2 versus type-1 fuzzy controllers can be more easily observed when the number of membership function are very small.

Bibliography

- [1] Urbanczyk, M., y1965. “Solar sails - a realistic propulsion for spacecraft”. *Astronautyka*. 3
- [2] Tsander, F. A., 1964. “Problems of flight by jet propulsion: Interplanetary flights”. *Israel Program for Scientific Translations*. 3
- [3] Mark Whorton, A. H., and Pinson, R., 2008. “Nanosail-d: The frist flight demonstration of solar sails for nanosatellites”. *22nd Annual AIAA/USU Conference on Small Satellites*. 3
- [4] Breakthrough Starshot. <https://breakthroughinitiatives.org/Initiative/3>. Accessed: June 11, 2017. 4
- [5] Advanced Exploration Systems: NEA Scout. <https://www.nasa.gov/content/nea-scout>. Accessed: June 11, 2017. 4
- [6] Wie, B., 2015. *Space Vehicle Guidance, Control, and Astrodynamics*. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia. 4, 8, 11
- [7] Wie, B., 2004. “Solar sail attitude control and dynamics, part 1”. *Journal of Guidance, Control, and Dynamics*. 4
- [8] Wie, B., 2004. “Solar sail attitude control and dynamics, part 2”. *Journal of Guidance, Control, and Dynamics*. 4

- [9] Fu, B., and Eke, F. O., 2015. “Attitude control methodology for large solar sails”. *Journal of Guidance, Control, and Dynamics*. 4
- [10] Eldad, O., and Lightsey, E. G., 2015. “Propellantless attitude control of a nonplanar solar sail”. *Journal of Guidance, Control, and Dynamics*. 5
- [11] Choi, M., and Damaren, C. J., 2015. “Structural dynamics and attitude control of a solar sail using tip vanes”. *Journal of Spacecraft and Rockets*. 5
- [12] Baculi, J., and Ayoubiy, M. A., 2016. “Fuzzy-logic supervisory pid attitude control of solar-sail”. *26th AAS/AIAA Space Flight Mechanics Meeting*. 5, 6, 7, 22, 23, 24
- [13] Mazmanyany, L., and Ayoubi, M. A., 2014. “Takagi-sugeno fuzzy model-based attitude control of spacecraft with partially-filled fuel tank”. *AIAA/AAS Astrodynamics Specialist Conference*. 5
- [14] Sendi, C., and Ayoubi, M. A., 2015. “Robust-optimal fuzzy model-based control of flexible spacecraft with actuator amplitude and rate constraints”. *ASME Dynamic Systems and Control Conference*. 5
- [15] Sendi, C., and Ayoubi, M. A., 2014. “Robust fuzzy logic-based tracking control of a flexible spacecraft with h-inf performance criteria”. *AIAA Space Conference and Exposition*. 5
- [16] Hagra, H., 2007. “Type-2 flcs: A new generation of fuzzy controllers”. *IEEE Computational Intelligence Magazine*. 6, 46
- [17] Guay, M., and Burns, D., 2014. “A comparison of extremum seeking algorithms applied to vapor compression system optimization”. *American Control Conference (ACC)*. 16

- [18] Ariyur, K. B., and Krstić, M., 2003. *Real-Time Optimization by Extremum-Seeking Control*. John Wiley and Sons, Inc. 16
- [19] Killingsworth, N. J., and Krstić, M., 2005. “Auto-tuning of pid controllers via extremum seeking”. *Proceedings of the American Control Conference*. 18, 19, 56
- [20] Killingsworth, N. J., and Krstić, M., 2006. “Pid tuning using extremum seeking”. *IEEE Control Systems Magazine*. 18, 19, 56
- [21] Jantzen, J., 2013. *Foundations of Fuzzy Control: A Practical Approach*. Wiley. 20
- [22] Kennedy, J., and Eberhart, R., 1995. “Particle swarm optimization”. *IEEE International Conference on Neural Networks*. 20
- [23] Azad Ghaffari, M. K., and Nešić, D. “Multivariable newtonâbased extremum seeking”. *2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*. 24
- [24] Garrido, A., 2012. “A brief history of fuzzy logic”. *Broad Research in Artificial Intelligence and Neuroscience*. 42
- [25] Mamdani, E. H., and Assilian, S., 1975. “An experiment in linguistic synthesis with a fuzzy logic controller”. *International Journal of Man-Machine Studies*. 43
- [26] Sugeno, M., 1985. *Industrial Applications of Fuzzy Control*. North-Holland. 44
- [27] Mendel, J. M., 2013. “On km algorithms for solving type-2 fuzzy set problems”. *IEEE Transactions on Fuzzy Systems*. 47
- [28] Wu, D., 2013. “Approaches for reducing the computational cost of interval type-2 fuzzy logic systems: Overview and comparisons”. *IEEE Transactions on Fuzzy Systems*. 47

- [29] , 1998. *Fuzzy Control*. Addison-Wesley Longman, Inc. 48
- [30] Wu, D., 2012. “On the fundamental differences between interval type-2 and type-1 fuzzy logic controllers”. *IEEE Transactions on Fuzzy Systems*. 50

Appendix: Computer Code

A PID-ES Controller

runSolarSailES.m

```
%Close any open waitbars that got left over from previous runs
set(groot, 'ShowHiddenHandles', 'on')
delete(get(groot, 'Children'))

%% Solar sail
% This code runs a 3DOF model of a solar sail put together by Joshua Baculi
% at Santa Clara University under the direction of Dr. Mohammad Ayoubi.

% 3 PID controllers, one for each DOF, will be optimized in this code.

% Z-N Gains used to initiate Joshua's PSO code
ZN = [0.0000006000000000, ... % Kp_Tx
      0.0003456000000000, ... % Kd_Tx
      0.00000000260417, ... % Ki_Tx
      0.0030000000000000, ... % Kp_z
      7.1280000000000000, ... % Kd_z
      0.000000315656566, ... % Ki_z
      0.0030000000000000, ... % Kp_y
      7.7760000000000000, ... % Kd_y
      0.000000289351852]*1e4; % Ki_y

Kp_phi = ZN(1);
Ki_phi = ZN(3);
Kd_phi = ZN(2);

Kp_theta = ZN(4);
Ki_theta = ZN(6);
Kd_theta = ZN(5);

Kp_psi = ZN(7);
Ki_psi = ZN(9);
```

```

Kd_psi = ZN(8);

% Specify initial conditions of Solar Sail craft
x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
x0 = x0*pi/180; % convert degrees to radians

% Specify desired final state of Solar Sail craft
xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
xf = xf*pi/180; % convert degrees to radians

% x=zeros(1,6);
% u=zeros(1,3);

% z_max = 28; % mast length [m]
% y_max = 28; % mast length [m]

% Fs = .01; % max thrust [N]

% T_max=(0.5/20)*13.3*Fs*sin(Theta_max); % From "EOM" in "Solar Sail Dynamics" subsystem

%{
=====
% 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-synchronous 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)
%=====
%}

% See simulink model for the rest of the model parameters.

%% ES parameters

Theta0 = [Kp_phi, Ki_phi, Kd_phi;... % Kp_phi, Ki_phi, Kd_phi
          Kp_theta, Ki_theta, Kd_theta;... % Kp_theta, Ki_theta, Kd_theta
          Kp_psi, Ki_psi, Kd_psi]; % Kp_psi, Ki_psi, Kd_psi

% gamma = 100*ones(size(Theta0));
gamma = [.2, 1, 1;...
        .2, 1, 1;...
        1, 1, 1]*1e-4;

%   gamma = ones(2,3)*2;

h = 0.5;

w = pi* [.1, .5125, .9250;...
        .2375, .650, 1.0625;
        .3750, .7875, 1.2];

T0 = 0:10:30e3; % Set duration of simulation
T = T0; % Length of simulation will be adjusted during ES to improve results

N = 500;

rstLenT = 25; % How often to reset length of simulation

%% ES Algorithm

```

```

%Global Parametr
ESendCondition = 1; % This means everything went right

% ES Algorithm Parametr
% a = 0.8; % Must be rational to gaurantee Nyquist criterion satisfied
% w = pi*[a^1, a^2, a^3]; % Perturbation frequency
% gamma = [1,1,1]; % estimation gain (dummy values)
% alpha = [1,1,1]; % perturbation size (dummy values)
% h = 0.5; % High Pass Filter (z-1)/(z+h)0<h<1

Theta = zeros(N, size(Theta0,1), size(Theta0,2)); % Initialize Theta
Theta(1, :, :) = Theta0;

ThetaHat = 0*Theta;
Xi = zeros(N,1);
J = zeros(N,1);

Y = zeros(N, length(T), 6); % Initialize output

%Initialize wait bar with cancel button for the impatient
runES = true;
waitBarMsg = {'Running ES Algorithm', ''};
hWaitBar = waitbar(0, waitBarMsg{1}, 'CreateCancelBtn', ...
    'runES = false;');

lenT = length(T);
iLast = 0;

% alpha = Theta0/15;

% Run the simulation
for i=1:N
    % Ignore the first run of the model, it will need to compile and it
    % will take forever, throwing off the remaining time approximation.
    if i==2
        tStart = tic;
    end
    iLast = i;
    % Check for Cancel button press
    if ~runES
        break
    end

    r_phi = xf(1);
    Kp_phi = Theta(i,1,1);
    Ki_phi = Theta(i,1,2);
    Kd_phi = Theta(i,1,3);

    r_theta = xf(2);
    Kp_theta = Theta(i,2,1);
    Ki_theta = Theta(i,2,2);
    Kd_theta = Theta(i,2,3);

```

```

r_psi = xf(3);
Kp_psi = Theta(i,3,1);
Ki_psi = Theta(i,3,2);
Kd_psi = Theta(i,3,3);

try
    sim('simSolarSail-noActuatorDynamics.slx');
catch ME
    %     disp(p)
    %     disp(ME.message);
    fprintf('Error Running Simulink Model\n')
    fprintf(ME.message)
    ESendCondition = -99;
    break
end

%     for k=1:size(Y,3)
%
%         Y(i,1:lenT,2) = SimOutput.Data(:,2);
%
%     end
Y(i, :, :) = SimOutput.Data(:, :, :);

if sum(abs(Y(i, lenT, :))) > 1e3
    fprintf('Simulation Output Divergence\n')
    ESendCondition = -1;
    break;
end

stateCostFunctionWeight = [1.5, 1.5, 1, .01, .01, .01]; % weights given to different states
                in the cost function

for k = 1:size(Y,3)
    J(i) = J(i) + 1/T(end) * trapz(T(1:lenT), T(1:lenT) .* abs( (Y(i, 1:lenT, k) * 0 + xf(k)) - Y(i, 1:
        lenT, k) ) ) * stateCostFunctionWeight(k);
end

%     J(i) = 2/T(end) * trapz(T, T .* abs(U(1, :)-Y(i, 1:lenT, 2))) + 10/T(end) * trapz(T, T .* abs(Y(i, 1:
lenT, 1)));

%     alpha = squeeze(exp(-1*i/N)*Theta(i, :, :)/10); % taper alpha slowly
alpha = squeeze(Theta(i, :, :)/10);
%     alpha = exp(-.0005*i)*Theta0/15;

switch i
    case 1
        % Zero gradient initial condition
        Xi(i) = J(i);
        ThetaHat(i+1, :, :) = Theta(i, :, :);
        Theta(i+1, :, :) = Theta(i, :, :);
    case N
        Xi(i) = -h*Xi(i-1) + J(i-1);

```

```

        break;
    otherwise
        Xi(i) = -h*Xi(i-1) + J(i-1);
        ThetaHat(i+1, :, :) = squeeze(ThetaHat(i, :, :)) - gamma.*alpha.*cos(w*i)*(J(i) - (1+h)*
            Xi(i));
        Theta(i+1, :, :) = squeeze(ThetaHat(i+1, :, :)) + alpha.*cos(w*(i+1));
    end
end
% Check if next Theta isn't horribly wrong
if (sum(sum(Theta(i+1, :, :))>1e9))>0 || (sum(sum(Theta(i+1, :, :))<0))>0
    endCondition = -98;
    fprintf('ES Algorithm Failed!\n');
    break
end

if mod(i,5)==0 && i>1
    waitBarMsg{2} = sprintf(' [%.0fs Remaining]', toc(tStart)/i*(N-i));
    waitbar(i/N, hWaitBar, [waitBarMsg{1}, waitBarMsg{2}]);
end

%Reset simulation length
if mod(i, rstLenT)==0
    [~, minCostIdx] = (min(J(i-rstLenT+1:i)));
    minCostIdx = minCostIdx + i - rstLenT + 1;
    %
    minCostIdx = i;
    stabBandPct = 10; % [percent] plus/minus tolerance for stability
    %Find the last time that the response exceeds the tolerance band
    %and set that as the new length of the simulation
    stabTimeLo = 0;
    stabTimeHi = 0;
    for j=1:3
        stabBandLo = xf(j) - abs(-xf(j)+x0(j))*stabBandPct/100;
        stabBandHi = xf(j) + abs(-xf(j)+x0(j))*stabBandPct/100;

        tempHi = T(find(Y(minCostIdx, :, j)<stabBandLo, 1, 'last'));
        if tempHi > stabTimeLo
            stabTimeLo = tempHi;
        end
        temp2 = T(find(Y(minCostIdx, :, j)>stabBandHi, 1, 'last'));
        if temp2 > stabTimeHi
            stabTimeHi = temp2;
        end
        fprintf(' j=%0d, stabTimeLo = %2d, stabTimeHi = %2d\n', j, tempHi, temp2);
    end
    end
    stabTime = round(min([max([stabTimeHi, stabTimeLo]), T(lenT)]), 0);
    lenT = find(T>=stabTime, 1, 'first');
    %
    T = 0:T0(2)-T0(1):stabTime;
    %
    lenT = length(T);
    %
    U = U(1:lenT);
    fprintf('Resetting simulation length. New length: %0d sec.\n', T(lenT))
end

end

if iLast~=N

```

```

        iLast = iLast+1;
    end

    try
        delete(hWaitBar);
    catch

    end

    tElapsed = toc(tStart);
    fprintf('This calculation took %0.2f sec \n',tElapsed)

%% Plot Results
close all

if ESendCondition == 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot N=1 response to step input

% figure()
% subplot(2,1,1)
% plot(SimOut.Time,SimOut.Data(:,1));
% ylabel('\Theta')
% title('System Response')
%
% subplot(2,1,2)
% plot(SimOut.Time,SimOut.Data(:,2));
% ylabel('x')
% xlabel('Time [s]')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% figure()
% subplot(2,1,1)
% plot(error_theta);
% ylabel('e_\Theta')
% title('Calculated Errors')
%
% subplot(2,1,2)
% plot(error_x);
% ylabel('e_x')
% xlabel('Time [s]')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% figure()
% plot(controlAction);
% ylabel('u')
% title('Control Action')
% xlabel('Time [s]')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot cost function evolution

if N>1
    figure()

```



```

semilogy(1:N,J)
xlabel('Iteration')
ylabel('Cost Function')
grid on
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot gain evolution
if N>1
figure()
for i=1:size(Theta0,1)
for j=1:size(Theta0,2)
subplot(size(Theta0,2),size(Theta0,1),(i-1)*size(Theta0,2)+j)
plot(1:iLast,Theta(1:iLast,i,j))
end
end

subplot(3,3,1)
title('\phi')
ylabel('K_p')
subplot(3,3,2)
title('\theta')
subplot(3,3,3)
title('\psi')
subplot(3,3,4)
ylabel('K_i')
subplot(3,3,7)
ylabel('K_d')
% xlabel('Iteration')
subplot(3,3,8)
xlabel('Iteration')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot time traces
figure()
% ax1 = axes();
rrows = 2;
ccols = 3;
% ax1 = subplot(2,3,1); % plot phi
if N-rstLenT+1 > 0
[~,indBest] = min(J(N-rstLenT+1:N));
indBest = indBest + N-rstLenT;
else
[~,indBest] = min(J(1:N));
end
if N>5
plots = round([1,N/10,N/4,N/2,indBest]);
else
plots=1:N;
end
names = {'\phi','\theta','\psi','d\phi/dt','d\theta/dt','d\psi/dt'};
% plots = round([1,N/3,indBest]);

```

```

lineNames = cell(length(plots),1);

stabTimeLo = 0;
stabTimeHi = 0;
for i = 1:length(plots)
    for j = 1:rrows;
        for k = 1:ccols;
            output = (j-1)*ccols+k;
            subplot(rrows, ccols, output)
            switch i
                case 1
                    plot(T0,180/pi*Y(plots(i), :, output), '-r')
                    hold on
                    grid on
                    % if N==1
                    % break;
                    % end
                case length(plots)
                    plot(T0,180/pi*Y(plots(i), :, output), '-k', 'LineWidth', 2)
                    plot([T0(1), T0(end)], [1, 1]*xf(output)*180/pi, '--r')

                    stabBandLo = xf(output) - abs(-xf(output)+x0(output))*stabBandPct/100;
                    stabBandHi = xf(output) + abs(-xf(output)+x0(output))*stabBandPct/100;

                    tempLo = T(find(Y(minCostIdx, 1:lenT, output)<stabBandLo, 1, 'last'));
                    if tempLo > stabTimeLo
                        stabTimeLo = tempLo;
                    end
                    tempHi = T(find(Y(minCostIdx, 1:lenT, output)>stabBandHi, 1, 'last'));
                    if tempHi > stabTimeHi
                        stabTimeHi = tempHi;
                    end
                    plot([T0(1), T0(end)], [1, 1]*stabBandLo*180/pi, ':k')
                    plot([T0(1), T0(end)], [1, 1]*stabBandHi*180/pi, ':k')
                    % fprintf(' output=%0d, stabTimeLo = %0d, stabTimeHi = %0d\n', output,
                    % temp1, temp2);

                    stabTime = round(max([stabTimeHi, stabTimeLo]), 2);
                otherwise
                    % plot(T, Yactual(plots(i), :, output))
                    plot(T0,180/pi*Y(plots(i), :, output), '-b')
            end
            ylabel(names(output));
        end
    end
end
% lineNames{2*i-1} = sprintf('N=%0.f actual', plots(i));
lineNames{i} = sprintf('N=%0.f', plots(i));
end

% xlabel(ax1, 'Time (s)')
if N>1
    legend(lineNames, 'location', 'SouthEast');
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot select response functions
%
%
%   plots = [10,50];
%   ind_best = find(min(J));
%
%   plots_legend = cell(1,2+length(plots));
%
%
%   figure()
%   ax1 = subplot(2,1,1);
%   plot(T',squeeze(Y(i, :, 1)), 'b', [0, T(end)], [0, 0], 'r')
%   hold on
%   ylabel('\Theta')
%   title('Select Response Plots')
%   ax2 = subplot(2,1,2);
%   plot(T',squeeze(Y(i, :, 2)), 'b', [0, T(end)], [U(1), U(end)], 'r')
%   hold on
%   ylabel('x')
%   xlabel('Time [s]')
%
%   plots_legend{1} = sprintf('N = %3.0f', 1);
%
%   for i=1:length(plots)
%       plot(ax1, T', Y(plots(i), :, 1));
%       plot(ax2, T', Y(plots(i), :, 2));
%
%       plots_legend{i+1} = sprintf('N = %3.0f', plots(i));
%   end
%
%   plot(ax1, T', squeeze(Y(plots(ind_best), :, 1)), 'LineWidth', 2);
%   plot(ax2, T', squeeze(Y(plots(ind_best), :, 2)), 'LineWidth', 2);
%   plots_legend{2+length(plots)} = sprintf('Best');
%   legend(plots_legend, 'Location', 'SouthEast')
end

% catch ME
%   delete(hWaitBar)
%   throw(ME)
% end

% Delete All Open Figures
% Use this to get rid of progress bars that won't close

%{
set(groot, 'ShowHiddenHandles', 'on')
delete(get(groot, 'Children'))
%}

```

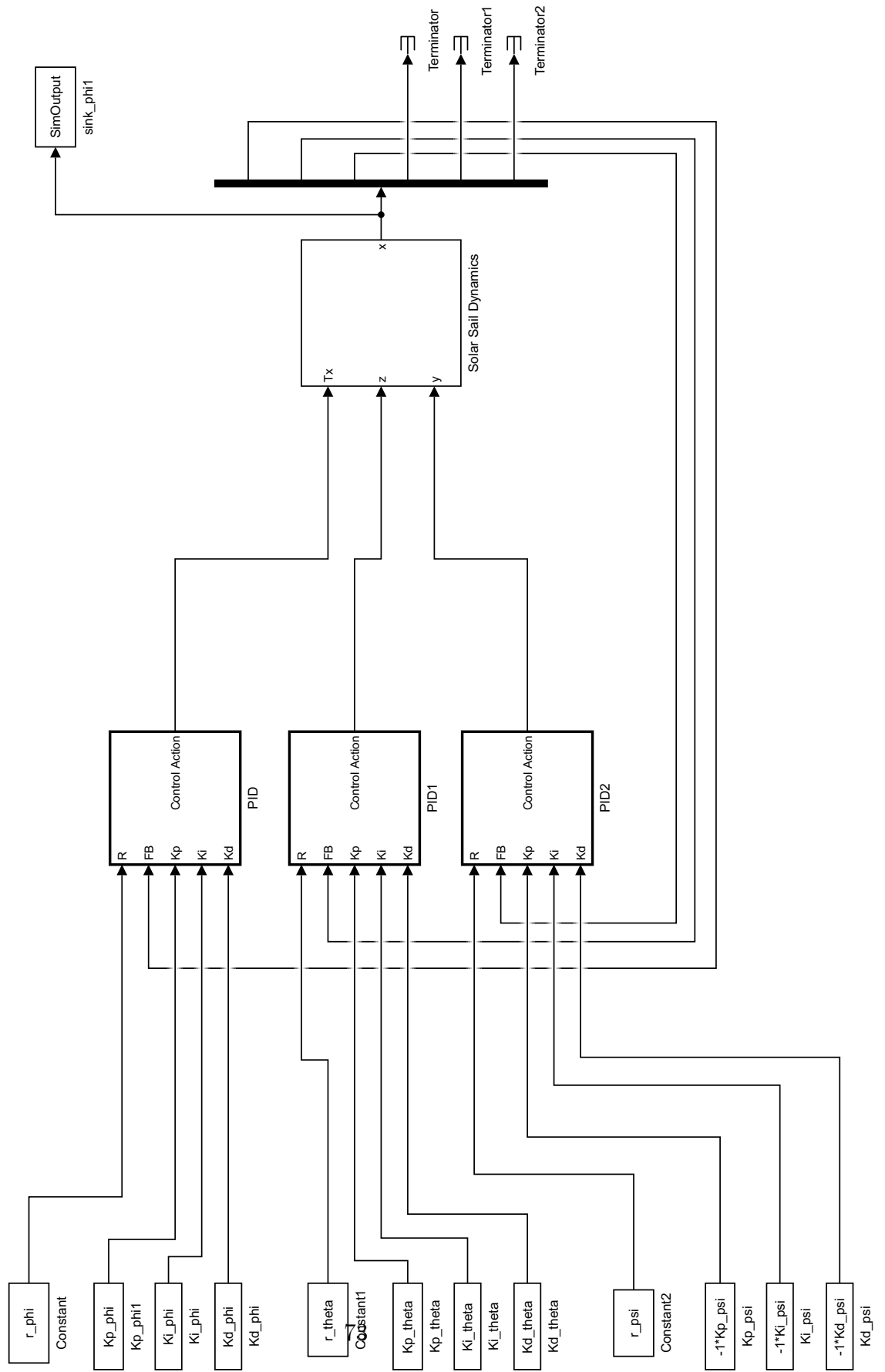


Fig. 6.1: Simulink model, PID ES controller, base.

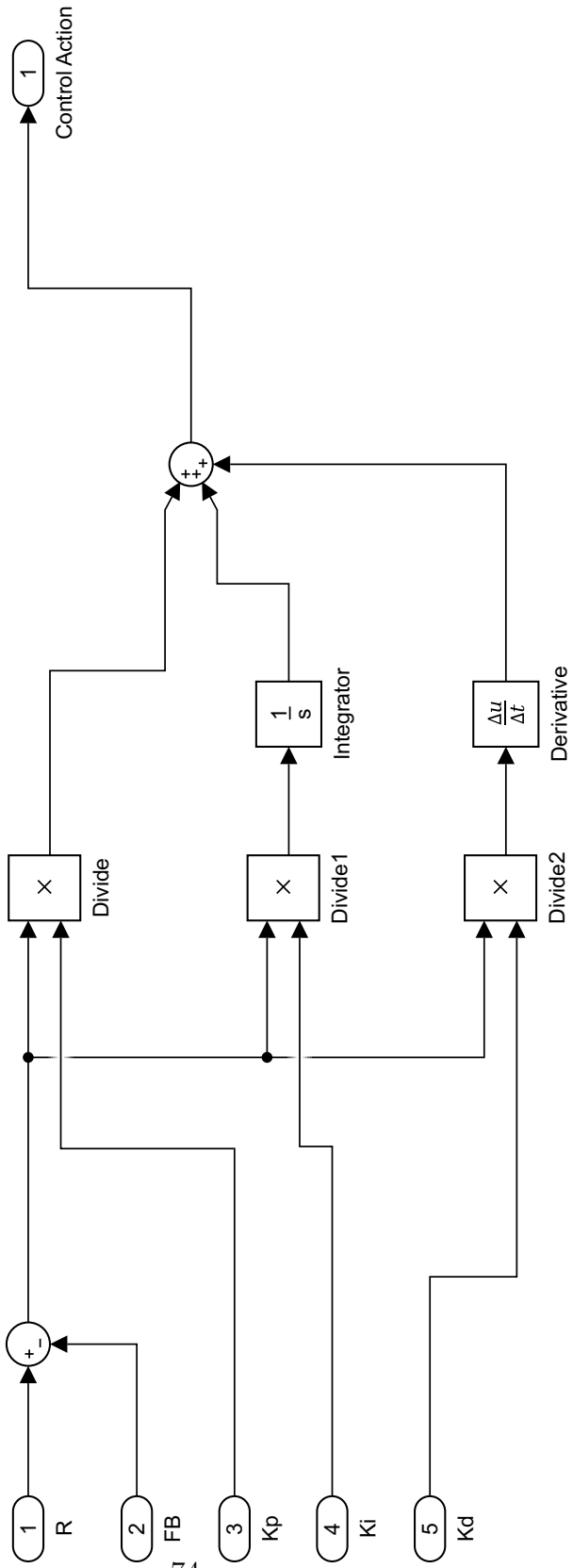


Fig. 6.2: Simulink model, PID ES controller, *PID Controller*.

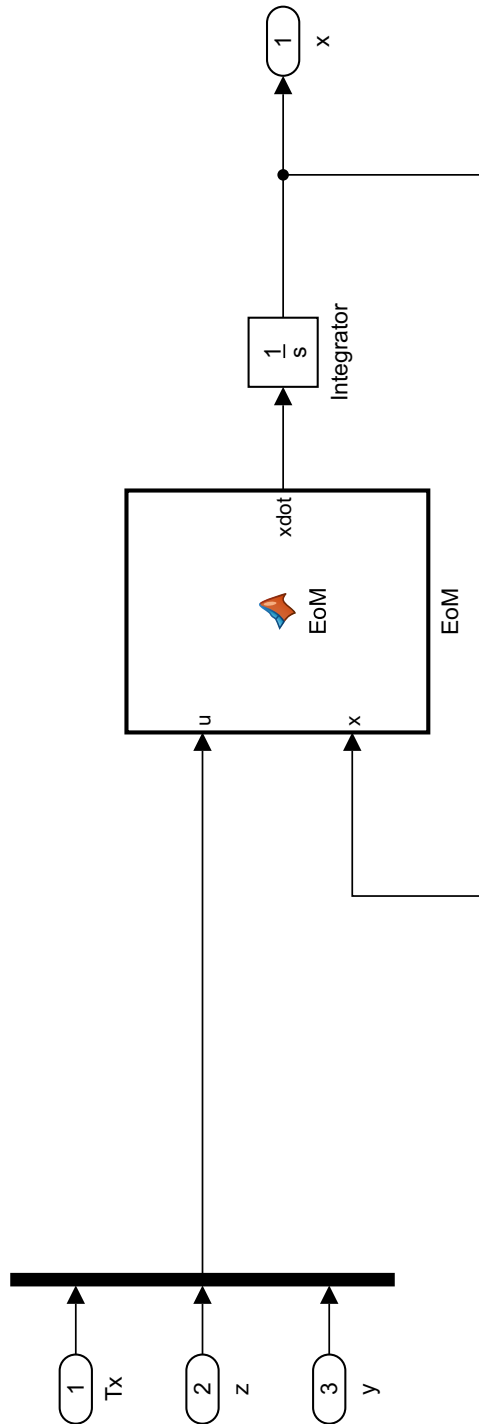


Fig. 6.3: Simulink model, Fuzzy ES controller, *Solar Sail Dynamics*.

EoM.m

```
function xdot = EoM(u,x)
%=====
% 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-synchronous 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
%=====
```

```

% Defining states and inputs
Tx = u(1);
z = u(2);
y = u(3);
Ty = 0;
Tz = 0;

% Calculating the J's
Jx = Ix+mr*(y(1)^2+z(1)^2);
Jy = Iy+mr*z(1)^2;
Jz = Iz+mr*y(1)^2;

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

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]';
end

```


B Fuzzy-ES Controller

runFuzzyES.m

```
%Close any open waitbars that got left over from previous runs
set(groot, 'ShowHiddenHandles', 'on')
delete(get(groot, 'Children'))

%% Solar sail
% This code runs a 3DOF model of a solar sail put together by Joshua Baculi
% at Santa Clara University under the direction of Dr. Mohammad Ayoubi.

% 3 PID controllers, one for each DOF, will be optimized in this code.

% Z-N Gains used to initiate Joshua's PSO code
ZN = [0.000000600000000, ... % Kp-Tx
      0.000345600000000, ... % Kd-Tx
      0.000000000260417, ... % Ki-Tx
      0.003000000000000, ... % Kp-z
      7.128000000000000, ... % Kd-z
      0.000000315656566, ... % Ki-z
      0.003000000000000, ... % Kp-y
      7.776000000000000, ... % Kd-y
      0.000000289351852]*1e4; % Ki-y

Ap = pi;
Ad = pi;
Ai = 1;

Kp_phi = ZN(1)*Ap;
Ki_phi = ZN(3)*Ai;
Kd_phi = ZN(2)*Ad;

%% Improved gains N=1000
% Kp_phi = 0.010559166559334;
% Ki_phi = 0.000004133928122;
% Kd_phi = 4.538597533303346;

% Improved gains N=1000 x 2
% Kp_phi = 0.018539605011348;
% Ki_phi = 0.000004366340152;
% Kd_phi = 9.331967227320478;

Kp_theta = ZN(4)*Ap;
Ki_theta = ZN(6)*Ai;
Kd_theta = ZN(5)*Ad;

Kp_psi = ZN(7)*Ap;
Ki_psi = ZN(9)*Ai;
Kd_psi = ZN(8)*Ad;
```

```

% Specify initial conditions of Solar Sail craft
x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
x0 = x0*pi/180; % convert degrees to radians

% Specify desired final state of Solar Sail craft
xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
xf = xf*pi/180; % convert degrees to radians

%
x=zeros(1,6);
%
u=zeros(1,3);

%
z_max = 28; % mast length [m]
%
y_max = 28; % mast length [m]

%
Fs = .01; % max thrust [N]

%
T_max=(0.5/20)*13.3*Fs*sin(Theta_max); % From "EOM" in "Solar Sail Dynamics" subsystem

%% ES parameters

%
Theta0 = [Kp_phi, Ki_phi, Kd_phi;... % Kp_phi, Ki_phi, Kd_phi
          Kp_theta, Ki_theta, Kd_theta;... % Kp_theta, Ki_theta, Kd_theta
          Kp_psi, Ki_psi, Kd_psi]; % Kp_psi, Ki_psi, Kd_psi

Theta0 = [Kp_phi, Ki_phi, Kd_phi];... % Kp_phi, Ki_phi, Kd_phi
%
          Kp_theta, Ki_theta, Kd_theta;... % Kp_theta, Ki_theta, Kd_theta
%
          Kp_psi, Ki_psi, Kd_psi]; % Kp_psi, Ki_psi, Kd_psi

% gamma = 100*ones(size(Theta0));
%
gamma = [.2, 1, 1;...
        .2, 1, 1;...
        1, 1, 1]*1e-4;

gamma = [60, 30, 5]*1e-3;
%
gamma = [60, 30, 5]*1e-3; % BEST (lucky)

%
gamma = ones(2,3)*2;

h = 0.5;

%
w = pi* [.1, .5125, .9250;...
        .2375, .650, 1.0625;
        .3750, .7875, 1.2];

w = pi* [.2, .5, .8];

T0 = 0:10:30e3; % Set duration of simulation
T = T0; % Length of simulation will be adjusted during ES to improve results

```

```

N = 25;

rstLenT = 5000; % How often to reset length of simulation
rstImmediately = true;

%% ES Algorithm

ringBufferLen = 20;
ringBuffer = zeros(1,ringBufferLen);

%Global Parametrs
ESendCondition = 1; % This means everything went right

% ES Algorithm Parametrs
% a = 0.8; % Must be rational to gaurantee Nyquist criterion satisfied
% w = pi*[a^1, a^2, a^3]; % Perturbation frequency
% gamma = [1,1,1]; % estimation gain (dummy values)
% alpha = [1,1,1]; % perturbation size (dummy values)
% h = 0.5; % High Pass Filter (z-1)/(z+h)0<h<1

Theta = zeros(N,size(Theta0,1),size(Theta0,2)); % Initialize Theta
Theta(1, :, :) = Theta0;

ThetaHat = 0*Theta;
Xi = zeros(N,1);
J = zeros(N,1);

Y = zeros(N,length(T),6); % Initialize output

%Initialize wait bar with cancel button for the impatient
runES = true;
waitBarMsg = {'Running ES Algorithm', ''};
hWaitBar = waitbar(0,waitBarMsg{1}, 'CreateCancelBtn', ...
    'runES = false;');

lenT = length(T);
iLast = 0;
alphaDen = 10;
alpha = squeeze(Theta0/alphaDen);

tStartTotal = tic;

% alpha = Theta0/15;

% Run the simulation
for i=1:N
    % Ignore the first run of the model, it will need to compile and it
    % will take forever, throwing off the remaining time approximation.

    tStart = tic;

    iLast = i;

```

```

% Check for Cancel button press
if ~runES
    break
end

r_phi = xf(1);
%     Kp_phi = Theta(i,1,1);
%     Ki_phi = Theta(i,1,2);
%     Kd_phi = Theta(i,1,3);
Kp_phi = Theta(i,1);
Ki_phi = Theta(i,2);
Kd_phi = Theta(i,3);

r_theta = xf(2);
%     Kp_theta = Theta(i,2,1);
%     Ki_theta = Theta(i,2,2);
%     Kd_theta = Theta(i,2,3);

r_psi = xf(3);
%     Kp_psi = Theta(i,3,1);
%     Ki_psi = Theta(i,3,2);
%     Kd_psi = Theta(i,3,3);

%     try
%         sim('simSolarSail_noActuatorDynamics_Fuzzy.slx');
%     catch ME
%         disp(p)
%         disp(ME.message);
%         fprintf('Error Running Simulink Model\n')
%         fprintf(ME.message)
%         ESendCondition = -99;
%         break
%     end

%     for k=1:size(Y,3)
%
%         Y(i,1:lenT,2) = SimOutput.Data(:,2);
%
%     end
Y(i, :, :) = SimOutput.Data(:, :, :);

if sum(abs(Y(i, lenT, :))) > 1e3 > 0
    fprintf('Simulation Output Divergence\n')
    ESendCondition = -1;
    break;
end

stateCostFunctionWeight = [1, 0, 0, 0, 0, 0]; % weights given to different states in the cost
function

for k = 1:size(Y,3)
    J(i) = J(i) + 1/T(end) * trapz(T(1:lenT), T(1:lenT). * abs( (Y(i, 1:lenT, k) * 0 + xf(k)) - Y(i, 1:
        lenT, k) ) ) * stateCostFunctionWeight(k);
end

```

```

%      J(i) = 2/T(end) * trapz(T,T.*abs(U(1,:)-Y(i,1:lenT,2))) + 10/T(end) * trapz(T,T.*abs(Y(i,1:
lenT,1)));

%      alpha = squeeze(exp(-1*i/N)*Theta(i,,:)/10); % taper alpha slowly
%      alpha = squeeze(Theta(i,+)/alphaDen); % Best so far
%      alpha = exp(-.0005*i)*Theta0/15;

switch i
    case 1
        % Zero gradient initial condition
        Xi(i) = J(i);
        ThetaHat(i+1,:) = Theta(i,:);
        Theta(i+1,:) = Theta(i,,:);
    case N
        Xi(i) = -h*Xi(i-1) + J(i-1);
        break;
    otherwise
        Xi(i) = -h*Xi(i-1) + J(i-1);
        ThetaHat(i+1,:) = squeeze(ThetaHat(i,:)) - gamma.*alpha.*cos(w*i)*(J(i) - (1+h)*Xi(i)
        );
        Theta(i+1,:) = squeeze(ThetaHat(i+1,:)) + alpha.*cos(w*(i+1));
end
% Check if next Theta isn't horribly wrong
if(sum(sum(Theta(i+1,,:))>1e9))>0 || (sum(sum(Theta(i+1,,:))<0))>0
    endCondition = -98;
    fprintf('ES Algorithm Failed!\n')
    break
end

ringBuffer(mod(i,ringBufferLen)+1) = toc(tStart);

if mod(i,5)==0 && i>1
    if(i>ringBufferLen)
        waitBarMsg{2} = sprintf(' [%0.0fs Remaining]',sum(ringBuffer)/ringBufferLen*(N-i));
    else
        waitBarMsg{2} = sprintf(' [%0.0fs Remaining]',sum(ringBuffer)/i*(N-i));
    end
    waitbar(i/N,hWaitBar,[waitBarMsg{1},waitBarMsg{2}]);
end

%Reset simulation length
if mod(i,rstLenT)==0 || (rstImmediately && i==1)
    if i==1
        minCostIdx = 1;
    else
        [~,minCostIdx] = (min(J(i-rstLenT+1:i)));
        minCostIdx = minCostIdx + i-rstLenT;
    end
    %
    minCostIdx = i;
    stabBandPct = 1; %[percent] plus/minus tolerance for stability
    %Find the last time that the response exceeds the tolerance band
    %and set that as the new length of the simulation

```

```

stabTimeLo = 0;
stabTimeHi = 0;
for j=1:1
    stabBandLo = xf(j) - abs(-xf(j)+x0(j))*stabBandPct/100;
    stabBandHi = xf(j) + abs(-xf(j)+x0(j))*stabBandPct/100;

    tempLo = T(find(Y(minCostIdx, :, j)<stabBandLo,1,'last'));
    if isempty(tempLo)
        tempLo = T(lenT);
    end
    if tempLo > stabTimeLo
        stabTimeLo = tempLo;
    end
    tempHi = T(find(Y(minCostIdx, :, j)>stabBandHi,1,'last'));
    if isempty(tempHi)
        tempHi = T(lenT);
    end
    if tempHi > stabTimeHi
        stabTimeHi = tempHi;
    end
    fprintf(' j=%0d, stabTimeLo = %0.2d, stabTimeHi = %0.2d\n',j ,tempLo ,tempHi);
end
stabTime = round(min([max([stabTimeHi ,stabTimeLo]) ,T(lenT)]),0);
lenT = find(T>=stabTime,1,'first');
% T = 0:T0(2)-T0(1):stabTime;
% lenT = length(T);
% U = U(1:lenT);
fprintf('Resetting simulation length. New length: %0d sec.\n',T(lenT))
% alphaDen = round(alphaDen*1.05,2);
alpha = squeeze(Theta(minCostIdx,:)/alphaDen);
end

end

if iLast~=N
    iLast = iLast+1;
end

try
    delete(hWaitBar);
catch

end

fprintf('This calculation took %0.2f sec \n',toc(tStartTotal))

%% Plot Results
close all

if ESendCondition == 1

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% Plot N=1 response to step input

% figure()
% subplot(2,1,1)
% plot(SimOut.Time,SimOut.Data(:,1));
% ylabel('\Theta')
% title('System Response')
%
% subplot(2,1,2)
% plot(SimOut.Time,SimOut.Data(:,2));
% ylabel('x')
% xlabel('Time [s]')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% figure()
% subplot(2,1,1)
% plot(error_theta);
% ylabel('e-\Theta')
% title('Calculated Errors')
%
% subplot(2,1,2)
% plot(error_x);
% ylabel('e-x')
% xlabel('Time [s]')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% figure()
% plot(controlAction);
% ylabel('u')
% title('Control Action')
% xlabel('Time [s]')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Plot cost function evolution

if N>1
    figure()
    semilogy(1:N,J)
    xlabel('Iteration')
    ylabel('Cost Function')
    grid on
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Plot gain evolution
if N>1
    figure()
    for i=1:size(Theta0,1)
    %
    %         for j=1:size(Theta0,2)
    %             subplot(size(Theta0,2),size(Theta0,1),(i-1)*size(Theta0,2)+j)
    %                 plot(1:iLast,Theta(1:iLast,i,j))
    %             end
    %         end
    %     end
    for i=1:length(Theta0)

        subplot(3,1,i)

```

```

        plot(1:iLast, Theta(1:iLast, i))

    end

    subplot(3,1,1)
    title('\phi')
    ylabel('K_p')
%     subplot(3,1,2)
%     title('\theta')
%     subplot(3,1,3)
%     title('\psi')
    subplot(3,1,2)
    ylabel('K_i')
    subplot(3,1,3)
    ylabel('K_d')
%     xlabel('Iteration')
%     subplot(3,3,8)
    xlabel('Iteration')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot time traces
iLast = iLast - 1;
figure()
% ax1 = axes();
rows = 2;
ccols = 3;
%     ax1 = subplot(2,3,1); % plot phi
if iLast-rstLenT+1 > 0
    [~, minCostIdx] = min(J(iLast-rstLenT+1:iLast));
    minCostIdx = minCostIdx + iLast-rstLenT;
else
    [~, minCostIdx] = min(J(1:iLast));
end
if iLast>5
%     plots = round([1, iLast/10, iLast/4, iLast/2, minCostIdx]);
    plots = round([1, minCostIdx]);
else
    plots=1:iLast;
end
names = {'\phi', '\theta', '\psi', 'd\phi/dt', 'd\theta/dt', 'd\psi/dt'};
%     plots = round([1, N/3, indBest]);

lineNames = cell(length(plots), 1);

stabTimeLo = 0;
stabTimeHi = 0;
for i = 1:length(plots)
    for j = 1:rows;
        for k = 1:ccols;
            output = (j-1)*ccols+k;
            subplot(rows, ccols, output)
            switch i

```



```

case 1
    plot(T0,180/pi*Y(plots(i),:,output),'-r')
    hold on
    grid on
    if N==1
        break;
    end
case length(plots)
    plot(T0,180/pi*Y(plots(i),:,output),'-k','LineWidth',2)
    plot([T0(1),T0(end)],[1,1]*xf(output)*180/pi,'--r')

    stabBandLo = xf(output) - abs(-xf(output)+x0(output))*stabBandPct/100;
    stabBandHi = xf(output) + abs(-xf(output)+x0(output))*stabBandPct/100;

    tempLo = T(find(Y(minCostIdx,1:lenT,output)<stabBandLo,1,'last'));
    if tempLo > stabTimeLo
        stabTimeLo = tempLo;
    end
    tempHi = T(find(Y(minCostIdx,1:lenT,output)>stabBandHi,1,'last'));
    if tempHi > stabTimeHi
        stabTimeHi = tempHi;
    end
    plot([T0(1),T0(end)],[1,1]*stabBandLo*180/pi,':k')
    plot([T0(1),T0(end)],[1,1]*stabBandHi*180/pi,':k')
    fprintf(' output=%0d, stabTimeLo = %0d, stabTimeHi = %0d\n',output,
temp1,temp2);

    stabTime = round(max([stabTimeHi,stabTimeLo]),2);
otherwise
    plot(T,Yactual(plots(i),:,output))
    plot(T0,180/pi*Y(plots(i),:,output),'-b')
end
ylabel(names(output));
end
end
% lineNames{2*i-1} = sprintf('N=%0.f actual',plots(i));
lineNames{i} = sprintf('N=%0.f',plots(i));
end

% xlabel(ax1,'Time (s)')
if N>1
    legend(lineNames,'location','SouthEast');
end
end
end

```

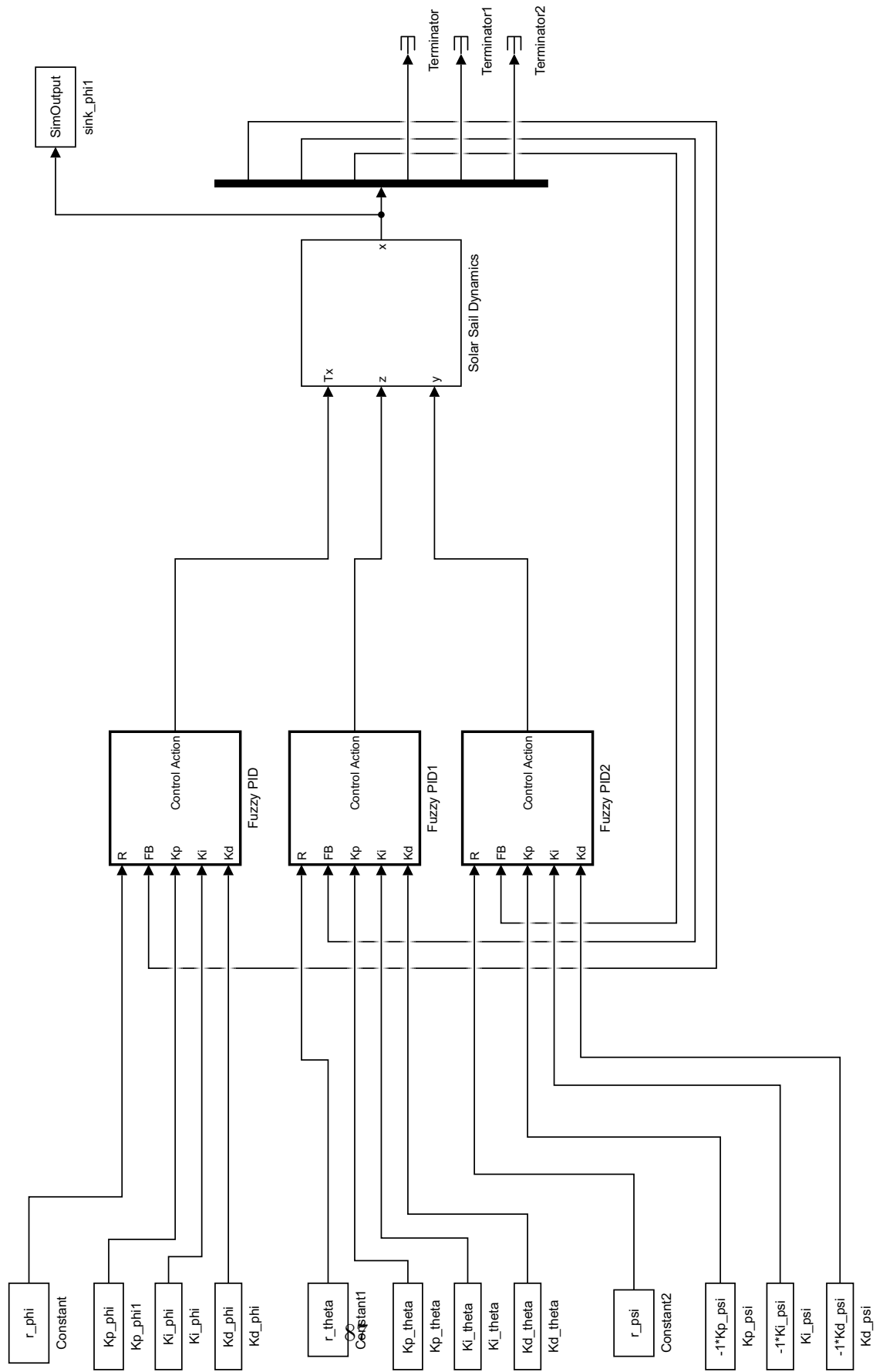


Fig. 6.4: Simulink model, Fuzzy ES controller, base.

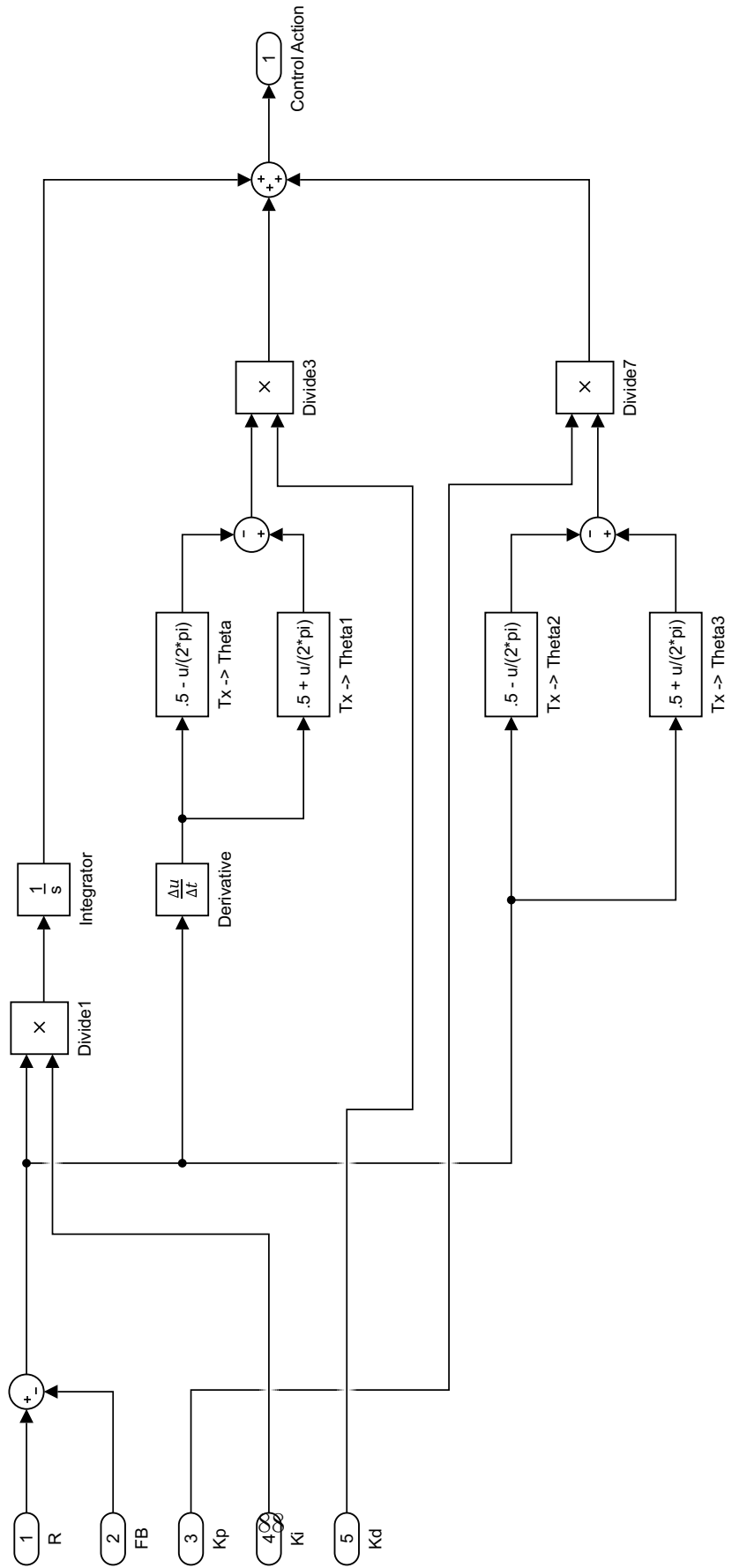


Fig. 6.5: Simulink model, Fuzzy ES controller, *Fuzzy PID Controller*.

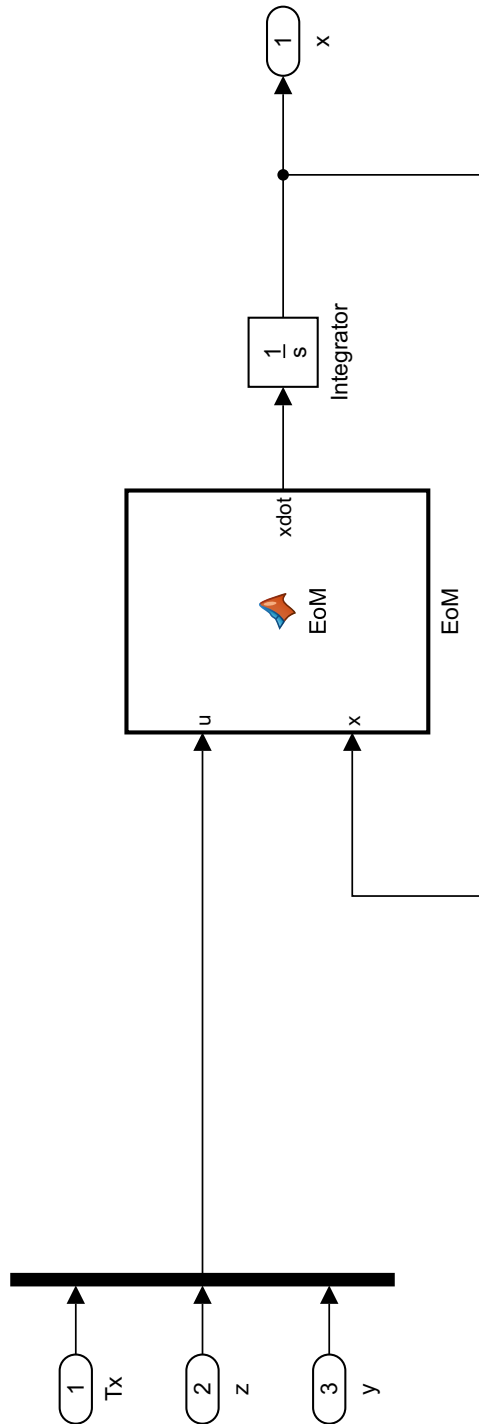


Fig. 6.6: Simulink model, Fuzzy ES controller, *Solar Sail Dynamics*.

EoM.m

```
function xdot = EoM(u,x)
%=====
% 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-synchronous 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
%=====
```

```

% Defining states and inputs
Tx = u(1);
z = u(2);
y = u(3);
Ty = 0;
Tz = 0;

% Calculating the J's
Jx = Ix+mr*(y(1)^2+z(1)^2);
Jy = Iy+mr*z(1)^2;
Jz = Iz+mr*y(1)^2;

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

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]';
end

```

C Type-1 vs Type-2 Fuzzy PID Driving Script

RUNME.m

```
%% Solar Sail Model Setup
close all
T = [0:50:150e3]'; % Set duration of simulation
%
%% Specify initial conditions of Solar Sail craft
x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
x0 = x0*pi/180; % convert degrees to radians
%
%% Specify desired final state of Solar Sail craft
xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
xf = xf*pi/180; % convert degrees to radians

% Solar sail PID Gains
% ZN = [1.2,... % Kp-Tx 12 gains
%      1e-4,... % Ki-Tx
%      .7e3,... % Kd-Tx
%      .5e-2*2,... % Ku-Tx
%      .53,... % Kp_z
%      1e-4,... % Ki_z
%      1.3e3,... % Kd_z
%      28*4,... % Ku_z
%      .54,... % Kp_y
%      1e-4,... % Ki_y
%      1.4e3,... % Kd_y
%      28*4]; % Ku_y

%Updated for new Scaling Factors
ZN = [11.11,... % Kp-Tx
      95.90,... % Ki-Tx
      40.18,... % Kd-Tx
      11.09,... % Kp_z
      96.92,... % Ki_z
      0.75,... % Kd_z
      6.64,... % Kp_y
      7.29,... % Ki_y
      10.96]; % Kd_y

%Scaling Factors
SF = [72/pi,... %P
      0,... %I
      1/.00125,... %D
      (.5e-3)*10,... %Ouput phi
      72/pi,... %P
      0,... %I
      1/.00125,... %D
      28*10,... %Output theta
```

```

72/pi,...      %P
0,...          %I
1/.00125,...   %D
28*10];       %Output psi

% PlotResultSS([ZN],SF,T,x0,xf)

%% Run Particle Swarm
%{
nvars = 9;

c = 0;
% lowerBound = [c c c c .5 c c c c .5 c c c c .5];
% lowerBound = ones(1,nvars)*c;
lowerBound = [c c c c c c c c c];
c = 500;
% a = 10;
% upperBound = [c c c c 3 c c c c 3 c c c c 3];
% upperBound = ones(1,nvars)*c;
upperBound = [c c c c c c c c c];

options = optimoptions('particleswarm',...
    'SwarmSize',100,...
    'InitialSwarmMatrix',ZN,...
    'InitialSwarmSpan',2000,...
    'Display','iter',...
    'DisplayInterval',5,...
    'UseParallel',true,...
    'FunctionTolerance',1e-6,...
    'MaxStallIterations',20,...
    'OutputFcn',@PSoutputDisplay);

% maxNumCompThreads(8);

% PSO_CostFunction = @(x) EvaluateCostFunctionSS(x,T,x0,xf) + EvaluateCostFunctionSS(x,T,xf,x0);
PSO_CostFunction = @(x) EvaluateCostFunctionSS(x,SF,T,x0,xf);

tstart = tic;
[x, fval, exitflag] = particleswarm(PSO_CostFunction, nvars, lowerBound, upperBound, options);

disp('PSO Min. Cost:')
disp(fval)

disp('PSO Output:')
disp(x)

fprintf('This calculation took %dh %dm %ds.\n',floor(toc(tstart)/3600),mod(floor(toc(tstart)/60),60),
    mod(floor(toc(tstart)),60));

save([datestr(now,'yyyy-mm-dd HH.MM'),' PSO_Result.mat']);
%}

```



```

%% Plot Result

% close all

% PlotResultSS ([ZN;x],SF,T(1:find(T>12e3,1)),x0,xf)
% PlotResultSS ([ZN;x],T,xf,x0)
%{
PlotResultSS ([ZN],T,x0,xf)
PlotResultSS ([ZN],T,xf,x0)
%}

%% Plot Stability Margin
% Vary xf to see how good gains are

%% Sweep different final positions for psi
% exclusion = 15;
% nPlots = 10;
% xf1 = [-55, linspace((0-exclusion),(-180+exclusion),nPlots-1)];
% xf2 = zeros(length(xf1),6);
% xf2(:,3) = xf1;
% xf2 = xf2*pi/180;

%% Sweep different final positions for theta
% step = 1; % degrees
% xf1 = [40:step:45];
% xf2 = repmat([0, 0, -55, 0, 0, 0],length(xf1),1);
% xf2(:,1) = xf1;
% xf2 = xf2*pi/180;

%% Sweep different final positions for phi
step = .5; % degrees
% xf1 = [85:step:90];
xf1 = [17];
xf2 = repmat([0, 0, -55, 0, 0, 0],length(xf1),1);
xf2(:,1) = xf1;
xf2 = xf2*pi/180;

load('2017-01-19 23.48 PSO_Result.mat','x')
% x = x/50;
x(2) = 0;
x(1) = x(1)/10;
% x(4) = x(4)/5;
% x(7) = x(7)/5;

%{
%% Sweep different final positions for theta
step = .5; % degrees
% xf1 = [85:step:90];
xf1 = [45,89];
xf2 = repmat([0, 0, -55, 0, 0, 0],length(xf1),1);
xf2(:,2) = xf1;
xf2 = xf2*pi/180;
%}

```

```

load('2017-01-19 23.48 PSO_Result.mat','x')
% x = x/50;
x(2) = 0;
x(1) = x(1)/10;
% x(4) = x(4)/5;
% x(7) = x(7)/5;
%}

%{
%% Sweep different final positions for psi
step = .5; % degrees
% xf1 = [85:step:90];
xf1 = [-5,-2.5,-1,-.5,-.25];
xf2 = repmat([0, 0, -55, 0, 0, 0],length(xf1),1);
xf2(:,3) = xf1;
xf2 = xf2*pi/180;

load('2017-01-19 23.48 PSO_Result.mat','x')
% x = x/50;
x(2) = 0;
x(1) = x(1)/10;
% x(4) = x(4)/5;
% x(7) = x(7)/5;
%}

close all

% noisePower = [ 1e-16;...
%               1e-12];
noisePower = [1e-4]*pi/180;

% [T1,T2] = PlotStabMarginSS_noise(x,SF,T(1:find(T>30e3,1)),x0,xf,noisePower);
% [T1,T2] = PlotStabMarginSS_noise(x,SF,T,x0,xf,noisePower);
[T1,T2] = PlotStabMarginSS2_xf(x,SF,T,x0,xf2,noisePower);

```

EvaluateCostFunctionSS.m

```
function [J] = EvaluateCostFunctionSS(gains ,SF,T,x0,xf)
    J = 0;
%   gains = [2/pi 2/pi 100 .1 1 10];
%   gains = [2/pi 2/pi 1 100 .01 .05 1 20];
%   gains = [1.960540257750709,... % PSO resultant gains
%           0.624562170035862,...
%           0.750539548850577,...
%           26.020674145798253,...
%           0.205202686751593,...
%           0.463034175535731,...
%           1.273135679062371,...
%           10.830490749450147];

    model = 'simSolarSail';
    load_system(model);

    dT = T(2) - T(1);

% Uncomment for fixed-step
%   set_param(model,'SolverType','Fixed-step');

    set_param(model,'StopTime',num2str(T(end)));
    set_param(model,'FixedStep',num2str(dT));
%   set_param(model,'x0',x0);
    set_param([model,'/x0'],'Value',[ '[' ,num2str(x0,12), ']' ]);
    set_param([model,'/xf'],'Value',[ '[' ,num2str(xf,12), ']' ]);
    set_param([model,'/gains'],'Value',[ '[' ,num2str(gains,12), ']' ]);
    set_param([model,'/Scaling Factors'],'Value',[ '[' ,num2str(SF), ']' ]);

    try
        [~,~,Ref,X,U,E] = sim([model,'.slx']);
    catch E
        J = 1e6;
        close_system(model,0);
        return
    end

    X = squeeze(X)';
    Ref = squeeze(Ref)';
%   U = squeeze(U)';

    xW = [1, 1, 1, 0, 0, 0]; % weights given to different states in the cost function
%   stateCostFunctionWeight = [10, 1, 1, 1];
    uW = 1e-2*[1, 1, 1];

    J = J + evalJSS(T,Ref,X,U,xW,uW);

    %{
```

```

set_param([model, '/x0'], 'Value', [' ', num2str(xf), ' ']);
set_param([model, '/xf'], 'Value', [' ', num2str(x0), ' ']);
%   set_param([model, '/gains'], 'Value', [' ', num2str(gains), ' ']);

try
    [T, ~, Ref, X, U] = sim([model, '.slx']);
catch E
    J = 1e6;
    close_system(model, 0);
    return
end

X = squeeze(X)';
%   U = squeeze(U)';

xW = [1, 1, 1, 0, 0, 0]; % weights given to different states in the cost function
%   stateCostFunctionWeight = [10, 1, 1, 1];
uW = 1e-2*[1, 1, 1];

J = J + evalJSS(T, Ref, X, U, xW, uW);
%}

close_system(model, 0);
end

% close all
% figure
% for jj = 1:4
%     subplot(4,1,jj)
%     plot(T, Y(:, jj)')
%     if jj==1
%         hold on
%         plot(T, R, '--k')
%     end
% end

%   for k = 1:3 % Pendulum cost function
%       switch k
%           case 1
%               J = J + 1/T(end) * trapz(T, T.*abs(R - Y(:,k))) * stateCostFunctionWeight(k);
%           case 2
%               % Energy used is the integral of the mass acceleration
%               J = J + trapz(T(1:end-2), abs(diff(diff(U(:,k))/dT)/dT)) *
energyCostFunctionWeight(k);
%               J = J + trapz(T(1:end-2), abs(diff(diff(sum(U,2))/dT)/dT)) *
energyCostFunctionWeight(k);
%           otherwise
%               J = J + 1/T(end) * trapz(T, T.*abs((Y(:,k)*0) - Y(:,k))) * stateCostFunctionWeight(k)
);
%       end
%   end

```

PlotResultSS.m

```
function [] = PlotResultSS(gains,SF,T,x0,xf)
%     gains = [ZN;x];
nLines = size(gains,1);
rrows = 3;
ccols = 2;
names_X = {'\phi','\theta','\psi','d\phi/dt','d\theta/dt','d\psi/dt'};
names_U = {'T_x','y','z'};
names_E = {'e_\phi','e_\theta','e_\psi'};
ulimit_U = [.5e-3,28,28];

dT = T(2)-T(1);
T = 1:dT/10:T(end);
dT = T(2)-T(1);

model = 'simSolarSail';
load_system(model);

h1 = figure();
h2 = figure();
for ii=1:nLines
%     global Kp_phi Ki_phi Kd_phi Kp_theta Ki_theta Kd_theta Kp_psi Ki_psi Kd_psi
%     var = {'Kp_phi','Ki_phi','Kd_phi','Kp_theta','Ki_theta','Kd_theta','Kp_psi','Ki_psi',
%           'Kd_psi'};
%     for i = 1:9
%         assignin('base',char(var(i)),gains(ii,i));
%     end

%     global xf x0 T
%     T = evalin('base','T');
%     Tend = T(end);
%     x0 = evalin('base','x0');
%     xf = evalin('base','xf');
%     lenT = length(T);

%     T = 0:10:30e3; % Set duration of simulation
%
%     % Specify initial conditions of Solar Sail craft
%     x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
%     x0 = x0*pi/180; % convert degrees to radians
%
%     % Specify desired final state of Solar Sail craft
%     xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
%     xf = xf*pi/180; % convert degrees to radians

%     sim('simSolarSail.slx');
%
%     Y = SimOutput.Data(:,:);

set_param(model,'FixedStep',num2str(dT/4));
```

```

%     set_param(model, 'T', T);
set_param(model, 'StopTime', num2str(T(end)));
%     set_param(model, 'x0', x0);
set_param([model, '/x0'], 'Value', ['[', num2str(x0,12), ']']);
set_param([model, '/xf'], 'Value', ['[', num2str(xf,12), ']']);
%     set_param([model, '/gains'], 'Value', ['[', num2str(gains(ii,:),:).*[ones(1,6),ones(1,3)*-1]], ']');
set_param([model, '/gains'], 'Value', ['[', num2str(gains(ii,:),12), ']']);
set_param([model, '/Scaling Factors'], 'Value', ['[', num2str(SF), ']']);
[T,~,Ref,X,U,E] = sim('simSolarSail.slx');

Y = squeeze(X)';
Ref = squeeze(Ref)';

%     stateCostFunctionWeight = [1, 1, 1, .1, .1, .1]./(abs(xf-x0)+(abs(xf-x0)==0)); % weights
%     given to different states in the cost function

%
%     J = 0;
%     for k = 1:size(Y,3)
%         J = J + 1/T(end) * trapz(T,T'.*abs((Y(:,k)*0+xf(k)) - Y(:,k))) *
stateCostFunctionWeight(k);
%     end

figure(h1)
for j = 1:ccols;
    for k = 1:rrows;
        output = (j-1)*rrows+k;
        subplot(rrows,ccols,(k-1)*ccols+j)
        switch ii
            case 1
                plot(T,180/pi*Y(:,output),'--r')
                hold on
                grid on
                plot(T,180/pi*Ref(:,output),'--k')
                if output<=3
                    plot(T,Ref(:,output)*180/pi,'--k')
                else
                    plot(T(1:end-1),180/pi*diff(Ref(:,output-3))/dT,'--k')
                end
            case 2
                plot(T,180/pi*Y(:,output),'--r')
                hold on
                grid on
                plot(T,180/pi*Ref(:,output),'--k')
                if output<=3
                    plot(T,Ref(:,output)*180/pi,'--k')
                else
                    plot(T(1:end-1),180/pi*diff(Ref(:,output-3))/dT,'--k')
                end
            case 3
                plot(T,180/pi*Y(:,output),'--r')
                hold on
                grid on
                plot(T,180/pi*Ref(:,output),'--k')
                if output<=3
                    plot(T,Ref(:,output)*180/pi,'--k')
                else
                    plot(T(1:end-1),180/pi*diff(Ref(:,output-3))/dT,'--k')
                end
            otherwise
                plot(T,180/pi*Y(:,output),'--r')
                hold on
                grid on
                plot(T,180/pi*Ref(:,output),'--k')
                if output<=3
                    plot(T,Ref(:,output)*180/pi,'--k')
                else
                    plot(T(1:end-1),180/pi*diff(Ref(:,output-3))/dT,'--k')
                end
        end
    end
end

%     case nLines
%         plot(T,180/pi*Y(:,output))
%     otherwise
%         plot(T,180/pi*Y(:,output))

```

```

        end
        ylabel(names_X(output));
    end
end

figure(h2)
for k = 1:3;

    output = k;
    subplot(3,2,1+2*(k-1))
    switch ii
        case 1
            plot(T,U(:,output),'-r')
            hold on
            grid on
        case nLines
            plot(T,U(:,output))
            % plot([T(1),T(end)],[1,1]*xf(output)*180/pi,'--k')
            plot([T(1),T(end)],[1,1]*ulimit_U(k),'-k')
            plot([T(1),T(end)],[-1,-1]*ulimit_U(k),'-k')
        otherwise
            plot(T,U(:,output))
    end
    ylabel(names_U(output));

    subplot(3,2,2+2*(k-1))
    switch ii
        case 1
            plot(T,180/pi*E(:,output),'-r')
            ylabel('Error (degrees)')
            hold on
            grid on
        case nLines
            plot(T,180/pi*E(:,output))
        otherwise
            plot(T,180/pi*E(:,output))
    end
    ylabel(names_E(output));

end

end

end

end

%{
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot time responses

iLast = iLast - 1;
figure()

```

```

% ax1 = axes();

%       ax1 = subplot(2,3,1); % plot phi
if iLast-rstLenT+1 > 0
    [~,minCostIdx] = min(J(iLast-rstLenT+1:iLast));
    minCostIdx = minCostIdx + iLast-rstLenT;
else
    [~,minCostIdx] = min(J(1:iLast));
end
if iLast>5
%       plots = round([1,iLast/10,iLast/4,iLast/2,minCostIdx]);
    plots = round([1,minCostIdx]);
else
    plots=1:iLast;
end
;
%       plots = round([1,N/3,indBest]);

lineNames = cell(length(plots)+1,1);

stabTimeLo = 0;
stabTimeHi = 0;
for i = 1:length(plots)

%       lineNames{2*i-1} = sprintf('N=%0.f actual',plots(i));
    lineNames{i+1} = sprintf('N=%0.f',plots(i));
end

% xlabel(ax1,'Time (s)')
if N>1
    legend(lineNames,'location','SouthEast');
end
end
%}

```


PlotStabMarginSS2_xf.m

```
function [outputT1,outputT2] = PlotStabMarginSS2_xf(gains,SF,T,x0,xf,noisePower)

    noisePower = noisePower(1);
%     outputT1 = struct();
%     outputT2 = struct();
    plotDerivatives = true;
    plotError = true;
    lockPlotYScale = true;
%     gains = [ZN;x];
    nLines = size(xf,1);
    rrows = 3;
    ccols = 2;
    names_X = {'\phi [deg]', '\theta [deg]', '\psi [deg]', 'd\phi/dt [deg/s]', 'd\theta/dt [deg/s]', 'd\psi/dt [deg/s]'};
    names_U = {'T_x [N-m]', 'y [m]', 'z [m]'};
    names_E = {'e-\phi [deg]', 'e-\theta [deg]', 'e-\psi [deg]'};
    ulimit_U = [.5e-3,28,28];

    dT = T(2)-T(1);
    T = [1:dT/4:T(end)]';
    dT = T(2)-T(1);

    model = {'simSolarSailT1', 'simSolarSailT2'};
%     modelT2 = 'simSolarSailT2';

%     load_system(modelT1);
%     load_system(modelT2);

    h1 = figure();
    if ~plotDerivatives
        h2 = figure();
    end
    h3 = figure();
    for ii=1:nLines
        load_system(model);
%         global Kp_phi Ki_phi Kd_phi Kp_theta Ki_theta Kd_theta Kp_psi Ki_psi Kd_psi
%         var = {'Kp_phi', 'Ki_phi', 'Kd_phi', 'Kp_theta', 'Ki_theta', 'Kd_theta', 'Kp_psi', 'Ki_psi', 'Kd_psi'};
%         for i = 1:9
%             assignin('base',char(var(i)),gains(ii,i));
%         end

%         global xf x0 T
%         T = evalin('base','T');
%         Tend = T(end);
%         x0 = evalin('base','x0');
%         xf = evalin('base','xf');
%         lenT = length(T);

%         T = 0:10:30e3; % Set duration of simulation
```

```

%
% % Specify initial conditions of Solar Sail craft
% x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
% x0 = x0*pi/180; % convert degrees to radians
%
% % Specify desired final state of Solar Sail craft
% xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
% xf = xf*pi/180; % convert degrees to radians

%
% sim('simSolarSail.slx');
%
% Y = SimOutput.Data(:, :);
for j=1:length(model)
    set_param(model{j}, 'FixedStep', num2str(dT));
%     set_param(model, 'T', T);
    set_param(model{j}, 'StopTime', num2str(T(end)));
%     set_param(model, 'x0', x0);
    set_param([model{j}, '/x0'], 'Value', [' ', num2str(x0,12), ' ']);
    set_param([model{j}, '/xf'], 'Value', [' ', num2str(xf(ii,:),12), ' ']);
%     set_param([model, '/gains'], 'Value', [' ', num2str(gains(ii,:).*[ones(1,6), ones(1,3)*-1])
%     , ' ']);
    set_param([model{j}, '/gains'], 'Value', [' ', num2str(gains,12), ' ']);
    set_param([model{j}, '/Scaling Factors'], 'Value', [' ', num2str(SF), ' ']);
    set_param([model{j}, '/Band-Limited White Noise'], 'Cov', [' ', num2str(noisePower), ' ']);
    set_param([model{j}, '/Band-Limited White Noise'], 'Ts', num2str(dT));
end
[T1,~, Ref1, X1, U1, E1] = sim([model{1}, '.slx']);
[T2,~, Ref2, X2, U2, E2] = sim([model{2}, '.slx']);

Y1 = squeeze(X1);
Y2 = squeeze(X2);
Ref1 = squeeze(Ref1)';
Ref2 = squeeze(Ref2)';

%     stateCostFunctionWeight = [1, 1, 1, .1, .1, .1]./(abs(xf-x0)+ (abs(xf-x0)==0)); % weights
%     given to different states in the cost function
%
%     J = 0;
%     for k = 1:size(Y,3)
%         J = J + 1/T(end) * trapz(T,T'.*abs((Y(:,k)*0+xf(k)) - Y(:,k))) *
stateCostFunctionWeight(k);
%     end
figure(h1)
if plotDerivatives
    for j = 1:ccols;
        for k = 1:rrows;
            output = (j-1)*rrows+k;
            subplot(rrows, ccols, (k-1)*ccols+j);
            switch ii
                case 1
                    plot(T1,180/pi*Ref1(:, output), ':k', 'LineWidth', 1)
                    if lockPlotYScale

```

```

        axis manual
    else
        axis auto
    end
    hold on
    grid on
    plot(T1,180/pi*Y1(:,output),'-r')
    plot(T2,180/pi*Y2(:,output),'--b')

%         if output<=3
%             plot(T,Ref(:,output)*180/pi,'--k')
%         else
%             plot(T(1:end-1),180/pi*diff(Ref(:,output-3))/dT,'--k')
%         end
%{

switch output
case 1
    temp = axis();
    axis([temp(1),temp(2),-10,10]);
case 2
    temp = axis();
    axis([temp(1),temp(2),-10,10]);
case 3
    temp = axis();
    axis([temp(1),temp(2),-100,10]);
otherwise

    end

%}

case nLines
    plot(T,180/pi*Y(:,output))
    axis
otherwise
    plot(T1,180/pi*Y1(:,output),'-')
    plot(T2,180/pi*Y2(:,output),'-k','LineWidth',1)

end
ylabel(names_X(output));
if (k-1)*ccols+j >=5
    xlabel('Time [s]')
end
end
if j==1
    legend('Ref','T1','T2','Location','Best');
end
end
else
for k = 1:rows;
    output = k;
    switch output
    case 3
        subplot(rows,ccols,[3,4,5,6])
    otherwise
        subplot(rows,ccols,output)
    end
end
end

```

```

switch ii
    case 1
        plot(T1,180/pi*Y1(:,output),'-r')
        hold on
        grid on
        plot(T2,180/pi*Y2(:,output),'--b','LineWidth',1)
        xlabel('Time [s]')
    otherwise
        plot(T1,180/pi*Y1(:,output),'-')
        plot(T2,180/pi*Y2(:,output),'-k','LineWidth',1)
end
ylabel(names_X(output));
end

end

if ~plotDerivatives
    figure(h2)
    for k = 1:3;
        output = k+3;
        subplot(3,1,k);
        switch ii
            case 1
                plot(T1(1:end-1),180/pi*diff(Y1(:,output))/dT,'-r')
                hold on
                grid on
                plot(T2(1:end-1),180/pi*diff(Y2(:,output))/dT,'--b','LineWidth',1)
                if output==1
                    title('Acceleration')
                end
            otherwise
                plot(T1(1:end-1),180/pi*diff(Y1(:,output))/dT,'-')
                plot(T2(1:end-1),180/pi*diff(Y2(:,output))/dT,'-k','LineWidth',1)
        end
        ylabel(names_X(output));
    end
end

%
figure(h3)
if plotError
    for k = 1:3;

        output = k;
        subplot(3,2,1+2*(k-1))
        switch ii
            case 1
                plot(T1,U1(:,output),'-r')
                hold on
                grid on
                plot(T2,U2(:,output),'--b')
                plot([T1(1),T1(end)], [1,1]*ulimit_U(k),'--k')
            otherwise
                plot(T1,U1(:,output),'-')
                plot(T2,U2(:,output),'-k')
        end
    end
end

```

```

%           plot([T1(1),T1(end)],[-1,-1]*ulimit_U(k),'--k')
%           xlabel('Time [s]')
case nLines
    plot(T1,U1(:,output))
    plot(T2,U2(:,output),'-.')
%           plot([T(1),T(end)],[1,1]*xf(output)*180/pi,'--k')

    otherwise
        plot(T1,U1(:,output))
end
ylabel(names_U(output));
if 1+2*(k-1) >=5
    xlabel('Time [s]')
end

subplot(3,2,2+2*(k-1))
switch ii
case 1
%           plot(T1,180/pi*E1(:,output),'-r')
%           ylabel('Error (degrees)')
        hold on
        grid on
        plot(T2,180/pi*E2(:,output),'--b')

    otherwise
        plot(T1,180/pi*E1(:,output))
        plot(T2,180/pi*E2(:,output),'-.')
end
ylabel(names_E(output));
if 2+2*(k-1) >=5
    xlabel('Time [s]')
end

end
else
for k = 1:3;

output = k;
subplot(3,1,k)
switch ii
case 1
    if output==1
        title('Control Action')
    end
    plot(T1,U1(:,output))
    hold on
    grid on
    plot(T2,U2(:,output),'-k')
    plot([T1(1),T1(end)],[1,1]*ulimit_U(k),'--k')
    plot([T1(1),T1(end)],[-1,-1]*ulimit_U(k),'--k')
%           xlabel('Time [s]')
    otherwise
        plot(T1,U1(:,output))

```

```

        plot(T2,U2(:,output),'-k')
    end
    ylabel(names_U(output));
end
end

close_system(model,0);
end
% close_system(modelT1,0);

outputT1 = struct('T',T1,...
    'Ref',Ref1,...
    'X',Y1,...
    'U',U1,...
    'E',E1);

outputT2 = struct('T',T2,...
    'Ref',Ref2,...
    'X',Y2,...
    'U',U2,...
    'E',E2);

end

```

PlotStabMarginSS_noise.m

```
function [outputT1,outputT2] = PlotStabMarginSS_noise(gains,SF,T,x0,xf,noisePower)

    noisePower = noisePower(1);
%     outputT1 = struct();
%     outputT2 = struct();
    plotDerivatives = true;
    plotError = true;
%     gains = [ZN;x];
    nLines = length(noisePower);
    rrows = 3;
    ccols = 2;
    names_X = {'\phi','\theta','\psi','d\phi/dt','d\theta/dt','d\psi/dt'};
    names_U = {'T_x','y','z'};
    names_E = {'e_\phi','e_\theta','e_\psi'};
    ulimit_U = [.5e-3,28,28];

    dT = T(2)-T(1);
    T = [1:dT/4:T(end)]';
    dT = T(2)-T(1);

    model = {'simSolarSailT1','simSolarSailT2'};
%     modelT2 = 'simSolarSailT2';

%     load_system(modelT1);
%     load_system(modelT2);

    h1 = figure();
    if ~plotDerivatives
        h2 = figure();
    end
    h3 = figure();
    for ii=1:nLines
        load_system(model);
%         global Kp_phi Ki_phi Kd_phi Kp_theta Ki_theta Kd_theta Kp_psi Ki_psi Kd_psi
%         var = {'Kp_phi','Ki_phi','Kd_phi','Kp_theta','Ki_theta','Kd_theta','Kp_psi','Ki_psi',
%             'Kd_psi'};
%         for i = 1:9
%             assignin('base',char(var(i)),gains(ii,i));
%         end

%         global xf x0 T
%         T = evalin('base','T');
%         Tend = T(end);
%         x0 = evalin('base','x0');
%         xf = evalin('base','xf');
%         lenT = length(T);

%         T = 0:10:30e3; % Set duration of simulation
%
%         % Specify initial conditions of Solar Sail craft
```

```

%     x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
%     x0 = x0*pi/180; % convert degrees to radians
%
%     % Specify desired final state of Solar Sail craft
%     xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
%     xf = xf*pi/180; % convert degrees to radians

%
%     sim('simSolarSail.slx');
%
%     Y = SimOutput.Data(:, :);
for j=1:length(model)
    set_param(model{j}, 'FixedStep', num2str(dT));
%     set_param(model, 'T', T);
    set_param(model{j}, 'StopTime', num2str(T(end)));
%     set_param(model, 'x0', x0);
    set_param([model{j}, '/x0'], 'Value', [' ', num2str(x0,12), ' ']);
    set_param([model{j}, '/xf'], 'Value', [' ', num2str(xf,12), ' ']);
%     set_param([model, '/gains'], 'Value', [' ', num2str(gains(ii,:)).*[ones(1,6),ones(1,3)*-1]
%     , ' ']);
    set_param([model{j}, '/gains'], 'Value', [' ', num2str(gains,12), ' ']);
    set_param([model{j}, '/Scaling Factors'], 'Value', [' ', num2str(SF), ' ']);
    set_param([model{j}, '/Band-Limited White Noise'], 'Cov', [' ', num2str(noisePower(ii)), ' ']);
    ;
    set_param([model{j}, '/Band-Limited White Noise'], 'Ts', num2str(dT));
end
[T1,~, Ref1,X1,U1,E1] = sim([model{1}, '.slx']);
[T2,~, Ref2,X2,U2,E2] = sim([model{2}, '.slx']);

Y1 = squeeze(X1);
Y2 = squeeze(X2);
Ref1 = squeeze(Ref1)';
Ref2 = squeeze(Ref2)';

%     stateCostFunctionWeight = [1, 1, 1, .1, .1, .1]./(abs(xf-x0)+ (abs(xf-x0)==0)); % weights
%     given to different states in the cost function
%
%     J = 0;
%     for k = 1:size(Y,3)
%         J = J + 1/T(end) * trapz(T,T'.*abs((Y(:,k)*0+xf(k)) - Y(:,k))) *
stateCostFunctionWeight(k);
%     end
figure(h1)
if plotDerivatives
    for j = 1:ccols;
        for k = 1:rrows;
            output = (j-1)*rrows+k;
            subplot(rrows,ccols,(k-1)*ccols+j);
            switch ii
                case 1

                    plot(T1,180/pi*Y1(:, output), '-');
                    hold on

```



```

end

end

if ~plotDerivatives
    figure(h2)
    for k = 1:3;
        output = k+3;
        subplot(3,1,k);
        switch ii
            case 1
                plot(T1(1:end-1),180/pi*diff(Y1(:,output))/dT,'-')
                hold on
                grid on
                plot(T2(1:end-1),180/pi*diff(Y2(:,output))/dT,'-k','LineWidth',1)
                if output==1
                    title('Acceleration')
                end
            otherwise
                plot(T1(1:end-1),180/pi*diff(Y1(:,output))/dT,'-')
                plot(T2(1:end-1),180/pi*diff(Y2(:,output))/dT,'-k','LineWidth',1)
        end
        ylabel(names_X(output));
    end
end

%
figure(h3)
if plotError
    for k = 1:3;

        output = k;
        subplot(3,2,1+2*(k-1))
        switch ii
            case 1
                plot(T1,U1(:,output),'-r')
                hold on
                grid on
                plot(T2,U2(:,output),'-')
                %
                plot([T1(1),T1(end)],[1,1]*ulimit_U(k),'--k')
                %
                plot([T1(1),T1(end)],[-1,-1]*ulimit_U(k),'--k')
                %
                xlabel('Time [s]')
            case nLines
                plot(T1,U1(:,output))
                plot(T2,U2(:,output),'-')
                %
                plot([T(1),T(end)],[1,1]*xf(output)*180/pi,'--k')

            otherwise
                plot(T1,U1(:,output))
        end
        ylabel(names_U(output));

        subplot(3,2,2+2*(k-1))
    end
end

```

```

switch ii
case 1
    plot(T1,180/pi*E1(:,output))
    ylabel('Error (degrees)')
    hold on
    grid on
    plot(T2,180/pi*E2(:,output),'-.')

otherwise
    plot(T1,180/pi*E1(:,output))
    plot(T2,180/pi*E2(:,output),'-.')
end
ylabel(names_E(output));

end
else
for k = 1:3;

    output = k;
    subplot(3,1,k)
    switch ii
    case 1
        if output==1
            title('Control Action')
        end
        plot(T1,U1(:,output))
        hold on
        grid on
        plot(T2,U2(:,output),'-k')
        plot([T1(1),T1(end)],[1,1]*ulimit_U(k),'-k')
        plot([T1(1),T1(end)],[-1,-1]*ulimit_U(k),'-k')
        xlabel('Time [s]')
    otherwise
        plot(T1,U1(:,output))
        plot(T2,U2(:,output),'-k')
    end
    ylabel(names_U(output));
end
end

close_system(model,0);
end
% close_system(modelT1,0);

outputT1 = struct('T',T1,...
    'Ref',Ref1,...
    'X',Y1,...
    'U',U1,...
    'E',E1);

outputT2 = struct('T',T2,...
    'Ref',Ref2,...

```

```
'X', Y2, ...  
'U', U2, ...  
'E', E2);
```

```
end
```

PlotStabMarginSSx0.m

```

function [] = PlotStabMarginSSx0(gains,SF,T,x0,xf)
    plotDerivatives = false;
%     gains = [ZN;x];
    nLines = size(x0,1);
    rrows = 3;
    ccols = 2;
    names_X = {'\phi','\theta','\psi','d\phi/dt','d\theta/dt','d\psi/dt'};
    names_U = {'T_x','y','z'};
    names_E = {'e_\phi','e_\theta','e_\psi'};
    ulimit_U = [.5e-3,28,28];

    dT = T(2)-T(1);
    T = [1:dT/10:T(end)]';
    dT = T(2)-T(1);

    model = {'simSolarSailT1','simSolarSailT2'};
%     modelT2 = 'simSolarSailT2';

%     load_system(modelT1);
%     load_system(modelT2);

    h1 = figure();
%     h2 = figure();
    for ii=1:nLines
        load_system(model);
%         global Kp_phi Ki_phi Kd_phi Kp_theta Ki_theta Kd_theta Kp_psi Ki_psi Kd_psi
%         var = {'Kp_phi','Ki_phi','Kd_phi','Kp_theta','Ki_theta','Kd_theta','Kp_psi','Ki_psi',
%         'Kd_psi'};
%         for i = 1:9
%             assignin('base',char(var(i)),gains(ii,i));
%         end

%         global xf x0 T
%         T = evalin('base','T');
%         Tend = T(end);
%         x0 = evalin('base','x0');
%         xf = evalin('base','xf');
%         lenT = length(T);

%         T = 0:10:30e3; % Set duration of simulation
%
%         % Specify initial conditions of Solar Sail craft
%         x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
%         x0 = x0*pi/180; % convert degrees to radians
%
%         % Specify desired final state of Solar Sail craft
%         xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
%         xf = xf*pi/180; % convert degrees to radians

```

```

%       sim('simSolarSail.slx');
%
%       Y = SimOutput.Data(:,:);
for j=1:length(model)
    set_param(model{j}, 'FixedStep', num2str(dT));
%       set_param(model, 'T', T);
    set_param(model{j}, 'StopTime', num2str(T(end)));
%       set_param(model, 'x0', x0);
    set_param([model{j}, '/x0'], 'Value', [' ', num2str(x0(ii,:), 12), ' ']);
    set_param([model{j}, '/xf'], 'Value', [' ', num2str(xf, 12), ' ']);
%       set_param([model, '/gains'], 'Value', [' ', num2str(gains(ii,:) .* [ones(1,6), ones(1,3)*-1]),
%       ' ']);
    set_param([model{j}, '/gains'], 'Value', [' ', num2str(gains, 12), ' ']);
    set_param([model{j}, '/Scaling Factors'], 'Value', [' ', num2str(SF), ' ']);
end
[T1,~, Ref1, X1, U1, E1] = sim([model{1}, '.slx']);
[T2,~, Ref1, X2, U2, E2] = sim([model{2}, '.slx']);

Y1 = squeeze(X1)';
Y2 = squeeze(X2)';
Ref1 = squeeze(Ref1)';

%       stateCostFunctionWeight = [1, 1, 1, .1, .1, .1]./(abs(xf-x0)+ (abs(xf-x0)==0)); % weights
%       given to different states in the cost function
%
%       J = 0;
%       for k = 1:size(Y,3)
%           J = J + 1/T(end) * trapz(T,T'.*abs((Y(:,k)*0+xf(k)) - Y(:,k))) *
stateCostFunctionWeight(k);
%       end
figure(h1)
if plotDerivatives
    for j = 1:ccols;
        for k = 1:rrows;
            output = (j-1)*rrows+k;
            subplot(rrows, ccols, (k-1)*ccols+j);
            switch ii
                case 1

                    plot(T1, 180/pi*Y1(:, output), '-');
                    hold on
                    grid on
                    plot(T2, 180/pi*Y2(:, output), ':k', 'LineWidth', 2)
                    plot(T, 180/pi*Ref(:, output), ':k', 'LineWidth', 2)
                    if output <= 3
                        plot(T, Ref(:, output)*180/pi, '--k')
                    else
                        plot(T(1:end-1), 180/pi*diff(Ref(:, output-3))/dT, '--k')
                    end
                    end

            switch output
                case 1
                    temp = axis();

```



```

%             plot ([T(1),T(end)], [1,1]* ulimit_U(k), '--k')
%             plot ([T(1),T(end)], [-1,-1]* ulimit_U(k), '--k')
%         case nLines
%             plot(T,U(:, output))
% %         plot ([T(1),T(end)], [1,1]* xf(output)*180/pi, '--k')
%
%         otherwise
%             plot(T,U(:, output))
%     end
%     ylabel(names_U(output));
%
%     subplot(3,2,2+2*(k-1))
%     switch ii
%     case 1
%         plot(T,180/pi*E(:, output), '--r')
%         ylabel('Error (degrees)')
%         hold on
%         grid on
%     case nLines
%         plot(T,180/pi*E(:, output))
%     otherwise
%         plot(T,180/pi*E(:, output))
%     end
%     ylabel(names_E(output));
%
%
%
%
%     end

    close_system(model,0);
end
%     close_system(modelT1,0);

end

%{
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot time responses

iLast = iLast - 1;
figure()
% ax1 = axes();

%     ax1 = subplot(2,3,1); % plot phi
if iLast-rstLenT+1 > 0
    [~, minCostIdx] = min(J(iLast-rstLenT+1:iLast));
    minCostIdx = minCostIdx + iLast-rstLenT;
else
    [~, minCostIdx] = min(J(1:iLast));
end
if iLast>5
%     plots = round([1,iLast/10,iLast/4,iLast/2,minCostIdx]);
    plots = round([1,minCostIdx]);

```



```

else
    plots=1:iLast;
end
;
%   plots = round([1,N/3,indBest]);

lineNames = cell(length(plots)+1,1);

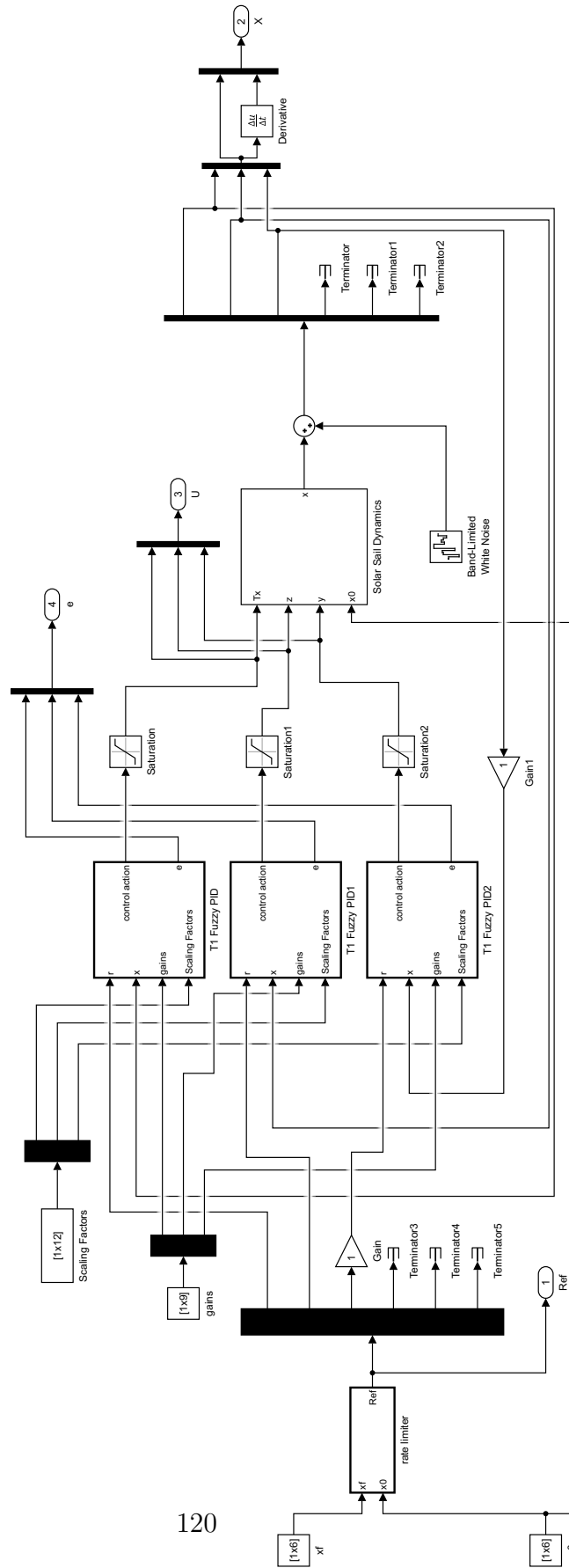
stabTimeLo = 0;
stabTimeHi = 0;
for i = 1:length(plots)

%   lineNames{2*i-1} = sprintf('N=%0.f actual',plots(i));
    lineNames{i+1} = sprintf('N=%0.f',plots(i));
end

% xlabel(ax1,'Time (s)')
if N>1
    legend(lineNames,'location','SouthEast');
end
end
%}

```

D Type-1 Fuzzy PID Controller



120

Fig. 6.7: Simulink model, Type-1 Fuzzy PID controller, base.

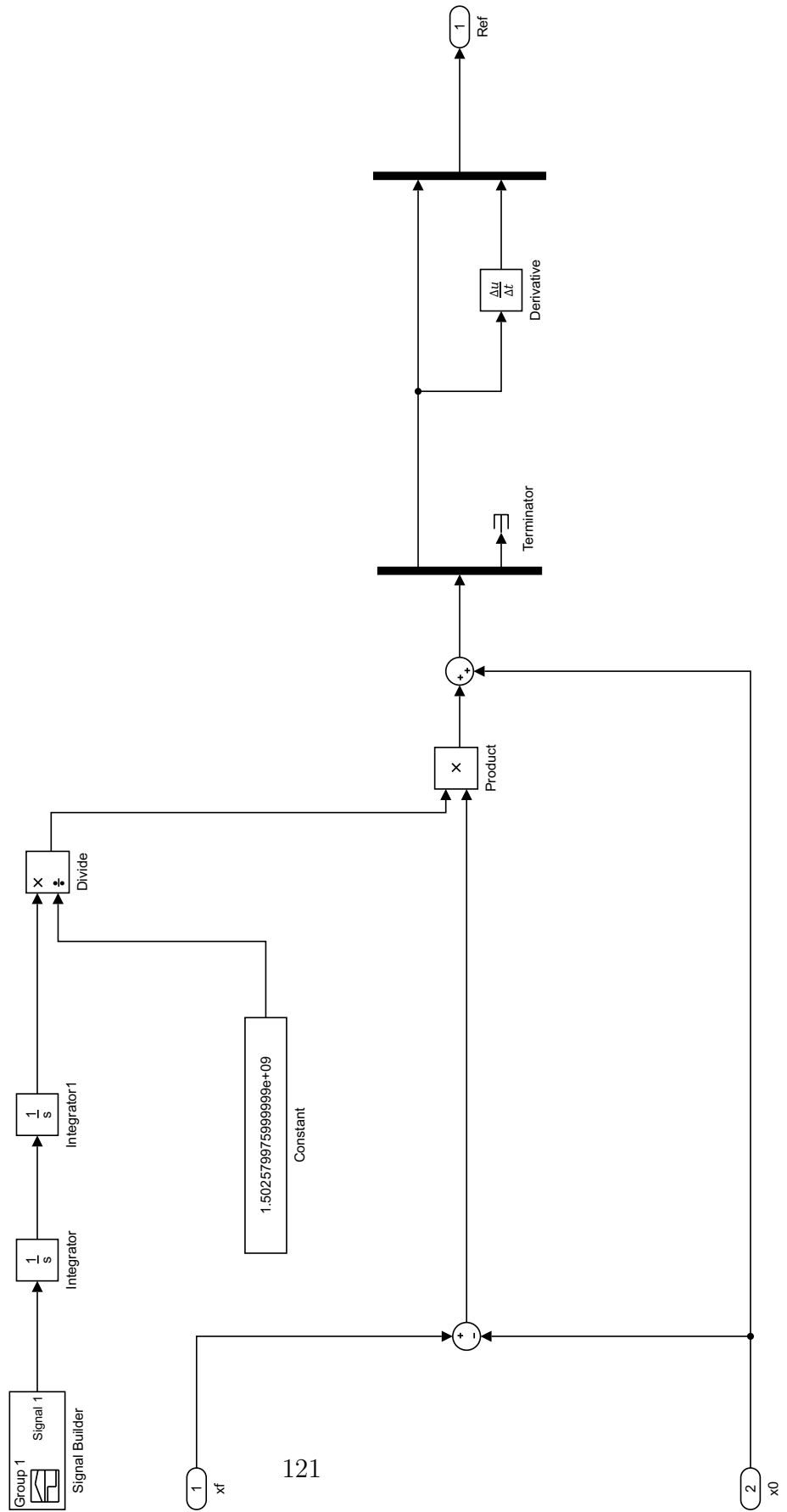


Fig. 6.8: Simulink model, Type-1 Fuzzy PID controller, *Trajectory Generator*.

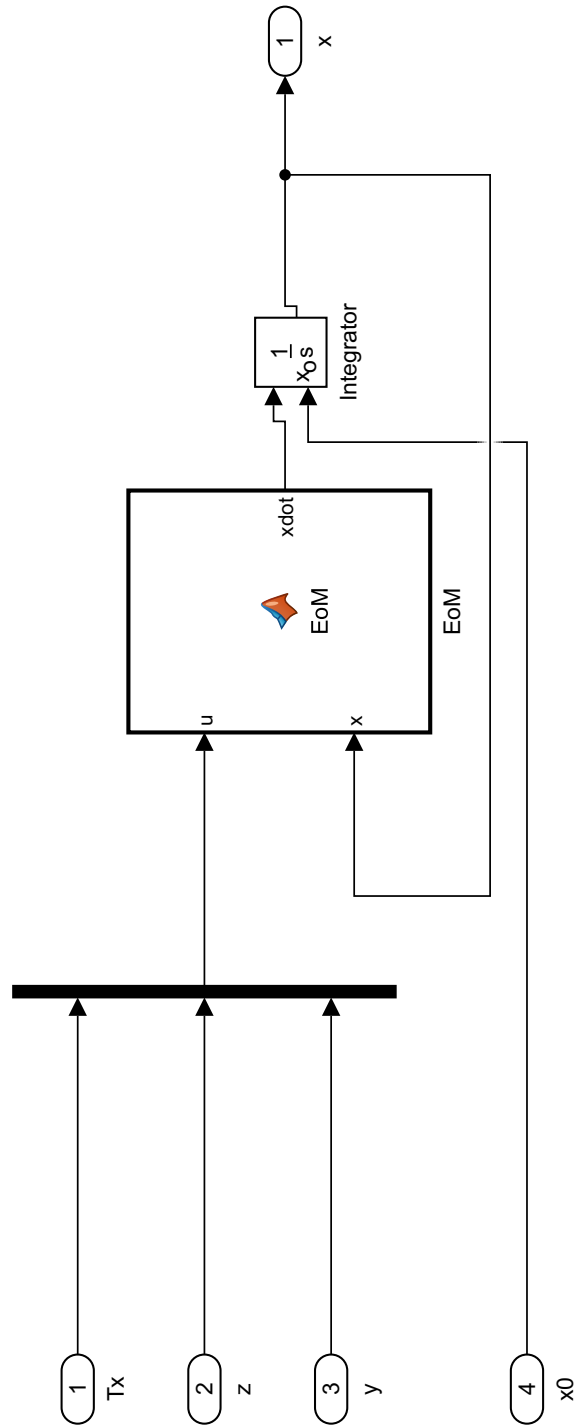


Fig. 6.9: Simulink model, Type-1 Fuzzy PID controller, *Solar Sail Dynamics*.

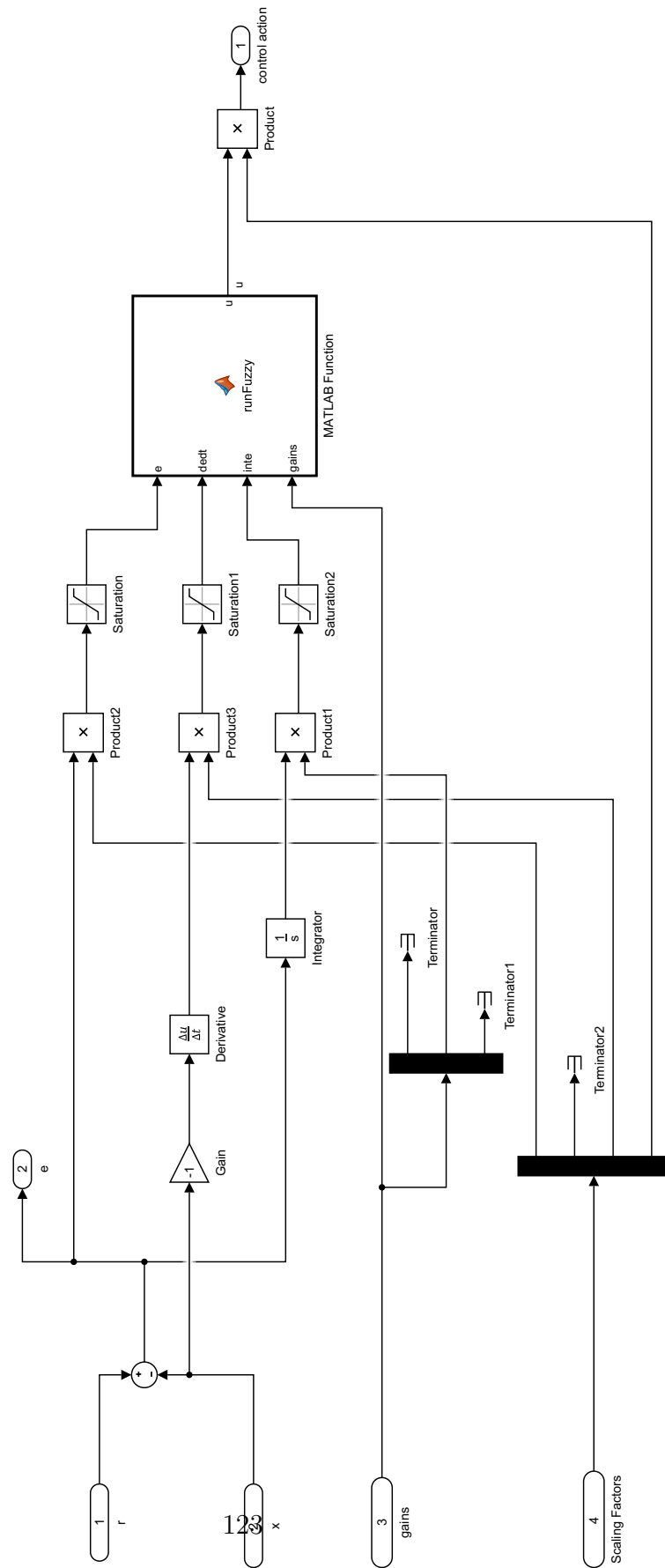


Fig. 6.10: Simulink model, Type-1 Fuzzy PID controller, *Type-1 Fuzzy PID Controller*.

EoM.m

```
function xdot = EoM(u,x)
    xdot = x*0;
    %=====
    % 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=0.05; %m %cp-cp offset (was 0.1)
    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-synchronous 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
```

```

%=====
% Defining states and inputs
Tx = u(1);
z = u(2);
y = u(3);
Ty = 0;
Tz = 0;

% 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);

% Linearized EOM about phi=0, theta=0
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;
%}

% Non-linear equations of motion (fugly)
phi      = x(1); % LVLH frame
theta    = x(2); % LVLH frame
psi      = x(3); % LVLH frame
Wx       = x(4); % Body frame
Wy       = x(5); % Body frame
Wz       = x(6); % Body frame

alpha = pi/2 + psi; % angle b/w sun line and roll axis (P161)
F = -Fs*cos(alpha)^2;

% Gravity gradient torques
Rx = -sin(theta);
Ry = sin(phi)*cos(theta);
Rz = cos(phi)*cos(theta);

Gx = -3*n^2*(Jy-Jz)*Ry*Rz;

```



```

Gy = -3*n^2*(Jz-Jx)*Rz*Rx;
Gz = -3*n^2*(Jx-Jy)*Rx*Ry;
% Gx = 0;
% Gy = 0;
% Gz = 0;

% Wx = dWxdt;
% Wy = dWydt;
% Wz = dWzdt;
dWxdt = (1/Jx)*((Jy - Jz)*Wy*Wz + Gx + 0.5*epsilon*F + Tx);
dWydt = (1/Jy)*((Jz - Jx)*Wz*Wx + Gy - m/(M+2*m)*(z*F) + epsilon*F + Ty);
dWzdt = (1/Jz)*((Jx - Jy)*Wx*Wy + Gz - m/(M+2*m)*(y*F) + epsilon*F + Tz);

% Orientation of Spacecraft in LVLH angles
% psi points to center of earth
% phi points along direction of travel (transverse)
% theta is perpendicular to orbit plane
% phi = dphidt;
% theta = dthetadt;
% psi = dpsidt;
dphidt = (1/cos(theta))*(Wx*cos(theta) + Wy*sin(phi)*sin(theta) + Wz*cos(phi)*sin(theta))
+ (n/cos(theta))*sin(psi);
dthetadt = (1/cos(theta))*(Wx*0 + Wy*cos(phi)*cos(theta) - Wz*sin(phi)*cos(theta))
+ (n/cos(theta))*cos(theta)*cos(psi);
dpsidt = (1/cos(theta))*(Wx*0 + Wy*sin(phi) + Wz*cos(phi)
+ (n/cos(theta))*sin(theta)*sin(psi));

xdot(1) = dphidt;
xdot(2) = dthetadt;
xdot(3) = dpsidt;
xdot(4) = dWxdt;
xdot(5) = dWydt;
xdot(6) = dWzdt;

% Zero-thrust angle has psi = 0
% Full-thrust angle has psi = -90;

```

end

runT1Fuzzy.m

```
function u = runFuzzy(e, dedt, inte, gains)

    x1 = e;      % x1 is state error
    x2 = dedt;  % x2 is derivative of state error
    x3 = inte;  % x3 is integral of state error

    % Define MFs
    % define 11 gaussian MFs
    nMF = 9;
    C = linspace(-1,1,nMF);
    sU = C(2)-C(1);
    %   sL = sU;
    %   kU = 1.0;
    %   kL = 0.8;

    xMFU = [-2,-sU,0,sU,2];
    %   xMFL = [-2,-sL,0,sL,2];
    yMF = [0,0,1,0,0];

    X1U = ones(1,nMF);
    %   X1L = ones(1,nMF);
    X2U = ones(1,nMF);
    %   X2L = ones(1,nMF);
    X3U = ones(1,nMF);
    %   X3L = ones(1,nMF);

    % Triangular MFs
    for i = 1:nMF
        X1U(i) = interp1q(xMFU'+C(i),yMF',x1);
    %   X1L(i) = interp1(xMFL+C(i),kL*yMF,x1);
        X2U(i) = interp1q(xMFU'+C(i),yMF',x2);
    %   X2L(i) = interp1(xMFL+C(i),kL*yMF,x2);
        X3U(i) = interp1q(xMFU'+C(i),yMF',x3);
    %   X3L(i) = interp1(xMFL+C(i),kL*yMF,x3);
    end

    % Gaussian MFs
    %   X1U = kU*exp((-0.5*(x1-C).^2)./sU.^2);
    %   X1L = kL*exp((-0.5*(x1-C).^2)./sL.^2);
    %
    %   X2U = kU*exp((-0.5*(x2-C).^2)./sU.^2);
    %   X2L = kL*exp((-0.5*(x2-C).^2)./sL.^2);
    %
    %   X3U = kU*exp((-0.5*(x3-C).^2)./sU.^2);
    %   X3L = kL*exp((-0.5*(x3-C).^2)./sL.^2);

    %   dedtExp = eExp;
    %   c0 = (nMF-1)/2;
    %   temp = -c0:c0;
    temp = linspace(-1,1,nMF);
```

```

temp_dedt = temp*gains(3);
%   temp_dedt(1) = -10;
%   temp_dedt(end) = 10;
%   temp_dedt = sign(temp_dedt).*(temp_dedt.^2);
temp_inte = temp * 1e-3;
%   temp_inte(1) = -3;
%   temp_inte(end) = 3;
%   temp_e = (sign(temp).*abs(temp).^eExp);
%   temp_dedt = sign(temp).*abs(temp).^dedtExp;
%   temp_inte = sign(temp).*abs(temp).^eExp;

rule_mat = repmat((temp_dedt)'*ones(1,nMF) + ones(1,nMF)'*(temp*gains(1)),[1,1,nMF]) + permute(
    repmat(ones(1,nMF)'*(temp_inte),[1,1,nMF]),[3,1,2]);

%{
Rows correspond to e MFs, columns to de/dt MFs, entries are crisp
outputs

    ans =

    10     9     8     7     6     5     4     3     2     1     0
     9     8     7     6     5     4     3     2     1     0    -1
     8     7     6     5     4     3     2     1     0    -1    -2
     7     6     5     4     3     2     1     0    -1    -2    -3
     6     5     4     3     2     1     0    -1    -2    -3    -4
     5     4     3     2     1     0    -1    -2    -3    -4    -5
     4     3     2     1     0    -1    -2    -3    -4    -5    -6
     3     2     1     0    -1    -2    -3    -4    -5    -6    -7
     2     1     0    -1    -2    -3    -4    -5    -6    -7    -8
     1     0    -1    -2    -3    -4    -5    -6    -7    -8    -9
     0    -1    -2    -3    -4    -5    -6    -7    -8    -9   -10

%}

%calculate firing strength of each rule using min function
FU = min(min(repmat(X2U'*ones(1,nMF),[1,1,nMF]),repmat(ones(1,nMF)'*X1U,[1,1,nMF])),permute(
    repmat(ones(1,nMF)'*X3U,[1,1,nMF]),[3,1,2])); % Firing strength of upper MFs
%   FL = min(min(repmat(X2L'*ones(1,nMF),[1,1,nMF]),repmat(ones(1,nMF)'*X1L,[1,1,nMF])),permute(
    repmat(ones(1,nMF)'*X3L,[1,1,nMF]),[3,1,2])); % Firing strength of lower MFs
%   FL = min(X2L'*ones(1,nMF),ones(1,nMF)'*X1L); % Firing strength of lower MFs

%   u = sum(rule_mat(:).*(FU(:)+FL(:)))/sum(FU(:)+FL(:));
u = sum(rule_mat(:).*FU(:))/sum(FU(:));
end

```

E Type-2 Fuzzy PID Controller

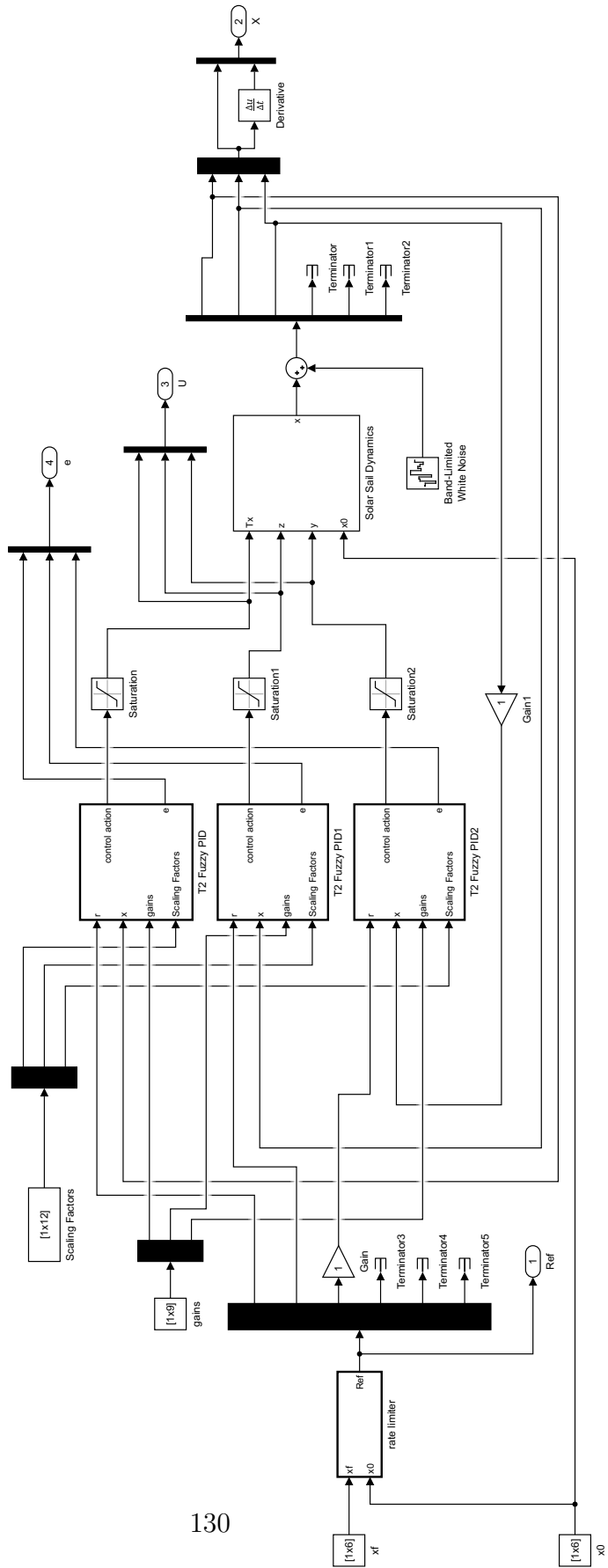


Fig. 6.11: Simulink model, Type-2 Fuzzy PID controller, base.

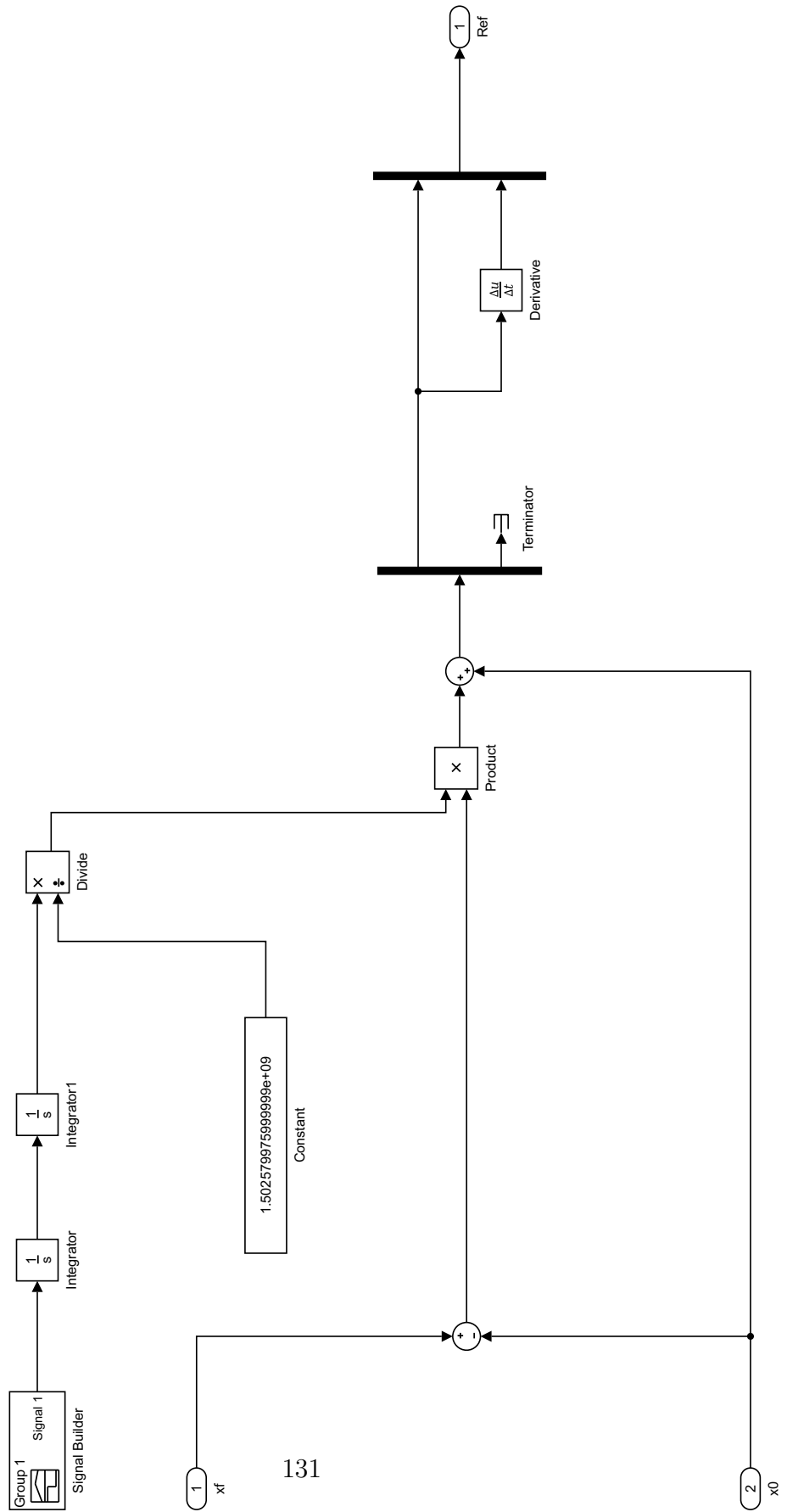


Fig. 6.12: Simulink model, Type-2 Fuzzy PID controller, *Trajectory Generator*.

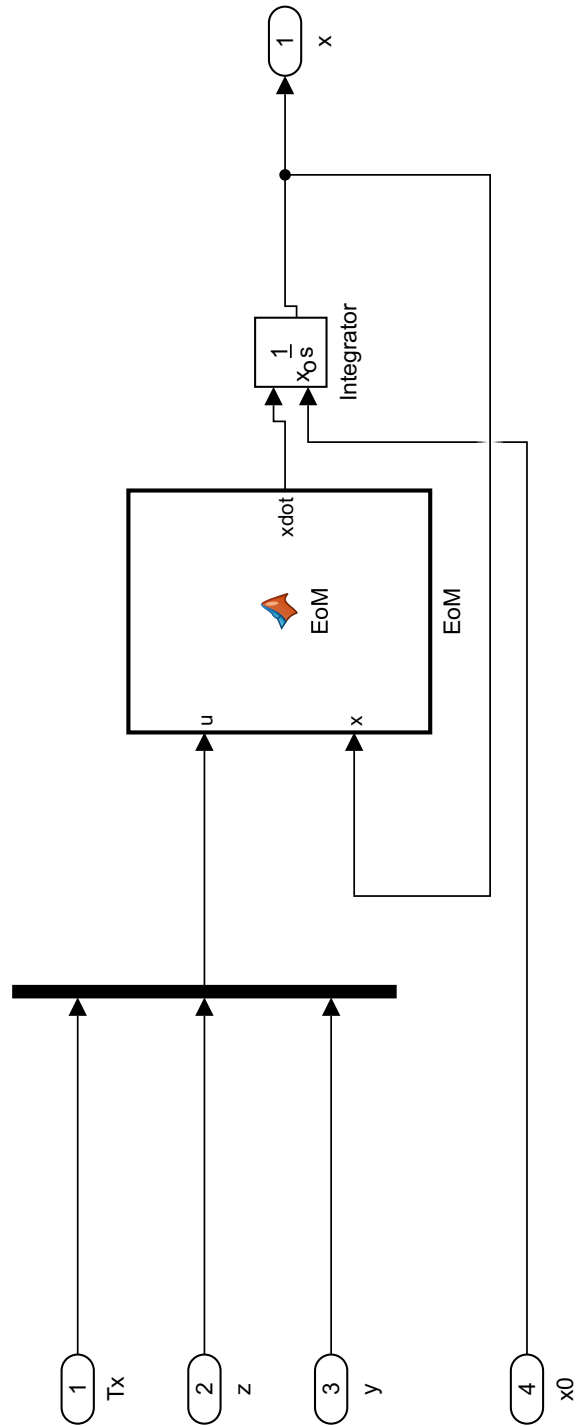


Fig. 6.13: Simulink model, Type-2 Fuzzy PID controller, *Solar Sail Dynamics*.

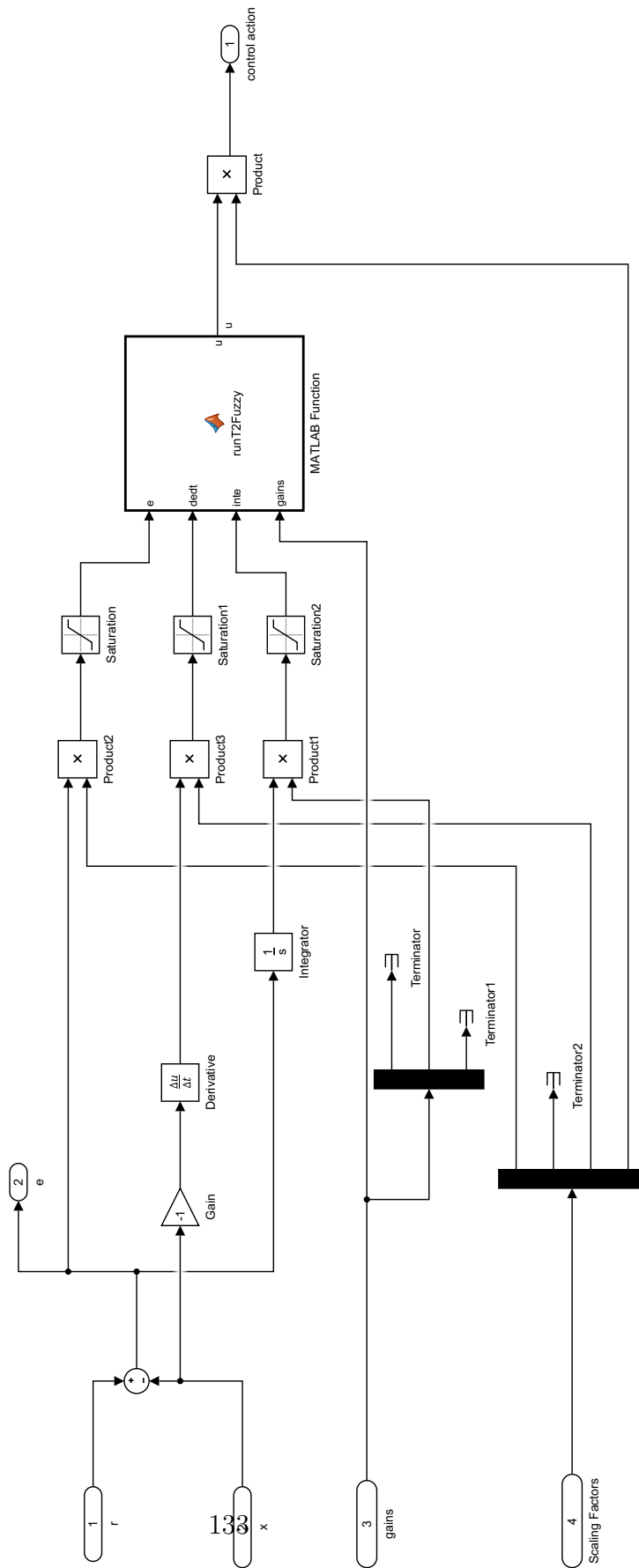


Fig. 6.14: Simulink model, Type-2 Fuzzy PID controller, *Type-2 Fuzzy PID Controller*.

EoM.m

```
function xdot = EoM(u,x)
    xdot = x*0;
    %=====
    % 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=0.05; %m %cp-cp offset (was 0.1)
    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-synchronous 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
```

```

%=====
% Defining states and inputs
Tx = u(1);
z = u(2);
y = u(3);
Ty = 0;
Tz = 0;

% 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);

% Linearized EOM about phi=0, theta=0
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;
%}

% Non-linear equations of motion (fugly)
phi      = x(1); % LVLH frame
theta    = x(2); % LVLH frame
psi      = x(3); % LVLH frame
Wx       = x(4); % Body frame
Wy       = x(5); % Body frame
Wz       = x(6); % Body frame

alpha = pi/2 + psi; % angle b/w sun line and roll axis (P161)
F = -Fs*cos(alpha)^2;

% Gravity gradient torques
Rx = -sin(theta);
Ry = sin(phi)*cos(theta);
Rz = cos(phi)*cos(theta);

Gx = -3*n^2*(Jy-Jz)*Ry*Rz;

```

```

Gy = -3*n^2*(Jz-Jx)*Rz*Rx;
Gz = -3*n^2*(Jx-Jy)*Rx*Ry;
% Gx = 0;
% Gy = 0;
% Gz = 0;

% Wx = dWxdt;
% Wy = dWydt;
% Wz = dWzdt;
dWxdt = (1/Jx)*((Jy - Jz)*Wy*Wz + Gx + 0.5*epsilon*F + Tx);
dWydt = (1/Jy)*((Jz - Jx)*Wz*Wx + Gy - m/(M+2*m)*(z*F) + epsilon*F + Ty);
dWzdt = (1/Jz)*((Jx - Jy)*Wx*Wy + Gz - m/(M+2*m)*(y*F) + epsilon*F + Tz);

% Orientation of Spacecraft in LVLH angles
% psi points to center of earth
% phi points along direction of travel (transverse)
% theta is perpendicular to orbit plane
% phi = dphidt;
% theta = dthetadt;
% psi = dpsidt;
dphidt = (1/cos(theta))*(Wx*cos(theta) + Wy*sin(phi)*sin(theta) + Wz*cos(phi)*sin(theta))
+ (n/cos(theta))*sin(psi);
dthetadt = (1/cos(theta))*(Wx*0 + Wy*cos(phi)*cos(theta) - Wz*sin(phi)*cos(theta))
+ (n/cos(theta))*cos(theta)*cos(psi);
dpsidt = (1/cos(theta))*(Wx*0 + Wy*sin(phi) + Wz*cos(phi)
+ (n/cos(theta))*sin(theta)*sin(psi);

xdot(1) = dphidt;
xdot(2) = dthetadt;
xdot(3) = dpsidt;
xdot(4) = dWxdt;
xdot(5) = dWydt;
xdot(6) = dWzdt;

% Zero-thrust angle has psi = 0
% Full-thrust angle has psi = -90;

```

end

runT2Fuzzy.m

```
function u = runT2Fuzzy(e,dedt,inte ,gains)

    x1 = e;      % x1 is state error
    x2 = dedt;  % x2 is derivative of state error
    x3 = inte;  % x3 is integral of state error

    % Define MFs
    % define 11 gaussian MFs
    nMF = 9;
    C = linspace(-1,1,nMF);
    s = C(2)-C(1);
    %   sL = sU;
    %   kU = 1.0;
    %   kL = 0.8;
    xMFu = [-2,-s,-s/4,0,s/4,s,2]';
    yMFu = [0,0,1,1,1,0,0]';
    xMF1 = [-2,-s*3/4,0,s*3/4,2]';
    yMF1 = [0,0,2/4,0,0]';

    %   xMFU = [-2,-sU,0,sU,2];
    %   xMFL = [-2,-sL,0,sL,2];
    %   yMF = [0,0,1,0,0];

    X1U = ones(1,nMF);
    X1L = ones(1,nMF);
    X2U = ones(1,nMF);
    X2L = ones(1,nMF);
    X3U = ones(1,nMF);
    X3L = ones(1,nMF);

    % Triangular MFs
    for i = 1:nMF
        X1U(i) = interp1q(xMFu+C(i),yMFu,x1);
        X1L(i) = interp1q(xMF1+C(i),yMF1,x1);
        X2U(i) = interp1q(xMFu+C(i),yMFu,x2);
        X2L(i) = interp1q(xMF1+C(i),yMF1,x2);
        X3U(i) = interp1q(xMFu+C(i),yMFu,x3);
        X3L(i) = interp1q(xMF1+C(i),yMF1,x3);
    end

    % Gaussian MFs
    %   X1U = kU*exp((-0.5*(x1-C).^2)./sU.^2);
    %   X1L = kL*exp((-0.5*(x1-C).^2)./sL.^2);
    %
    %   X2U = kU*exp((-0.5*(x2-C).^2)./sU.^2);
    %   X2L = kL*exp((-0.5*(x2-C).^2)./sL.^2);
    %
    %   X3U = kU*exp((-0.5*(x3-C).^2)./sU.^2);
    %   X3L = kL*exp((-0.5*(x3-C).^2)./sL.^2);
```

```

%     dedtExp = eExp;
%     c0 = (nMF-1)/2;
%     temp = -c0:c0;
temp = linspace(-1,1,nMF);
temp_dedt = temp*gains(3);
temp_dedt(1) = -10;
temp_dedt(end) = 10;
temp_inte = temp*gains(2);
temp_inte(1) = -3;
temp_inte(end) = 3;
%     temp_e = (sign(temp).*abs(temp).^eExp);
%     temp_dedt = sign(temp).*abs(temp).^dedtExp;
%     temp_inte = sign(temp).*abs(temp).^eExp;

rule_mat = repmat((temp_dedt)'*ones(1,nMF) + ones(1,nMF)'*(temp*gains(1)), [1,1,nMF]) + permute(
    repmat(ones(1,nMF)'*(temp_inte), [1,1,nMF]), [3,1,2]);

%{
Rows correspond to e MFs, columns to de/dt MFs, entries are crisp
outputs

    ans =

    10     9     8     7     6     5     4     3     2     1     0
     9     8     7     6     5     4     3     2     1     0    -1
     8     7     6     5     4     3     2     1     0    -1    -2
     7     6     5     4     3     2     1     0    -1    -2    -3
     6     5     4     3     2     1     0    -1    -2    -3    -4
     5     4     3     2     1     0    -1    -2    -3    -4    -5
     4     3     2     1     0    -1    -2    -3    -4    -5    -6
     3     2     1     0    -1    -2    -3    -4    -5    -6    -7
     2     1     0    -1    -2    -3    -4    -5    -6    -7    -8
     1     0    -1    -2    -3    -4    -5    -6    -7    -8    -9
     0    -1    -2    -3    -4    -5    -6    -7    -8    -9   -10

%}

%calculate firing strength of each rule using min function
FU = min(min(repmat(X2U'*ones(1,nMF), [1,1,nMF]), repmat(ones(1,nMF)'*X1U, [1,1,nMF])), permute(
    repmat(ones(1,nMF)'*X3U, [1,1,nMF]), [3,1,2])); % Firing strength of upper MFs
FL = min(min(repmat(X2L'*ones(1,nMF), [1,1,nMF]), repmat(ones(1,nMF)'*X1L, [1,1,nMF])), permute(
    repmat(ones(1,nMF)'*X3L, [1,1,nMF]), [3,1,2])); % Firing strength of lower MFs
%     FL = min(X2L'*ones(1,nMF),ones(1,nMF)'*X1L); % Firing strength of lower MFs

u = sum(rule_mat(:).*(FU(:)+FL(:)))/sum(FU(:)+FL(:));
%     u = sum(rule_mat(:).*FU(:))/sum(FU(:));
end

```

F PSO-PID Controller

RUNME.m

```
%% Solar Sail Model Setup
close all
T = 0:10:30e3; % Set duration of simulation

% Specify initial conditions of Solar Sail craft
x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
x0 = x0*pi/180; % convert degrees to radians

% Specify desired final state of Solar Sail craft
xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dpsl] in degrees
xf = xf*pi/180; % convert degrees to radians

r_phi = xf(1);
r_theta = xf(2);
r_psi = xf(3);

% Particle swarm optimization algorithm applied to Solar Sail PID gains

nvars = 9;

lowerBound = zeros(nvars,1);
upperBound = ones(nvars,1)*1e7;

ZN = [0.000000600000000,... % Kp-Tx
      0.00000000260417,... % Ki-Tx
      0.000345600000000,... % Kd-Tx?
      0.003000000000000,... % Kp-z
      0.000000315656566,... % Ki-z
      7.128000000000000,... % Kd-z
      0.003000000000000,... % Kp-y
      0.000000289351852,... % Ki-y
      7.776000000000000]*1e4; % Kd-y

options = optimoptions('particleswarm',...
    'SwarmSize',100,...
    'InitialSwarmMatrix',ZN,...
    'InitialSwarmSpan',500,...
    'Display','iter',...
    'DisplayInterval',5,...
    'UseParallel',true,...
    'FunctionTolerance',.001,...
    'OutputFcn',@PSoutputDisplay);

tstart = tic;
```

```

[x, fval, exitflag] = particleswarm(@EvaluateCostFunction, nvars, lowerBound, upperBound, options);

disp('Kp_phi, Ki_phi, Kd_phi, Kp_theta, Ki_theta, Kd_theta, Kp_psi, Ki_psi, Kd_psi')
disp(x)

disp(sprintf('This calculation took %dh %dm %ds.', floor(toc(tstart)/3600), floor(toc(tstart)/60), mod(
    floor(toc(tstart)),60)));

%% Save Result
save('lastWORKSPACE.mat');

currentTime = clock;

fileID = fopen('RESULT.txt','w');
fprintf(fileID, '%4.f - %2.f - %2.f\n%2.f : %2.f : %2.f\n',currentTime);

fprintf(fileID, 'X:\n')
fprintf(fileID, 'Kp_phi, Ki_phi, Kd_phi, Kp_theta, Ki_theta, Kd_theta, Kp_psi, Ki_psi, Kd_psi\n')
fprintf(fileID, '%E, %E, %E, %E, %E, %E, %E, %E, %E, %E', x)
fprintf(fileID, '\n')

fprintf(fileID, 'fval:\n')
fprintf(fileID, '%E', fval)
fprintf(fileID, '\n')

fprintf(fileID, 'exitflag:\n')
fprintf(fileID, '%i', exitflag)
fprintf(fileID, '\n')

fclose(fileID);

%% Plot Result
close all
PlotResult([ZN;x])

```

EvaluateCostFunction.m

```
function [J] = EvaluateCostFunction(gains)
% gains = [0.000000600000000,... % Kp-Tx
% 0.000000000260417,... % Ki-Tx
% 0.000345600000000,... % Kd-Tx?
% 0.003000000000000,... % Kp-z
% 0.000000315656566,... % Ki-z
% 7.128000000000000,... % Kd-z
% 0.003000000000000,... % Kp-y
% 0.000000289351852,... % Ki-y
% 7.776000000000000]*1e4; % Kd-y
% Gains Kp_phi, Ki_phi, Kd_phi, Kp_theta, Ki_theta, Kd_theta, Kp_psi, Ki_psi, Kd_psi
    model = 'simSolarSail';
    load_system(model);

% global Kp_phi Ki_phi Kd_phi Kp_theta Ki_theta Kd_theta Kp_psi Ki_psi Kd_psi
% var = {'Kp_phi', 'Ki_phi', 'Kd_phi', 'Kp_theta', 'Ki_theta', 'Kd_theta', 'Kp_psi', 'Ki_psi', '
Kd_psi'};
% for i = 1:9
%     assignin('base',char(var(i)),gains(i));
%     set_param(model,char(var(i)),gains(i));
% end

% T_max=(0.5/20)*13.3*Fs*sin(Theta_max); % From "EOM" in "Solar Sail Dynamics" subsystem

%{
%=====
% 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-synchronous 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)

=====
%}

% See simulink model for the rest of the model parameters.

% global xf x0 T
% T = evalin('base','T');
% x0 = evalin('base','x0');
% xf = evalin('base','xf');
% lenT = length(T);

% T = 0:10:30e3; % Set duration of simulation
Tend = 45e3;
%
% % Specify initial conditions of Solar Sail craft
x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
x0 = x0*pi/180; % convert degrees to radians

% Specify desired final state of Solar Sail craft
xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
xf = xf*pi/180; % convert degrees to radians

% vars = {'r_phi','r_theta','r_psi'};
% for i=1:3
%     set_param(model,char(vars(i)),xf(i));
% end

% % Uncomment for variable step
% set_param(model,'SolverType','Variable-step');

```

```

%     set_param(model, 'RelTol', '2.84e-12');

% Uncomment for fixed-step
%     set_param(model, 'SolverType', 'Fixed-step');
set_param(model, 'FixedStep', num2str(10));

%     set_param(model, 'FixedStep', num2str(T(2)-T(1)));
%     set_param(model, 'T', T);
set_param(model, 'StopTime', num2str(Tend));
%     set_param(model, 'x0', x0);
set_param([model, '/x0'], 'Value', [' ', num2str(x0), ' ']);
set_param([model, '/xf'], 'Value', [' ', num2str(xf), ' ']);
set_param([model, '/gains'], 'Value', [' ', num2str(gains.*[ones(1,6), ones(1,3)*-1]), ' ']);
try
    [T,~,X,U] = sim('simSolarSail.slx');

    Y = squeeze(X)';
%     U = squeeze(U)';
    dT = T(2) - T(1);

%     stateCostFunctionWeight = [1, 1, 1, 0, 0, 0]./(abs(xf-x0)+ (abs(xf-x0)==0)); % weights
%     given to different states in the cost function
stateCostFunctionWeight = [1, 1, 1, 0, 0, 0];
energyCostFunctionWeight = 1e-3*[1, 1, 1];

J = 0;
for k = 1:size(Y,2)
    if k<=3
        J = J + 1/T(end) * trapz(T,T.*abs((Y(:,k)*0+xf(k)) - Y(:,k))) *
            stateCostFunctionWeight(k);
    else
        if max(Y(:,k))>(1*pi/180) %Enforce max angular late of 1 deg/sec
            J = J + 1e6;
        end
    end
end

for k = 1:3
    switch k
        case 1
            % Tx
            % Energy used is integral of Tx
            J = J + trapz(T,abs(U(:,k))) * energyCostFunctionWeight(k);
        otherwise
            % y and z
            % Energy used is the integral of the mass acceleration
            J = J + trapz(T(1:end-2),abs(diff(diff(U(:,k))/dT)/dT)) *
                energyCostFunctionWeight(k);
        end
    end
end

catch E
    J = 1e6;
end

```

end

PSoutputDisplay.m

```
function stop = PSoutputDisplay (optimValues , state)

stop = false; % This function does not stop the solver
nParticles = size (optimValues .swarm,1);
nplot = 2;
switch state
    case 'init'
        %         nplot = size (optimValues .swarm,2); % Number of dimensions

        %         for i = 1:nplot % Set up axes for plot
        %             subplot (nplot,1,i);
        %             subplot (nplot,1,1);
        %             tag = sprintf ('psoplotrange_var_%g',i); % Set a tag for the subplot
        %             semilogy (optimValues .iteration ,0,'-k', 'Tag',tag); % Log-scaled plot
        %             semilogy (optimValues .iteration ,0,'-', 'Tag',tag); % Log-scaled plot
        %             if i ==1
        %                 hold on
        %             end
        %             ylabel (num2str (i))
        %         end

        subplot (nplot,1,1);
        tag = sprintf ('psoplotrange_var_1'); % Set a tag for the subplot
        %             semilogy (optimValues .iteration ,0,'-k', 'Tag',tag); % Log-scaled plot
        semilogy (optimValues .iteration ,0,'-o', 'Tag',tag); % Log-scaled plot
        grid on
        ylabel ('Best f(x)');
        %             hold on

        subplot (nplot,1,2);
        tag = sprintf ('psoplotrange_var_99'); % Set a tag for the subplot
        semilogy (optimValues .iteration ,0,'-o', 'Tag',tag); % Log-scaled plot
        grid on
        ylabel ('Avg Time per Sim');
        %             hold on

        xlabel ('Iteration', 'interp', 'none'); % Iteration number at the bottom
        subplot (nplot,1,1) % Title at the top
        title ('Log range of particles by component')
        setappdata (gcf, 't0', tic); % Set up a timer to plot only when needed
        setappdata (gcf, 't1', tic);
        %             drawnow
    case 'iter'
        %         nplot = size (optimValues .swarm,2); % Number of dimensions
        %         for i = 1:nplot
        %             subplot (nplot,1,i);
        %             subplot (nplot,1,1);
        %             % Calculate the range of the particles at dimension i
        %             irange = max (optimValues .swarm (:,i)) - min (optimValues .swarm (:,i));
        %             tag = sprintf ('psoplotrange_var_%g',i);
```

```

%         plotHandle = findobj(get(gca,'Children'),'Tag',tag); % Get the subplot
%         xdata = plotHandle.XData; % Get the X data from the plot
%         newX = [xdata optimValues.iteration]; % Add the new iteration
%         plotHandle.XData = newX; % Put the X data into the plot
%         ydata = plotHandle.YData; % Get the Y data from the plot
%         newY = [ydata irange]; % Add the new value
%         plotHandle.YData = newY; % Put the Y data into the plot
%     end
%     subplot(nplot,1,i);
subplot(nplot,1,1);
% Calculate the range of the particles at dimension i
%     irange = max(max(optimValues.swarm(:,i)) - min(optimValues.swarm(:,i)));
tag = sprintf('psoplotrange_var-1');
plotHandle = findobj(get(gca,'Children'),'Tag',tag); % Get the subplot
xdata = plotHandle.XData; % Get the X data from the plot
newX = [xdata optimValues.iteration]; % Add the new iteration
plotHandle.XData = newX; % Put the X data into the plot
ydata = plotHandle.YData; % Get the Y data from the plot
newY = [ydata optimValues.bestfval]; % Add the new value
plotHandle.YData = newY; % Put the Y data into the plot

subplot(nplot,1,2);
tag = sprintf('psoplotrange_var-99');
plotHandle = findobj(get(gca,'Children'),'Tag',tag); % Get the subplot
xdata = plotHandle.XData; % Get the X data from the plot
newX = [xdata optimValues.iteration]; % Add the new iteration
plotHandle.XData = newX; % Put the X data into the plot
ydata = plotHandle.YData; % Get the Y data from the plot
newY = [ydata toc(getappdata(gcf,'t1'))/nParticles]; % Add the new value
plotHandle.YData = newY; % Put the Y data into the plot
setappdata(gcf,'t1',tic);

if toc(getappdata(gcf,'t0')) > 1% If 1 s has passed
    drawnow % Show the plot
    setappdata(gcf,'t0',tic); % Reset the timer
end
case 'done'
    % No cleanup necessary
end

```

PlotResult.m

```
function [] = PlotResult(gains)
%     gains = [ZN;x];
nLines = size(gains,1);
rrows = 3;
ccols = 2;
names_X = {'\phi [deg]', '\theta [deg]', '\psi [deg]', 'd\phi/dt [deg/s]', 'd\theta/dt [deg/s]', 'd\psi/dt [deg/s]'};
names_U = {'T_x [N-m]', 'y [m]', 'z [m]'};
ulimit_U = [.5e-3,28,28];

model = 'simSolarSail';
load_system(model);

h1 = figure();
h2 = figure();
for ii=1:nLines
%     global Kp_phi Ki_phi Kd_phi Kp_theta Ki_theta Kd_theta Kp_psi Ki_psi Kd_psi
%     var = {'Kp_phi', 'Ki_phi', 'Kd_phi', 'Kp_theta', 'Ki_theta', 'Kd_theta', 'Kp_psi', 'Ki_psi', 'Kd_psi'};
%     for i = 1:9
%         assignin('base',char(var(i)),gains(ii,i));
%     end

%     global xf x0 T
T = evalin('base','T');
Tend = T(end);
x0 = evalin('base','x0');
xf = evalin('base','xf');
%     lenT = length(T);

%     T = 0:10:30e3; % Set duration of simulation
%
%     % Specify initial conditions of Solar Sail craft
%     x0 = [5, -5, -90, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
%     x0 = x0*pi/180; % convert degrees to radians
%
%     % Specify desired final state of Solar Sail craft
%     xf = [0, 0, -55, 0, 0, 0]; % [phi theta psi dphi ptheta dps] in degrees
%     xf = xf*pi/180; % convert degrees to radians

%     sim('simSolarSail.slx');
%
%     Y = SimOutput.Data(:, :);

%     set_param(model, 'FixedStep', num2str(T(2)-T(1)));
%     set_param(model, 'T', T);
set_param(model, 'StopTime', num2str(Tend));
%     set_param(model, 'x0', x0);
set_param([model, '/x0'], 'Value', ['[', num2str(x0), ']']);
```

```

set_param([model, '/xf'], 'Value', [' ', num2str(xf), ' ']);
set_param([model, '/gains'], 'Value', [' ', num2str(gains(ii, :)) * [ones(1,6), ones(1,3)*-1], ' ']);
[T,~,X,U] = sim('simSolarSail.slx');

Y = squeeze(X)';

%         stateCostFunctionWeight = [1, 1, 1, .1, .1, .1]./(abs(xf-x0)+ (abs(xf-x0)==0)); % weights
%         given to different states in the cost function
%
%         J = 0;
%         for k = 1:size(Y,3)
%             J = J + 1/T(end) * trapz(T,T'.*abs((Y(:,k)*0+xf(k)) - Y(:,k))) *
stateCostFunctionWeight(k);
%         end
figure(h1)
for j = 1:ccols;
    for k = 1:rrows;
        output = (j-1)*rrows+k;
        subplot(rrows,ccols,(k-1)*ccols+j)
        switch ii
            case 1
                plot(T,180/pi*Y(:,output), '--k')
                hold on
                grid on
                axis tight
%            case nLines
                plot(T,180/pi*Y(:,output))
                plot([T(1),T(end)], [1,1]*xf(output)*180/pi, ':k')
            otherwise
                plot(T,180/pi*Y(:,output), '-k', Linewidth, 2)
        end
        ylabel(names_X(output));
        if (k-1)*ccols+j>=5
            xlabel('Time [s]')
        end
    end
end
if j == 1:ccols
    legend('ZN Gains', 'PSO-PID Gains', 'Location', 'Best')
end
end

figure(h2)
for k = 1:3

    output = k;
    subplot(3,1,output)
    switch ii
        case 1
            plot(T,U(:,output), '--k')
            hold on
            grid on
%            axis tight
        case nLines
            plot(T,U(:,output))
    end
end

```

```

%                               plot([T(1),T(end)],[1,1]*xf(output)*180/pi,'--k')
    plot([T(1),T(end)],[1,1]*ulimit_U(k),'k')
    plot([T(1),T(end)],[-1,-1]*ulimit_U(k),'k')
    otherwise
        plot(T,U(:,output))
    end
    ylabel(names_U(output));
    if k>=3
        xlabel('Time [s]')
    end
end
end
if j == 1:ccols
    legend('ZN Gains','PSO-PID Gains','Location','Best')
end
end

end

end

%{
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot time responses

iLast = iLast - 1;
figure()
% ax1 = axes();

%       ax1 = subplot(2,3,1); % plot phi
if iLast-rstLenT+1 > 0
    [~,minCostIdx] = min(J(iLast-rstLenT+1:iLast));
    minCostIdx = minCostIdx + iLast-rstLenT;
else
    [~,minCostIdx] = min(J(1:iLast));
end
end
if iLast>5
%       plots = round([1,iLast/10,iLast/4,iLast/2,minCostIdx]);
    plots = round([1,minCostIdx]);
else
    plots=1:iLast;
end
end
;
%       plots = round([1,N/3,indBest]);

lineNames = cell(length(plots)+1,1);

stabTimeLo = 0;
stabTimeHi = 0;
for i = 1:length(plots)

%       lineNames{2*i-1} = sprintf('N=%0.f actual',plots(i));
    lineNames{i+1} = sprintf('N=%0.f',plots(i));
end
end

```



```
% xlabel(ax1, 'Time (s)')
if N>1
    legend(lineNames, 'location', 'SouthEast');
end
end
%}
```

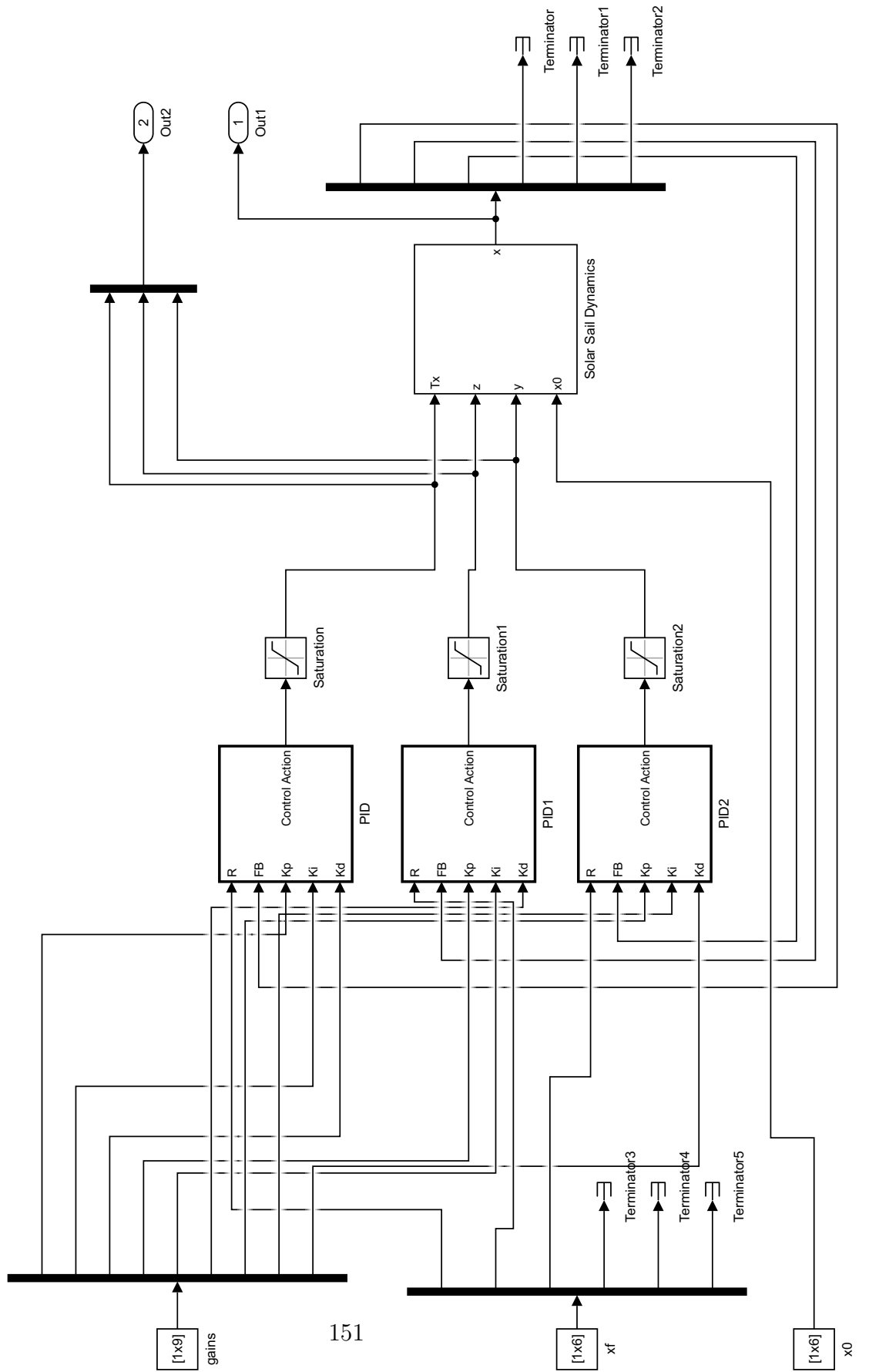


Fig. 6.15: Simulink model, PSO-PID controller, base.

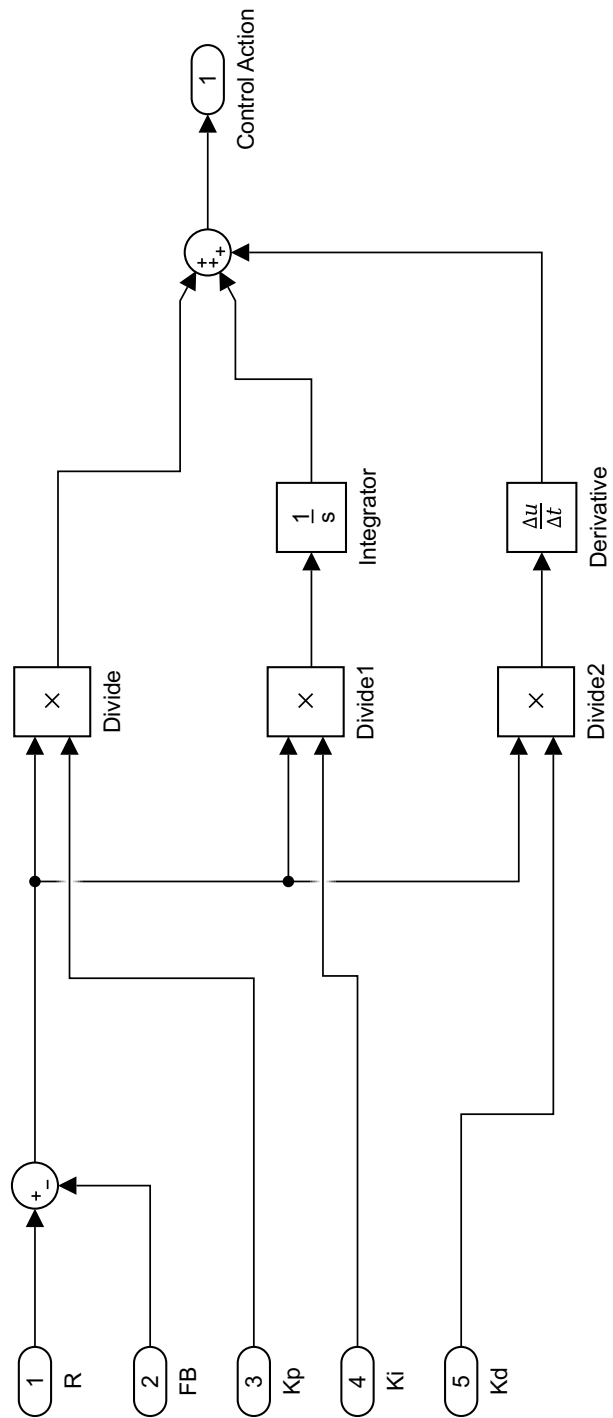


Fig. 6.16: Simulink model, PSO-PID controller, *PID Controller*.

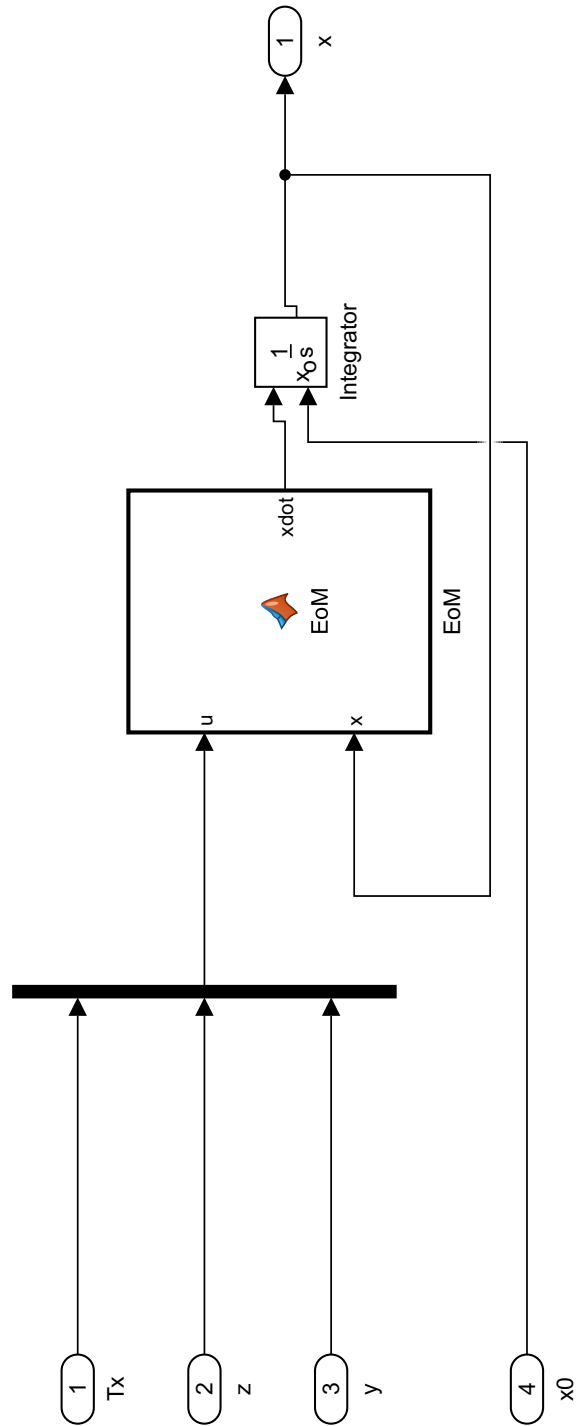


Fig. 6.17: Simulink model, PSO-PID controller, *Solar Sail Dynamics*.

EoM.m

```
function xdot = EoM(u,x)
%=====
% 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-synchronous 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
%=====
```

```

% Defining states and inputs
Tx = u(1);
z = u(2);
y = u(3);
Ty = 0;
Tz = 0;

% Calculating the J's
Jx = Ix+mr*(y(1)^2+z(1)^2);
Jy = Iy+mr*z(1)^2;
Jz = Iz+mr*y(1)^2;

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

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]';
end

```