ABSTRACT

Title of dissertation: SIMULTANEOUS ATTITUDE AND MANIPULATOR CONTROL FOR DETUMBLING COUPLED SATELLITES WITH APPENDAGES

Barrett E. Dillow, Doctor of Philosophy, 2019

Dissertation directed by: Professor David L. Akin
Department of Aerespace Engineering

Satellite servicing is of increasing interest in industry. Thousands of satellites, functional and not, are tumbling and would likely need to have their attitude stabilized for servicing or orbital change maneuvers. Further, a well-studied method for achieving capture between a servicer and a client satellite is with a robotic manipulator. This research presents a set of algorithms to achieve attitude stabilisation while reducing client appendage motion. A nonlinear quaternion feedback controller is presented; its stability proven and its utility discussed. A method of client appendage mode motion is presented. Finally a manipulator control algorithm to reduce client appendage motion is presented. These methods can be employed to increase the potential number of non-cooperative serviceable satellites.

SIMULTANEOUS ATTITUDE AND MANIPULATOR CONTROL FOR DETUMBLING COUPLED SATELLITES WITH APPENDAGES

by

Barrett E. Dillow

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Dedication

To Anna, Callum, and Emmeline:

Thank you for your unending patience and support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACS Attitude Control System

ARMCTRL Arm Control

AVGS Advanced Video Guidance Sensor

cmd Commanded

CMG Control Moment Gyro

CoM Center of Mass

ConOps Concept of Operations

CSA Canadian Space Agency

DAE Differential Algebraic Equation

DARPA Defense Advance Research Projects Agency

DCM Direction Cosine Matrix

des Desired

DH Denavit-Hartenberg

DOF Degree of Freedom

DRT Discrete Real Time

ECI Earth-Centered Inertial

EF Exposed Facility

est Estimated

ETS Engineering Test Satellite

FFT Fast Fourier Transform

FMA Force-Moment Accommodation

FPGA Field Programmable Gate Array

FREND Front-end Robotics Enabling Near-term Demonstration

FRGF	Flight Releasable Grapple Fixture
FTS	force-torque sensor
GEO	Geosynchronous Earth Orbit
GNC	Guidance, Navigation and Control
GOES	Geostationary Operational Environmental Satellite
GRAV	Gravity
GSFC	Goddard Space Flight Center
GSL	GNU Scientific Library
HIL	Hardware In the Loop
HST	Hubble Space Telescope
HTV	H-IIa Transfer Vehicle
ISS	International Space Station
JAXA	Japanese Aerospace Exploration Agency
JEM	Japanese Experimental Module
JEMRMS	Japanese Experimental Module Remote Manipulator System
LEO	Low Earth Orbit
LVLH	Local Vertical Local Horizontal
MANIP	Manipulator
MDA	MacDonald, Dettwiler and Associates
MER	Mars Exploration Rover
MEV	Mission Extension Vehicle
MILP	Mixed-Integer Linear Programming
MMS	Magnetosphere Multiscale Mission
MSFC	Marshall Space Flight Center

NASA National Aeronautics and Space Administration

NASDA National Space Development Agency of Japan

NIMA National Imagery and Mapping Agency

PD Proportional-Derivative

PID Proportional-Integral-Derivative

PROP Propulsion

react Reaction or Reactive Control response

RLS Recursive Least Squares

RPM Revolutions Per Minute

RPO Rendezvous & Proximity Operations

RMS Remote Manipulator System

RNS Relative Navigation System

RxNS Reactionless Null-Space

RSGS Robotic Servicing of Geosynchronous Satellites

RSS Root Sum Squared

RT Real Time

RWA Reaction Wheel Assembly

SAA South Atlantic Anomaly

SADA Solar Array Drive Actuator

SFA Small Fine Arm

SM4 Servicing Mission 4

SOLSYS Solar System

SRP Solar Radiative Pressure

SSPD Satellite Servicing Projects Directorate

STS	Shuttle Transport System
VSD	Velocity State Dynamics

Chapter 1: Introduction

This dissertation investigates methods for detumbling a satellite prior to servicing. The motivations will be discussed after some basic definitions are given; then the specific methods proposed will be discussed in detail. In satellite servicing, there is a client satellite receiving various upgrades or alterations and a servicing satellite providing those upgrades. These will be referred to in this document as the servicer (that which provides the upgrades) and the client (that which receives the upgrades). In this dissertation, the post-grapple phase of a satellite servicing mission is considered, with the client potentially tumbling. Controllers for satellite attitude and a robotic manipulator will be presented for use in detumbling of a system of two satellites coupled through a robotic manipulator, in the case where the system has a non-zero inertial attitude rate. A method of vibration detection and reduction in this system will also be presented such that modal excitation of a client can be reduced during a detumbling maneuver. The progression of this dissertation is as follows:

1. This chapter presents the motivation for satellite servicing and explains the focus on the post-grapple phase, as well as the emphasis on satellite pairs coupled through robotic manipulators.

2. The second chapter will provide background on some previously proposed nonlinear attitude controllers; introduce a novel nonlinear attitude controller; show proof of its stability, and; provide cases in which it has improved performance in comparison to other controllers. (Contribution 1)

3. The third chapter provides a review of existing manipulator controllers; identifies how they are deficient for the use case presented, and; presents a novel manipulator controller for reduction of system vibration. (Contribution 2)

4. The fourth chapter discusses the impact of the necessary sliding-window Fast Fourier Transform (FFT) parameters and presents a novel method of automating the detection of dominant peaks within its output in the presence of noise, which is a necessary step for the realization of Contribution 2. (Contribution 3)

5. The fifth chapter presents a detailed description of the combined system in a 6-Degree of Freedom (DOF) orbital simulation and presents the results of several experiments.

6. The sixth and final chapter will summarize the technical contributions of this dissertation and include suggestions for further development of this area in future research.

1.1 Motivation

Humans have been launching satellites for over sixty years. The satellites they have launched often had flexible appendages such as solar arrays for power, or long antennae for communications or scientific measurements. Sputnik, the first artificial satellite of Earth, launched in 1957, had four relatively large antenna (2.4 - 2.9 meters), compared to its body size (a 0.58 meter sphere). [1] According to the Union of Concerned Scientists, as of 30 April 2018, there were 1,886 active satellites in orbit. [2]

The servicing of satellites while on-orbit is a topic of increasing interest. Servicing is an umbrella phrase that can cover many different activities, including

- orbit modification, e.g.
  - lower to de-orbiting,
  - raise to return to desired operational altitude or a graveyard orbit, or
  - continuous maintainenance, that is, addition of a new propulsion system that remains attached;
- replenishment of satellite fuels (refueling); or,
- complete replacement of major components.

The vast majority of satellites are designed to be expendable, disposed of after component failure or fuel depletion, as discussed by Reedman et al. [3] in discussion of requirements for future servicable satellites. One prime example of component replacement on orbit is the Hubble Space Telescope (HST), which was designed from its inception to be servicable in orbit by human crews. Some of the lessons learned from the first three servicing missions to HST are discussed by Werneth [4]. Long et al. [5] illustrate how the business case for component replacement and satellite augmentation for Geosynchronous communications satellites already exists. Joppin and Hastings [6] investigate how the addition of servicing can add flexibility and value to science missions. Long and Joppin both use the lessons learned from the HST servicing to inform their analyses. Naasz et al. [7] show the business case for refueling of geosynchronous satellites, in adition to repair or inspection. It is possible that a fully-functional satellite is moved to a graveyard orbit simply because it is running low on the fuel needed to maintain its position within its assigned orbital slot, or a failure in its primary propuslion system as was the case with PanAmSat's Galazy 8i [3]. Satellites such as this could be returned to useful service with additional fuel or replacement propulsion systems. Significant savings could be achieved if repair and refueling can be performed on orbit as long as launch costs remain one of the largest portions of total mission cost, as shown by Sullivan et al. in [8]. Satellites that have had other failures may be in desirable orbits and their disposal would provide the benefit of decreasing space debris and returning their orbits to productive users. Disposal can take the form of de-orbiting the spacecraft to burn up in the atmosphere or performing a maneuver to place it in a so-called graveyard orbit. Ellery et al. [9] study the benefits of this disposal capability in the geosynchronous satellite market.

The business case has been made so well that two commercial ventures are contracted for some form of on-orbit servicing, as of 2018. Orbital ATK (now Northrup Grumman Innovation Systems) has announced it is now taking orders for its Mission Extension Vehicle

(MEV), which will attach to a client satellite in Geosynchronous Earth Orbit (GEO) to assume all attitude and stationkeeping functions for clients that are low on fuel. [10] Effective Space, a UK company, has also announced contracts to provide its own "Space Drone" system for similar propulsion assistance to communications satellites. [11] In both cases, the client satellites will be operational at the time of rendezvous meaning the orbit, attitude and body rates of the client can be well controlled prior to capture. Also, as GEO communications satellites have a preferred nominal orientation to point their large antennae and reflectors towards Earth's surface, their nominal attitude and body rates are well behaved, which eases the rendezvous process.

1.1.1 Clients may be tumbling

Within this work, the term "tumbling" is used to indicate the state of an uncontrolled multi-axis attitude rate. There are several reasons a potential client may be tumbling. Some satellites are spin-stabilized design but are not de-spun at end of life once they become uncontrolled. An analysis by Kaplan et al. [12] determined there were over 100 GEO objects that were spin stabilized and could still have spin rates as high as 40 Revolutions Per Minute (RPM) (240 deg/sec) as of 2010. Other satellites may be tumbling due to internal failures such as fuel depletion, or actuator failures. If the spacecraft's attitude cannot be controlled, then tumbling can be induced by the space environment. Even at geostationary altitudes, atmospheric drag, gravity gradient and Solar Radiative Pressure (SRP) can induce a tumble in an uncontrolled satellite. [13] Tumbling can be induced by conjunction events, i.e. collisions with space debris or derelict spacecraft. Additionally, energetic failure modes can lead to a lack of Attitude Control System (ACS) control authority such as shortages of battery circuits leading to explosions, or fuel tank ruptures. One example of an energetic failure inducing a tumble is the Japanese Aerospace Exploration Agency (JAXA) X-Ray observatory Hitomi (also know as ASTRO-H). It mis-fired its attitude control system until it spun fast enough to rip itself apart after a series of errors initiated as it flew through the South Atlantic Anomaly (SAA). [14] Early optical observations of the satellite, post-breakup, estimated it rotating once every 5.2 seconds, or 69.2 deg/sec. The spacecraft broke apart into several pieces, ten of which are still large enough to track and are still in orbit, according the tracking website Heavens-above.com. Those pieces, too, are likely to have substantial body rates.

One example of naturally induced tumbling is the communications satellite KOREASAT-1. It completed its useful mission and was raised from its initial geostationary orbit to a graveyard orbit 200 km above in 2005. When its spin rate was estimated with optical measurements in 2013, natural perturbations had increased its rate to 4.8 deg/sec. [15] Finally, tumbling can be caused by forces exerted on the client by the servicing vehicle during the grapple procedure. The ability to service tumbling satellites increases the pool of potential customers, further strengthening the business case for satellite servicing. A tumbling spacecraft presents several challenges to servicing, one of which, the post-grapple detumbling procedure, will be addressed in this research.

1.1.2 Rendezvous

The controlled arrival of one spacecraft within the vicinity of another spacecraft can most generally be described as a rendezvous.[1] This research is focusing on detumbling of clients. Detumbling methods can be classified into two categories: those requiring contact and those that do not. There has been interesting research by Sugai, et al. on the use of electrical eddy currents to affect a change in spin rate of a client spacecraft without contact [16], [17]. That avenue remains an area of active research, but the vast majority of other detumbling methods require contact. Rendezvous methods that involve contact can be generally classified as either a docking or a grapple. A docking involves the secure mating of two spacecraft via a pre-designed interface for the transfer of fluids (e.g. fuel, atmosphere) or people and other solid cargo. A grapple involves the secure mating of two spacecraft via an active component on the servicer side and a passive component on the client/receiving side, but not necessarily a pre-designed interface for in-space contact on the client side. A docking requires two vehicles to have been designed with this interfacing in mind, and therefore is classified as a cooperative rendezvous. The rendezvous of the Soyuz and Progress Russian spacecraft with Mir and International Space Station (ISS), as well as the docking of the

[1]As oppossed to an uncontrolled rendezvous, potentially involving the impact of the two spacecraft in question, which would be best described as a conjunction event.

Space Shuttle with the ISS are both examples of cooperative docking. A grappling event may be achieved either with a cooperative or uncooperative client.

Past missions that were designed to grapple other free-flying components in microgravity, e.g. Shuttle Remote Manipulator System (RMS), ISS RMS, Engineering Test Satellite (ETS)-VII, and Orbital Express, were operated with flight rules or Concept of Operations (ConOps) that required zero or very low relative linear velocity and attitude rates between the client and the grasping spacecraft. They also required clients to have an active ACS that is able to maintain attitude until the grapple is to be attempted, immediately before which time the client ACS would be put into free drift to avoid the client ACS from fighting the servicer manipulator and servicer ACS. [18] The presence of an active ACS system ensures desirable relative rates, alleviating potential problems caused by excessive tumbling (and subsequent detumbling maneuvers), but restricts the pool of possible clients for servicing.

1.2 Robotic manipulators for operational adaptability

A spacecraft is described as "non-cooperative" if it has been designed without aids for rendezvous and docking while on-orbit. Robotic manipulators are commonly proposed mechanisms for grappling of cooperative and non-cooperative satellites. [19] Robotic manipulators have been flown on many missions in the past and several are currently in orbit today. A very thorough review was published by Flores-Abad in 2014 [20], several relevant devices and missions are discussed below, supporting the design of the manipulator proposed within this work, and providing necessary contrast to the focus of this research.

To motivate the selection of manipulator kinematics and general sizing chosen for this research, as well as the motivation for the proposed control algorithm, several previous and proposed space servicing manipulators will be discussed in Chapter 3.

1.3 Current Methods Of Modal Avoidance

The methods presented below each address pieces of the overall problem, but not the whole, and present areas to begin the investigation. Initial methods of mitigating susceptibility of satellite appendages to vibration included designing the structure of the overall system to be stiff enough and strong enough to survive and damp out any disturbances that might occur [21], with subsequent methods focused on limiting the excitation of primary modes, either by avoiding them or actively reacting to them.

1.3.1 Structural Filters

A common method of avoiding vibrational modes or undesirable frequencies in actuation commands are structural filters, e.g. low-pass, band-pass, high-pass and notch filters. Low-pass filters only allow frequencies of signal below a specified frequency to be executed; band-pass filters only allow frequencies between two defined frequencies to be executed; high-pass filters only allow frequencies above a specified frequency to be actuated; and, finally, notch filters, only allow frequencies *outside* a certain frequency range to be actuated. A priori knowledge of sensitive structural modes are used to design structural filters. An ACS command is passed through the filter and modified such that the output set of commands will have less content in the frequencies of concern. It is that modified command that is then passed to the actuators. For example, notch filters were successfully employed on the Space Shuttle to avoid exciting modes of its payloads and of vehicles to which it was docked. [22]

1.3.2 Piezo-electric Actuators

Whereas structural filters, above are designed to limit actuation within a desired frequency range, piezo-electric actuators can be thought of as the opposite. Azadi [23], Gennaro [24] [25], Hu [26] [27], Meyer [28], Oh [29], Sabatini [30], Singhose [31], Song [32] [33], and Zarafshan [34], among others, present different ways that piezo-electric actuators can be used to detect and dampen vibration in flexible structures. Their biggest limitation, as they relate to this research, is that the devices must be built into the structure which one wishes to detect or dampen the vibrations. The focus of this research is the reduction vibrations in uncooperative clients by servicing vehicles during post-grapple detumbling maneuvers, therefore modifications to the client structure are not possible prior to a successful detumble which would precede servicing. Therefore this form of actuation will not be considered for this research.

Satellite actuation can be performed on satellites with a variety of devices. Thrusters provide forces and torques. Reaction wheels, Control Moment Gyros (CMGs) and magnetic torquer bars [35], all provide only torque. Robotic manipulators also provide torque to the base body, or forces if there is another body to react against.

To limit the scope of this research, only methods that can perform attitude control of the detumble with thruster-based attitude control are investigated. This allows for larger torques to be actuated, limiting the time necessary for given detumbling procedures.

1.4 Conclusion

The motivation for this work and a variety of partial solutions has been presented. In the next chapter, a nonlinear attitude controller for detumbling will be presented, its stability will be proven and its benefit in contrast to some other nonlinear controllers will be presented. Subsequently, manipulator control methods that have been previously presented will be discussed and how they fail to meet the design requirements for this problem will be discussed. Then a manipulator control strategy for disturbance rejection will be presented. The details of disturbance detection using the force-torque sensor (FTS) will be presented in Chapter 4. The details of the 6-DOF orbital simulation will be presented in Chapter 5 along with the results of several examples with the combined attitude controller, disturbance detection and manipulator controller working in unison. Finally, in Chapter 6, the contributions will be summarized and future work will be proposed.

Chapter 2: Nonlinear Quaternion Feedback Controller

2.1 Introduction

In this chapter, a nonlinear quaternion feedback controller for attitude regulation, given an error quaternion, $\mathbf{q}_{err} = [\boldsymbol{\epsilon}^T, \eta]^T$ and rate error, $\boldsymbol{\omega}$, Eq. (2.1), will be presented and a proof of its stability will be given.

$$\mathbf{u} = -\text{sign}\left(\eta\right)\mathbf{K}\boldsymbol{\epsilon} - \left(1 - \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\mathbf{D}\boldsymbol{\omega} \tag{2.1}$$

$$\text{sign}\left(\eta\right) = \begin{cases} 1 & \text{for } \eta \ \geq \ 0 \\ -1 & \text{for } \eta \ < \ 0 \end{cases} \tag{2.2}$$

This controller uses matrices, $\mathbf{K}$ and $\mathbf{D}$, to scale each axis of the gains separately based on the spacecraft inertia. The following stability proofs will show first that it is stable and furhter that its stability does not require perfect knowledge of the inertia of the spacecraft. The importance of stability in the presence of inertia knowledge in a system of a servicer satellite coupled with a client satellite or piece of orbital debris is that the inertia of the couple load may not be well known. Indeed, in the case of orbital debris such as that created by the disintegration of Hitomi X-ray observatory, as discussed earlier, large pieces of debris were created with high tumble rates. [14] If these pieces must be detumbled for deorbit, it would be very difficult, a priori, to attain a reliable estimate of debris inertias.[1] The cases in which this controller is superior to a scalar gain case will also be presented below.

2.1.1 Notation, Identities, Dynamics

Let the attitude of a spacecraft be represented by a rotational axis unit vector, $\mathbf{e}$, an Euler Axis, and a rotation about that axis, ϕ, then

$$\mathbf{q} = \begin{bmatrix} \mathbf{e}\sin\left(\frac{\phi}{2}\right) \\ \\ \cos\left(\frac{\phi}{2}\right) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon} \\ \\ \eta \end{bmatrix}$$

where $\boldsymbol{\epsilon} = [\epsilon_x, \epsilon_y, \epsilon_z]^T$. It should be noted that $\|\mathbf{q}\| = 1$, i.e. $\epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 + \eta^2 = 1$. In other words, a quaternion is of unit length. Let $\boldsymbol{q}_{{}^b/_i}$ (read as "cue eye to bee") be the quaternion representation of the current actual spacecraft body coordinate frame, 'b', with respect to the the inertial reference frame, 'i'. Let $\boldsymbol{q}_{{}^d/_i}$ be the quaternion representation of the desired spacecraft body coordinate frame, 'd', with respect to the inertial reference frame. Then, the error quaternion from the current body frame to the desired body frame can be computed by $\boldsymbol{q}_e = \boldsymbol{q}_{{}^d/_b} = \boldsymbol{q}_{{}^d/_i} \otimes \mathbf{q}_{{}^b/_i}^{-1}$, where $\otimes$ is the quaternion multiplication operation, which will

[1]This case would represent one of least cooperative of uncooperative clients.

be defined below. Let $\mathbf{q}_a = \left[\boldsymbol{\epsilon}_a^T, \eta_a\right]^T$ and $\mathbf{q}_b = \left[\boldsymbol{\epsilon}_b^T, \eta_b\right]^T$ be two arbitrary unit quaternions. Then the product of their multiplication is

$$\mathbf{q}_c = \mathbf{q}_a \otimes \mathbf{q}_b = \begin{bmatrix} \eta_b \boldsymbol{\epsilon}_a + \eta_a \boldsymbol{\epsilon}_b - \boldsymbol{\epsilon}_a \times \boldsymbol{\epsilon}_b \\ \eta_a \eta_b - \boldsymbol{\epsilon}_a^T \boldsymbol{\epsilon}_b \end{bmatrix}.$$

Other identies of use are

$$\begin{aligned} -\mathbf{q} &= \mathbf{q} \\ \mathbf{q}^{-1} &= \left[\, -\boldsymbol{\epsilon}^T,\ \eta \,\right]^T = \left[\, \boldsymbol{\epsilon}^T,\ -\eta \,\right]^T \\ \mathbf{q} \otimes \mathbf{q}^{-1} &= [\, 0,\ 0,\ 0,\ \pm 1 \,]^T . \end{aligned}$$

Another useful definition is the cross-product matrix operator, $[\cdot]^\times$. Given a vector $\mathbf{v} = [\, v_x,\ v_y,\ v_z \,]^T$ and $\mathbf{u} = [\, u_x,\ u_y,\ u_z \,]^T$,

$$\mathbf{v}^\times = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix},$$

such that $[\mathbf{v}^\times]\, \mathbf{u} = \mathbf{v} \times \mathbf{u}$.

Given a system with inertia tensor $\mathbf{J}$ which is about the center of mass and evaluated in the system's body frame; an attitude $\mathbf{q} = \left[\boldsymbol{\epsilon}^T, \eta\right]^T$ which is evaluated from a reference frame (inertial space, for example), to the same system body frame; a body attitude rate $\boldsymbol{\omega}$, as measured from the reference frame to the body frame; and a torque input $\mathbf{u}$, all expressed in the spacecraft's body frame, then the dynamics of the system are described by Eqs. (2.3), (2.4), and (2.5) below.

$$\dot{\boldsymbol{\epsilon}} = \frac{1}{2}\boldsymbol{\epsilon}^{\times}\boldsymbol{\omega} + \frac{1}{2}\eta\boldsymbol{\omega} \tag{2.3}$$

$$\dot{\eta} = -\frac{1}{2}\boldsymbol{\epsilon}^{T}\boldsymbol{\omega} \tag{2.4}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}\left[\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega} + \mathbf{u}\right] \tag{2.5}$$

2.1.2 Prior Work

Several previous quaternion feedback controllers are presented below for comparison. Wie and Barba in [36] proposed three quaternion feedback laws:

$$\mathbf{u} = -T_c k\boldsymbol{\epsilon} - T_c\mathbf{D}\boldsymbol{\omega} \tag{2.6}$$

$$\mathbf{u} = -\frac{T_c k}{\eta^3}\boldsymbol{\epsilon} - T_c\mathbf{D}\boldsymbol{\omega} \tag{2.7}$$

$$\mathbf{u} = -T_c \text{sign}\left(\eta\right) k\boldsymbol{\epsilon} - T_c\mathbf{D}\boldsymbol{\omega} \tag{2.8}$$

where $\mathbf{u}$ is the commanded control torque vector; T_c is the positive scalar control torque level of the reaction jets; k is a scalar positive gain; $\mathbf{D} = [\ k_1,\ 0,\ 0;\ 0,\ k_2,\ 0;\ 0,\ 0,\ k_3\ ] > 0$ is a positive definite diagonal gain matrix; $\mathbf{q} = \left[\ \boldsymbol{\epsilon}^T,\ \eta\ \right]^T$ is the attitude error quaterion; and,

$\boldsymbol{\omega}$ is the spacecraft inertial rate. They note, however, that when the initial attitude error is near 180^o, the quaternion scalar, η is near zero resulting in nearly infinite control torque commanded in (2.7). They also note that for $\eta > 0$, Equations (2.6) and (2.8) are identical, but when $\eta < 0$, Eq. (2.8) ensures the shortest path to reorientation.

Subsequently, Wie proposed with Weiss and Arapostathis [37] the following quaternion feedback controller:

$$\mathbf{u} = -\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega} - \mathbf{D}\boldsymbol{\omega} - \mathbf{K}\mathbf{q}_e$$

where $\mathbf{D}$ and $\mathbf{K}$ are 3×3 constant gain matrices. They note that the inclusion of the gyroscopic component increases performance with high velocity maneuvers.

Wen and Kreutz-Delgado proposed [38], and later Joshi, Kelkar and Wen [39] proved the stability of a different quaternion feedback for attitude stabilization:

$$\mathbf{u} = -\frac{1}{2}\left[\left(\boldsymbol{\epsilon}^{\times} + \eta\mathbf{I}\right)\mathbf{G}_p + \gamma\left(1-\eta\right)\mathbf{I}\right]\boldsymbol{\epsilon} - \mathbf{G}_r\boldsymbol{\omega},$$

where $\mathbf{G}_p$ and $\mathbf{G}_r$ are symmetric positive definite (3×3) matrices, γ is a positive scalar and $\mathbf{I}$ is the (3×3) identity matrix.

Later still, Markley and Crassidis propose in [40] (as Eq. (7.14)) a shortest-distance control law:

$$\mathbf{u} = -k_p \mathrm{sign}\left(\eta\right)\boldsymbol{\epsilon} - k_d\left(1 \pm \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\boldsymbol{\omega}, \tag{2.9}$$

where $\boldsymbol{\epsilon}$ and η are the vector and scalar components, respectively, of the attitude error quaternion; k_p and k_d are positive scalar gains; and $\boldsymbol{\omega}$ is the spacecraft inertial body rate.

2.2 Quaternion Feedback Attitude Controller Stability

2.2.1 Static Inertia, Perfect Knowledge

The new quaternion feedback controller presented here is similar to that of Markley [40], presented as Eq. (2.9) above, except the position and rate gains are positive definite symmetric matrices, not scalars. In this subsection, its stability will be proven in the case of a static moment of inertia with perfect knowledge. Note that the stability proof below also holds for gains that are positive definite diagonal matrices. The attitude control torque is defined by Eq. (2.1), introduced above. The gains will be defined as functions of the spacecraft inertia, controller scalar response frequency, $\omega_c > 0$, and controller scalar response damping ratio, $\zeta_c > 0$. See Eqs. (2.10) and (2.11), below.

$$\mathbf{K} = \omega_c^2 \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix} \tag{2.10}$$

$$\mathbf{D} = 2\zeta_c\omega_c \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix} \tag{2.11}$$

The stability of this controller will be proven via Lyapunov analysis. The Lyapunov candidate function, Eq. (2.12) below, is similar to that proposed by Wie and others in [37], but modified to be piece-wise, similiar to Thienel and Sanner in [41] to handle both positive and negative signs of η.

$$V = \frac{1}{2}\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{J}\boldsymbol{\omega} + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 + \begin{cases} (\eta - 1)^2 \text{ for } \eta \geq 0 \\ (\eta + 1)^2 \text{ for } \eta < 0 \end{cases} \tag{2.12}$$

$\mathbf{K}^{-1}$ exists because $\mathbf{K}$ is symmetric positive definite. It is clear from Eq. (2.12) that $V \geq 0 \ \forall \ \mathbf{q}, \boldsymbol{\omega}$, it is zero at the origin, $V(\mathbf{q}_0, \boldsymbol{\omega}_0) = 0$ ($\mathbf{q}_0 = [\ 0,\ 0,\ 0,\ \pm 1\ ]^T$ and $\boldsymbol{\omega}_0 = [\ 0,\ 0,\ 0\ ]^T$) and that it is continuous. Note that by definition, a quaternion must satisfy $\epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 + \eta^2 = 1$. This can be rearranged to $\epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = 1 - \eta^2$. Then Eq. (2.12) becomes

$$V = \frac{1}{2}\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{J}\boldsymbol{\omega} + 1 - \eta^2 + \begin{cases} (\eta - 1)^2 & \text{for } \eta \geq 0 \\ \\ (\eta + 1)^2 & \text{for } \eta < 0 \end{cases}$$

and can be further rearranged to

$$V = \frac{1}{2}\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{J}\boldsymbol{\omega} + \begin{cases} 2(1 - \eta) & \text{for } \eta \geq 0 \\ \\ 2(1 + \eta) & \text{for } \eta < 0 \end{cases} . \tag{2.13}$$

Note here it is assumed that the inertia is constant, therefore $\dot{\mathbf{J}} = [0]$. Taking the derivative of Eq. (2.13) with time, yields:

$$\dot{V} = \frac{1}{2}\dot{\boldsymbol{\omega}}^T\mathbf{K}^{-1}\mathbf{J}\boldsymbol{\omega} + \frac{1}{2}\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{J}\dot{\boldsymbol{\omega}} + \begin{cases} -2\dot{\eta} & \text{for } \eta \ \geq 0 \\ \\ 2\dot{\eta} & \text{for } \eta < 0 \end{cases} \tag{2.14}$$

Noting that $\mathbf{K}^{-1}\mathbf{J} = (\mathbf{K}^{-1}\mathbf{J})^T$, Eq. (2.14) simplifies to

$$\dot{V} = \boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{J}\dot{\boldsymbol{\omega}} \begin{cases} -2\dot{\eta} & \text{for } \eta \geq 0 \\ \\ +2\dot{\eta} & \text{for } \eta < 0 \end{cases} . \tag{2.15}$$

Substituting the dynamics, Eq. (2.4) and Eq. (2.5), into Eq. (2.15):

$$\dot{V} = \boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{J}\mathbf{J}^{-1}\left[\boldsymbol{\omega}^\times\mathbf{J}\boldsymbol{\omega} + \boldsymbol{u}\right] \begin{cases} -2\left(-\frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{\omega}\right) & \text{for } \eta \geq 0 \\ \\ +2\left(-\frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{\omega}\right) & \text{for } \eta < 0 \end{cases} .$$

Simplifying and replacing $\mathbf{u}$ with Eq. (2.1), the above becomes:

$$\dot{V} = \boldsymbol{\omega}^T\mathbf{K}^{-1}\left[\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega} - \operatorname{sign}(\eta)\,\mathbf{K}\boldsymbol{\epsilon} - \left(1-\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\mathbf{D}\boldsymbol{\omega}\right] + \begin{cases} \boldsymbol{\epsilon}^T\boldsymbol{\omega} & \text{for } \eta \geq 0 \\ -\boldsymbol{\epsilon}^T\boldsymbol{\omega} & \text{for } \eta < 0 \end{cases}.$$

Next, remove the brackets by multiplying through by $\boldsymbol{\omega}^T\mathbf{K}^{-1}$.

$$\dot{V} = \boldsymbol{\omega}^T\mathbf{K}^{-1}\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega} - \operatorname{sign}(\eta)\,\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{K}\boldsymbol{\epsilon} - \left(1-\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{D}\boldsymbol{\omega} + \begin{cases} \boldsymbol{\epsilon}^T\boldsymbol{\omega} & \text{for } \eta \geq 0 \\ -\boldsymbol{\epsilon}^T\boldsymbol{\omega} & \text{for } \eta < 0 \end{cases} \tag{2.16}$$

Then note that $\mathbf{K}^{-1}\mathbf{K} = \mathbf{I}_3$, where $\mathbf{I}_3$ is the 3×3 identity matrix, and Eq. (2.16) becomes

$$\dot{V} = \boldsymbol{\omega}^T\mathbf{K}^{-1}\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega} - \operatorname{sign}(\eta)\,\boldsymbol{\omega}^T\boldsymbol{\epsilon} - \left(1-\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{D}\boldsymbol{\omega} + \begin{cases} \boldsymbol{\epsilon}^T\boldsymbol{\omega} \text{ for } \eta \geq 0 \\ -\boldsymbol{\epsilon}^T\boldsymbol{\omega} \text{ for } \eta < 0 \end{cases}. \tag{2.17}$$

Evaluating $\operatorname{sign}(\eta)$ in Eq. (2.17), and recognizing that $\boldsymbol{\omega}^T\boldsymbol{\epsilon} = \boldsymbol{\epsilon}^T\boldsymbol{\omega}$, it becomes:

$$\begin{aligned} \dot{V} =& \boldsymbol{\omega}^T\mathbf{K}^{-1}\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega} - \left(1-\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{D}\boldsymbol{\omega} + \begin{cases} -(1)\,\boldsymbol{\omega}^T\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^T\boldsymbol{\omega} & \text{for } \eta \geq 0 \\ -(-1)\,\boldsymbol{\omega}^T\boldsymbol{\epsilon} - \boldsymbol{\epsilon}^T\boldsymbol{\omega} & \text{for } \eta < 0 \end{cases} \\ \dot{V} =& \boldsymbol{\omega}^T\mathbf{K}^{-1}\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega} - \left(1-\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{D}\boldsymbol{\omega} \end{aligned} \tag{2.18}$$

Let us inspect the first scalar term of Eq. (2.18) and recall that by construction $\mathbf{K}$ is symmetric positive definite, therefore its inverse, $\mathbf{K}^{-1}$, exists and $\left(\mathbf{K}^{-1}\right)^T = \mathbf{K}^{-1}$; also noting that the dense inertia matrix, $\mathbf{J}$, is symmetric positive definite, (see [42]) therefore $\mathbf{J} = \mathbf{J}^T$; and further noting that the cross-product matrix operator produces a skew-symmetric matrix such that $(\boldsymbol{\omega}^{\times})^T = -\boldsymbol{\omega}^{\times}$, the following is true:

$$\begin{aligned}
\left[\boldsymbol{\omega}^T\mathbf{K}^{-1}\boldsymbol{\omega}^{\times}\mathbf{J}\boldsymbol{\omega}\right]^T &= \boldsymbol{\omega}^T\mathbf{J}^T\left(\boldsymbol{\omega}^{\times}\right)^T\left(\mathbf{K}^{-1}\right)^T\left(\boldsymbol{\omega}^T\right)^T \\
&= \boldsymbol{\omega}^T\mathbf{J}\left(-\boldsymbol{\omega}^{\times}\right)\mathbf{K}^{-1}\boldsymbol{\omega} \\
&= -\boldsymbol{\omega}^T\mathbf{J}\boldsymbol{\omega}^{\times}\mathbf{K}^{-1}\boldsymbol{\omega} \qquad (2.19) \\
&\leq -\|\boldsymbol{\omega}\|^2\|\mathbf{J}\|\|\boldsymbol{\omega}^{\times}\|\|\mathbf{K}^{-1}\| \leq 0. \qquad (2.20)
\end{aligned}$$

The transition from Eq. (2.19) to Eq. (2.20) follows from the definition of a vector or matrix norm and the equivalence of norms principle [43]. So, the candidate function derivative becomes:

$$\dot{V} \leq -\|\boldsymbol{\omega}\|^2\|\mathbf{J}\|\|\boldsymbol{\omega}^{\times}\|\|\mathbf{K}^{-1}\| - \left(1 - \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right)\boldsymbol{\omega}^T\mathbf{K}^{-1}\mathbf{D}\boldsymbol{\omega} \leq 0 \qquad (2.21)$$

Therefore, this controller is globally stable because $V >= 0$ and $\dot{V} \leq 0 \ \forall \ \mathbf{q}, \boldsymbol{\omega}$

2.2.2 Static Inertia, Imperfect Knowledge

In this section, the stability of the case where the inertia is unchanging, but knowledge of the that inertia is imperfect will be presented. The motivation for this discussion is manifold. First, the inertial knowledge of any spacecraft that is not perfectly solid is never completely accurate, even the inaccuracies can be bounded prior to launch. Second, in the case of derelict or non-cooperative clients, the true orientation of appendages, or state of fuel depletion may not be well known. Finally, for space debris, such as was generated by the breakup of the Hitomi observatory [14], the pieces that require detumbling and control may have inertias and masses that are impractical, if not impossible, to predict with much accuracy. Stability of the attitude controller in the face of error in the estimate of the coupled system inertia tensor is an important factor to consider. First, revisit Eqs. (2.10) and (2.11) but some error, E_i, will be explicitly included, on the true inertia when defining the gains.

$$\mathbf{K} = \omega_c^2 \left(\begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix} + \begin{bmatrix} E_{xx} & E_{xy} & E_{xz} \\ E_{xy} & E_{yy} & E_{yz} \\ E_{xz} & E_{yz} & E_{zz} \end{bmatrix} \right) = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{xy} & K_{yy} & K_{yz} \\ K_{xz} & K_{yz} & K_{zz} \end{bmatrix} \tag{2.22}$$

$$\mathbf{D} = 2\zeta_c\omega_c \left(\begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix} + \begin{bmatrix} E_{xx} & E_{xy} & E_{xz} \\ E_{xy} & E_{yy} & E_{yz} \\ E_{xz} & E_{yz} & E_{zz} \end{bmatrix} \right) = \begin{bmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{xy} & D_{yy} & D_{yz} \\ D_{xz} & D_{yz} & D_{zz} \end{bmatrix} \tag{2.23}$$

The fact that both gains remain symmetric positive definite matrices will be employed to prove the stability. Revisiting Eq. (2.16),

$$\dot{V} = \boldsymbol{\omega}^T \mathbf{K}^{-1} \boldsymbol{\omega}^\times \mathbf{J} \boldsymbol{\omega} - \operatorname{sign}(\eta)\, \boldsymbol{\omega}^T \mathbf{K}^{-1} \mathbf{K} \boldsymbol{\epsilon} - \left(1 - \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}\right) \boldsymbol{\omega}^T \mathbf{K}^{-1} \mathbf{D} \boldsymbol{\omega} + \begin{cases} \boldsymbol{\epsilon}^T \boldsymbol{\omega} & \text{for } \eta \geq 0 \\ -\boldsymbol{\epsilon}^T \boldsymbol{\omega} & \text{for } \eta < 0 \end{cases} . \tag{2.24}$$

As already stated, $\mathbf{K}^{-1}$ exists because $\mathbf{K}$ is symmetric positive definite by construction, then $\mathbf{K}^{-1}\mathbf{K} = \mathbf{I}_3$ for any choice of $\mathbf{K}$, regardless of the quality of the inertia knowledge. Further, it is never the case that the choice of gains, $\mathbf{K}$ or $\mathbf{D}$, are used to cancel any inertia terms from the Lyapunov candidate function or its derivative, i.e. no $\mathbf{K}^{-1}\mathbf{J}$, $\mathbf{D}^{-1}\mathbf{J}$, $\mathbf{J}^{-1}\mathbf{K}$, nor $\mathbf{J}^{-1}\mathbf{D}$ terms appear and thus must be removed. The proof above remains unchanged and $\dot{V} \leq 0 \;\forall\; \mathbf{q}, \boldsymbol{\omega}$ and the controller is globally stable with imperfect moment of inertia knowledge.

2.3 Performance of Diagonal Gain vs. Scalar: Stabilize From Tumble

This section will analyze where and how the proposed controller, Eq. 2.1, utilizing diagonal gains performs better than a scalar gain quaternion feedback controller as in Markley and Crassidis' controller, Eq. 2.9. Three cases will be compared - the diagonal matrix gain using the diagonal of the moment of inertia; the scalar gain using the minimum moment of inertia; and, the scalar gain using the maximum moment of inertia. (Note that in systems modeled as multiple bodies, "moment of inertia" in this case would be the composite moment of inertia of all bodies as expressed in the base body coordinate frame.) All other aspects of the systems will be the same; only the controller and its gains will differ.

The ACS system was executed at 10 Hz. The natural frequency of each controller was $\omega_n = 2\pi\ 0.001$ rad/sec and the controllers' damping ratio was $\zeta = \sqrt{2}/2$. Consider a system of a servicer satellite connected to a client satellite through a perfectly rigid robotic manipulator of unchanging configuration, as shown in Figure 2.1. The manipulator, described kinematically in Appendix A, is rigid at joint angles

$$\boldsymbol{\theta} = [\ 0^o,\ -60^o,\ 0^o,\ -120^o,\ 0^o,\ -60^o,\ 0^o\ ]\,,$$

and grappled to the client satellite at $\mathbf{r}^{TB}_{GP/_{TB}} = [\ -0.138,\ -0.5,\ -0.06\ ]^T$ meters, in the client's body frame; with the gripper's frame 75^o rotation about the client's Z axis, i.e. $\mathbf{q}_{GPd/_{TB}} = [\ 0,\ 0,\ 0.609,\ 0.793\ ]^T$.

As Wiktor notes in [44], there are three different measures for controller performance. For a given error quaternion, $\mathbf{q}$ and rate error, $\boldsymbol{\omega}$, and resulting torque command $\mathbf{M_b} = f(\mathbf{q}, \boldsymbol{\omega})$, the following is true:

1. the controller that minimizes $||\mathbf{M_b}||_1$ minimizes total instantaneous effort (i.e. fuel flow rate in the case of thruster-based attitude control);
2. the controller that minimizes $||\mathbf{M_b}||_2$ minimizes power; and,
3. the controller that minimizes $||\mathbf{M_b}||_\infty$ minimizes peak individual DOF torque.

The composite inertia of the system at the origin of the servicer (to the nearest 10

$kg \cdot m^2$), as expressed at the servicer origin in the servicer coordinates is

$$\mathbf{J_{SYS}^{SVR}} = \begin{bmatrix} 19450 & -450 & -4000 \\ -450 & 19260 & -610 \\ -4000 & -610 & 12510 \end{bmatrix}. \tag{2.25}$$

The masses of the servicer and client are 1,000 kg each, excluding the manipulator. The manipulator mass is 67.4 kg total. The masses and inertias of the two solar arrays are included, but the connections are rigid and flexibility is not present in this simulation for simplicity. The coupled system has an initial tumble rate of [30, 30, 30] degrees per second, as measured in the servicer body frame. The desired body rate with respect to inertial is zero. The initial and target inertial attitude is assigned to point the +Z axis of the servicer towards the Earth and point the +X axis of the servicer into the velocity vector. This Local Vertical Local Horizontal (LVLH) orientation allows the solar arrays to easily track the sun throughout the orbit. Initially this is equivalent to [88.8, −48.5, 178.0] degrees (to the nearest tenth of of a degree) in X-Y-Z Euler angles. The orbit is in GEO at 42,000 km. The simulation lasts 400 seconds. Applied torque was not limited nor quantized. The ACS controller is initially off, and is enabled one second into the simulation. When the ACS is enabled, it will apply torque to the servicer to reattain the desired attitude with zero body rate.[2] Figure 2.1 is the visualization of the coupled system from the Freespace simulation,

[2]To maintain a true LVLH attitude, the servicer would try to maintain a body rate about its Y axis equal

with the axes of each vehicle visualized as colored arrows. For both vehicles, the colors of the axis arrows are blue, green and red for X, Y and Z of each respective spacecraft body. The origin of the Servicer is at the center of the face nearest the client, also the face to which the robotic manipulator is mounted. (This face is hidden in Figure 2.1 because it is pointing towards the client.) The origin of the client is where the central axis of the marman ring intersects with the spacecraft's aft bulkhead, which is the bulkhead facing the viewer in the image. In Figure 2.1, the origin is visible at the location where the blue and green arrows of the Client's X and Y axes intersect. The Y axis of each spacecraft is parallel to the nominal axis of rotation for its respective solar arrays. The Z axis of the client is defined positive through the vehicle, away from the separation plane of the marman ring. The X axis of the client is perpendicular to both the Y and Z axes with its sign chosen to complete the right hand rule. The Z axis of the servicer is definied positive emanating out of the bulkhead where the manipulator is mounted. The X axis of the servicer is is perpendicular to both Y and Z axes with its sign chosen to complete the right hand rule. The results of the comparison between scalar gains (minimum and maximum inertia components) versus the diagonal are given below in terms of attitude and rate errors, as well as torque applied and time taken to settle errors below stated limits (to be defined below).

Figure 2.2 shows a comparison of the total attitude angular error as represented by

to the average orbital rate, but at this altitude, that is only 0.004 degrees per second, so a zero rate was chosen for this demonstration.

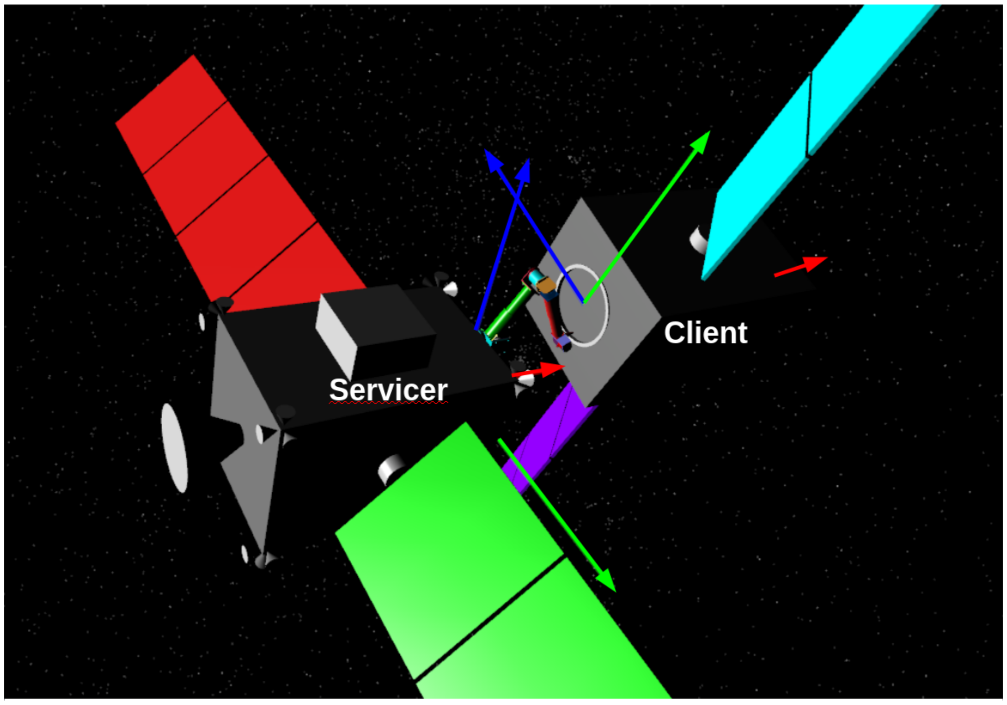


Figure 2.1: *Satellite body axes labeled in the grappled configuration. Blue, green and red arrows are the positive X, Y and Z axis, respectively, of each vehicle.*

just the error component of the Eigen Axis/Angle set. It should be remembered that in the Eigen Axis/Angle representation of an attitude error, a 0^o error is equivalent to a 360^o error. The sign from which 0^o (or 360^o) is approached only indicates whether the reduction of the error was achieved in a clockwise or counter-clockwise direction. All three methods converge to nearly no attitude error after 280 seconds. If one inspects the attitude error at 80 seconds, for instance, it is clear that the Scalar Max is closest to zero error, i.e. converging faster; next closest to zero error is the Diagonal controller; and finally, the Scalar Min controller is furthest from no error (even as quantified as its difference from a 360^o error). It will be discussed below by what metric this apparent deficiency of the Diagonal Gain case performs better than the Scalar Max case.

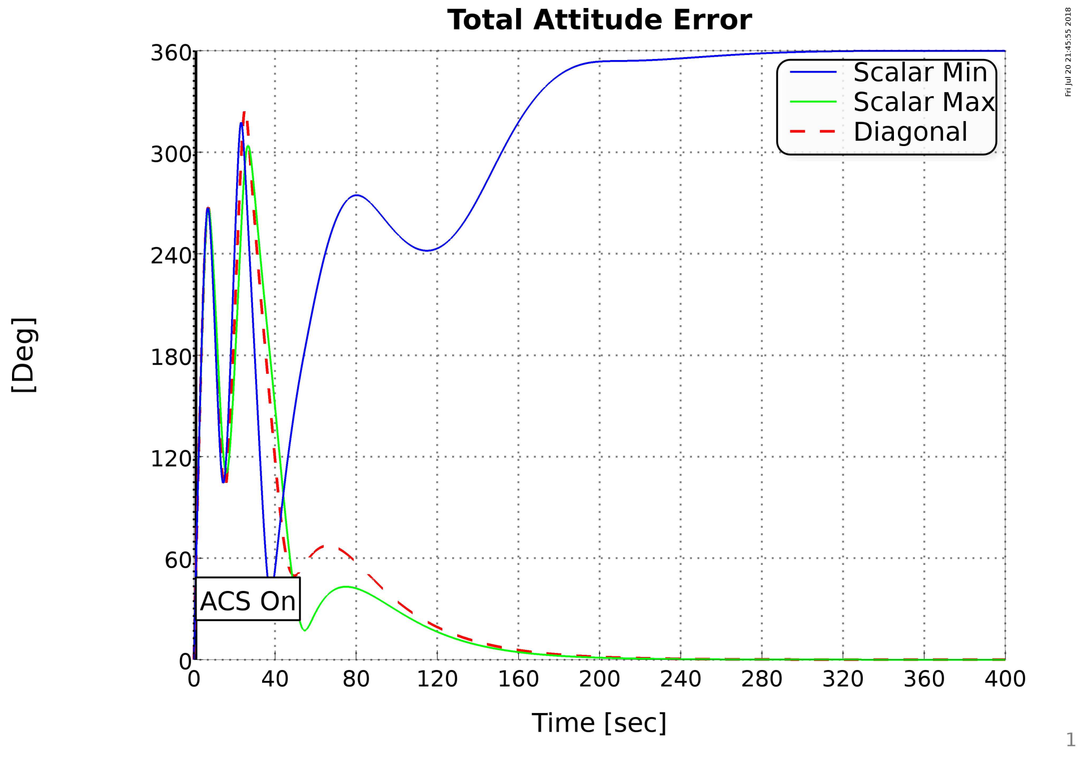


Figure 2.2: *Comparison of gain types and effect on total attitude angle error (in the Eigen Axis/Angle sense). Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation. Note that the domain of the Eigen rotation angle parameter is* $[0^o, 360^o)$ *therefore the scalar minimum controller is asymptotically approaching zero error, but from the other direction.*

Next, the attitude error on an individual axis basis will be reviewed, as represeneted by X-Y-Z Euler angles. Figures 2.3, 2.4, and 2.5 show the attitude error of each method in the

X, Y, and Z axes, respectively. Recall from Eq. (2.25) that the coupled system composite moments of inertia are within $40 kg \cdot m^2$ for the X and Y axes of the Servicer body, and the Z axis inertia is roughly 65% of X or Y. It may not be intuitive from the inertia tensor of the composite spacecraft, but in this case of a tumble rate equal across all three axes, the diagonal inertia scaling controller decreases the settling time for the Y axis best. The diagonal gain controller performs no worse than the scalar minimum controller case, but its utility beyond the scalar maximum will be shown be further metrics.

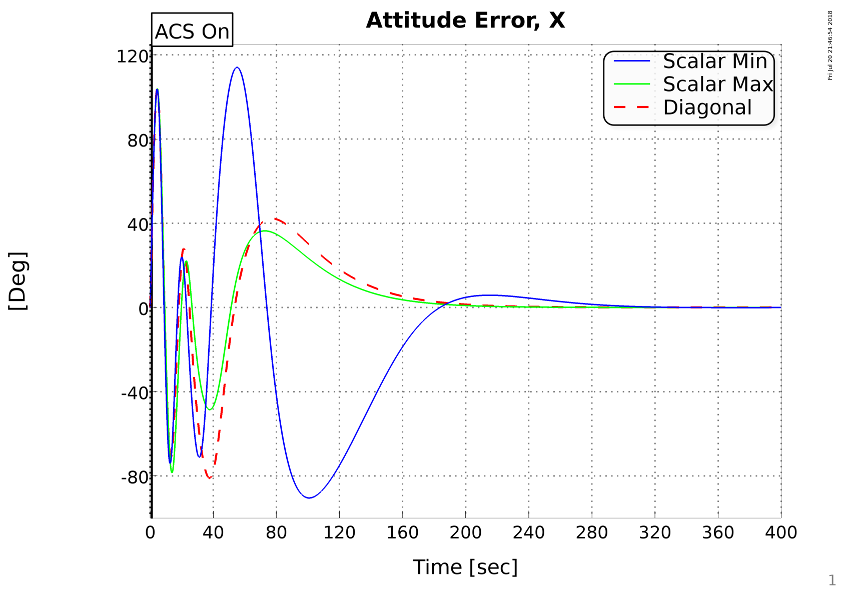


Figure 2.3: *Comparison of gain types and effect on X attitude angle error.*

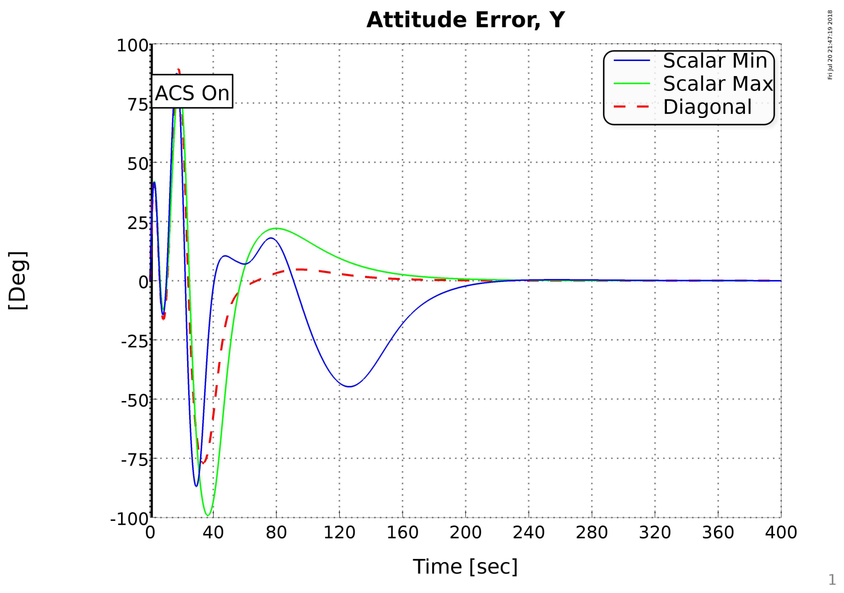


Figure 2.4: *Comparison of gain types and effect on Y axis attitude angle error.*

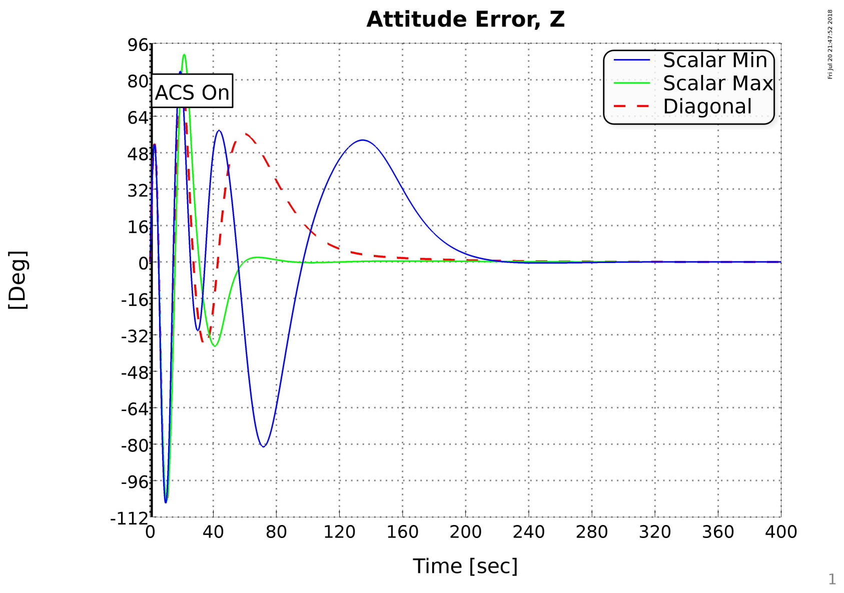


Figure 2.5: *Comparison of gain types and effect on Z axis attitude angle error.*

Figure 2.6 shows the RSS body rate of the servicer-client system over time as the it is brought under control. Again, the ACS system in this simulation was turned on at one second into the simulation. The ability of the diagonal gain controller to bring the tumble of the system to near zero is very similar to the scalar gains, as they all are assymptotically approaching zero by 120 seconds into the simulation, but the merits of the diagonal controller will be demonstrated by other metrics. Now, let us consider the rate errors on a per-axis basis. Figure 2.7 shows the differing performance between the contollers on the X axis of the servicer attitude rate; Fig. 2.8 shows the performance on the Y axis; and, Figure 2.9 shows the performance on the Z axis. Not surprisingly, the diagonal gain controller settles faster than the minimum scalar controller in all axes, but it is also able to settle faster than the scalar max controller in the Y axis (see Fig. 2.8). This is depsite the inertia of this axis being nearly as large as X axis.

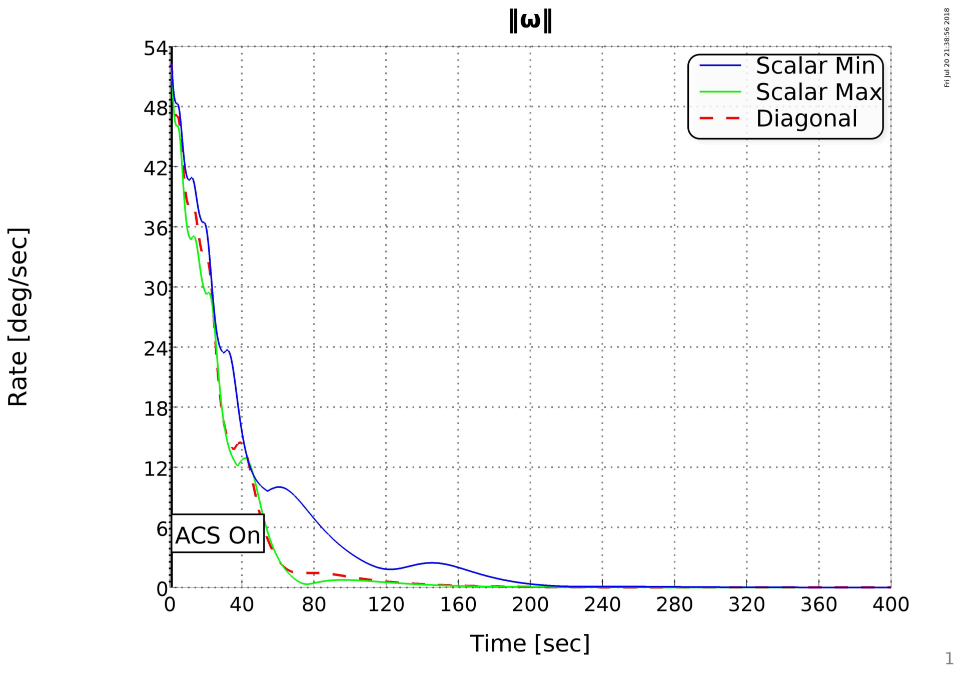


Figure 2.6: *Comparison of gain types and effect on RSS body rate error. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

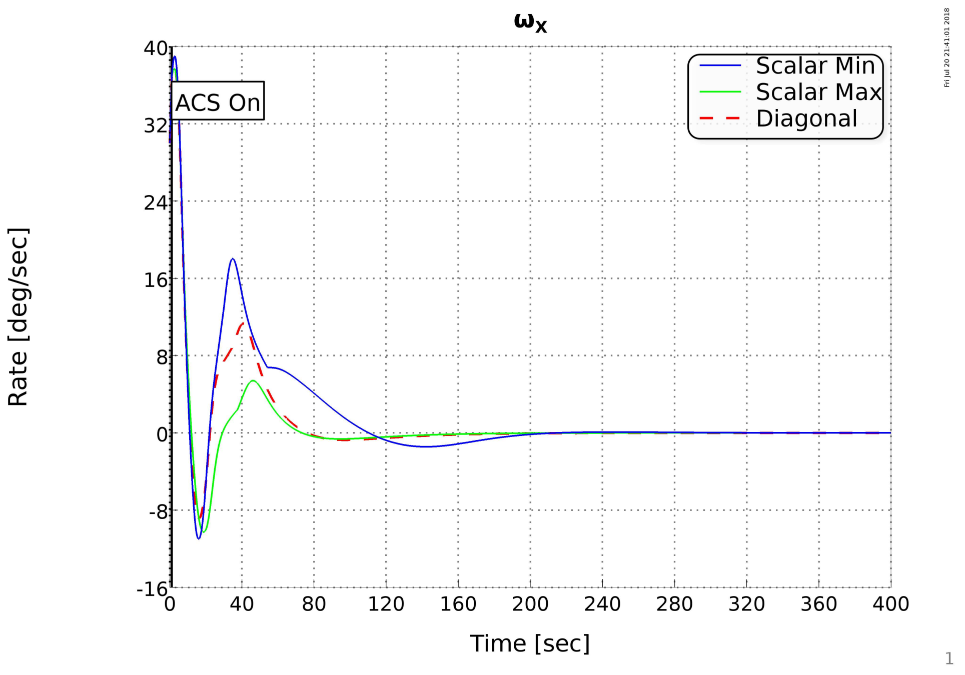


Figure 2.7: *Comparison of gain types and effect on X axis body rate error. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

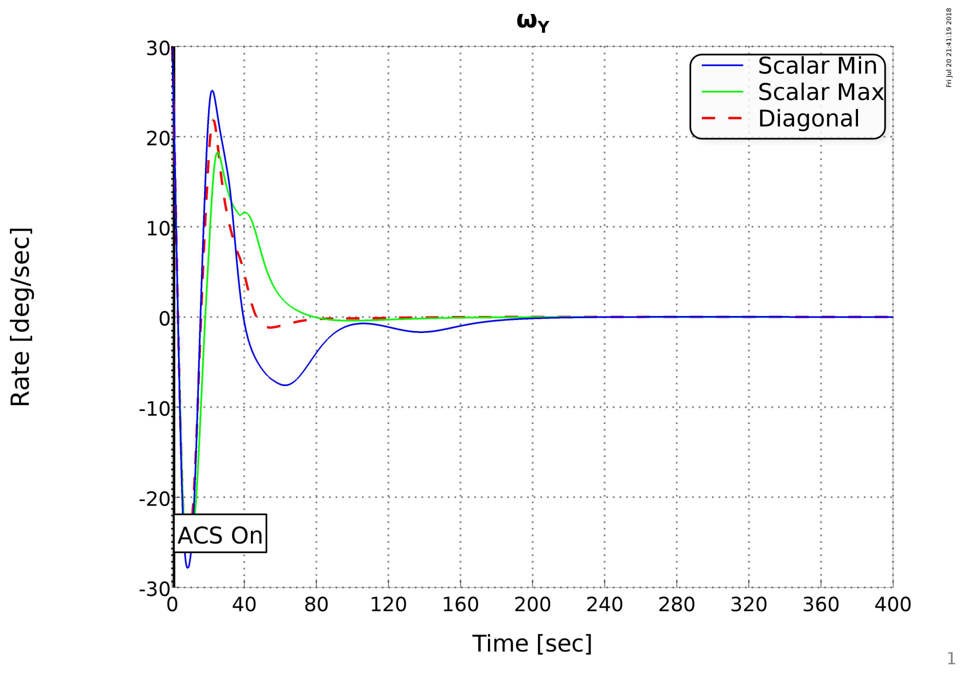


Figure 2.8: *Comparison of gain types and effect on Y axis body rate error. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

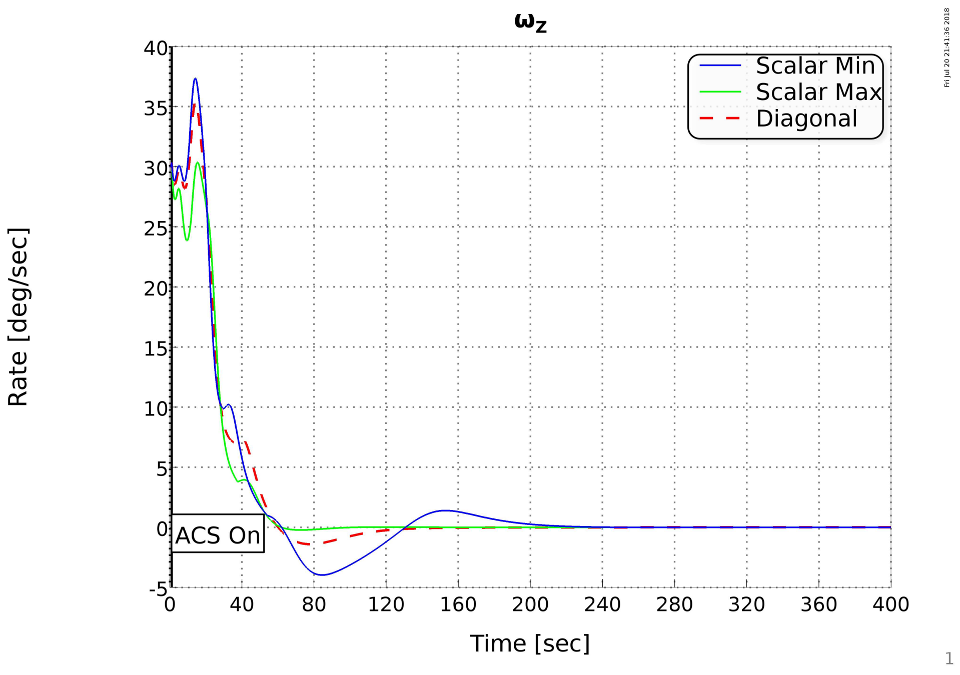


Figure 2.9: *Comparison of gain types and effect on Z axis body rate error. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

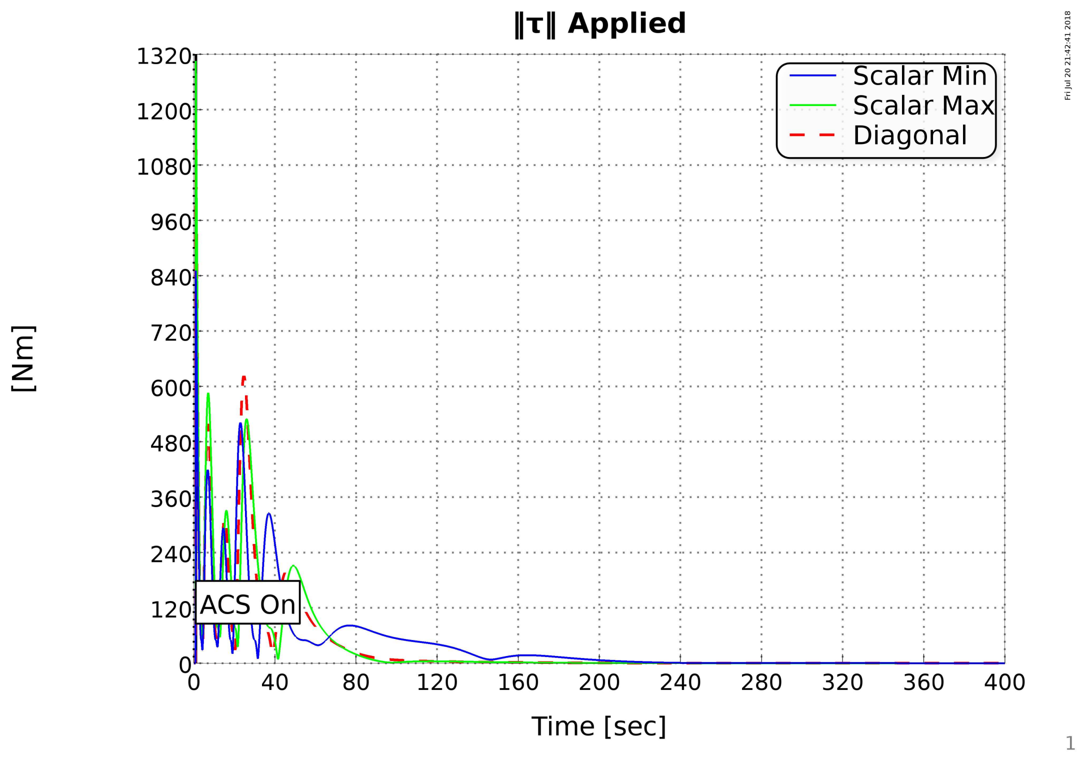


Figure 2.10: *Comparison of gain types and effect on applied RSS torque. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

Next, the torque required over time for each controller type will be inspected. Figure 2.10 shows the Root Sum Squared (RSS) of the torque applied over time to stabilize the system. Figures 2.11, 2.12, and 2.13 show the per-axis torque applied in the X, Y and Z axes, respectively. Figure 2.10 shows that the scalar minimum is the slowest to coverage, as epected, but also that the diagonal matrix gain converges slightly quicker than the scalar maximum controller.

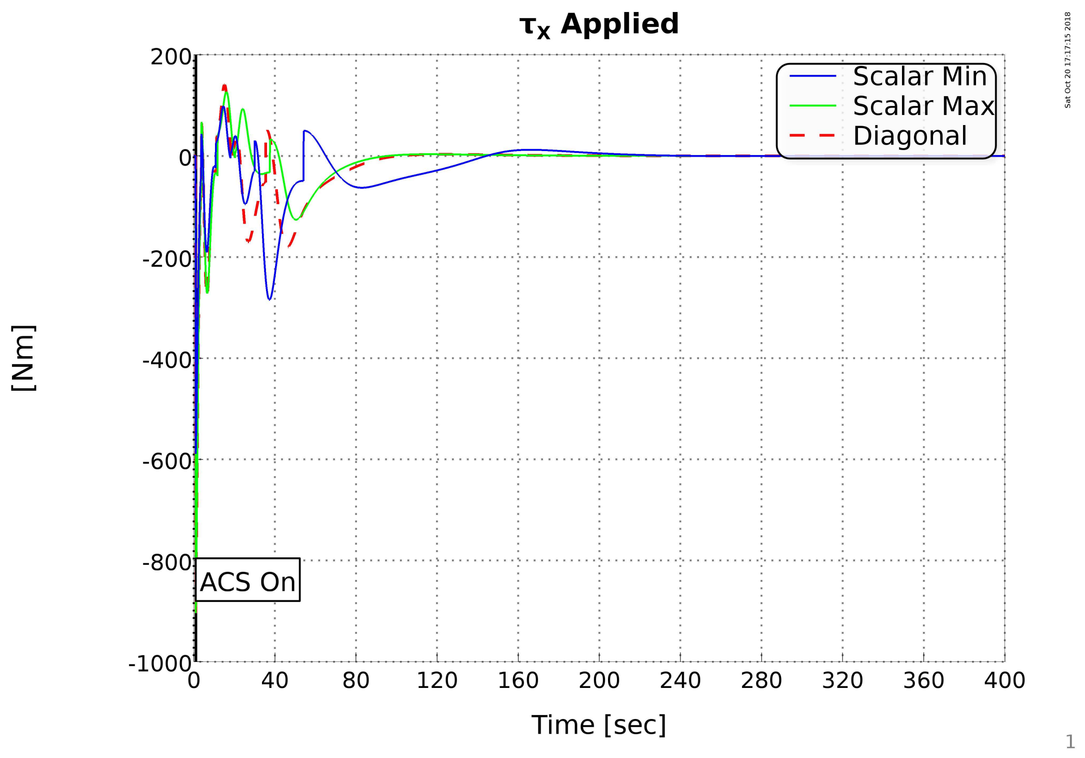


Figure 2.11: *Comparison of gain types and effect on applied X axis torque. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

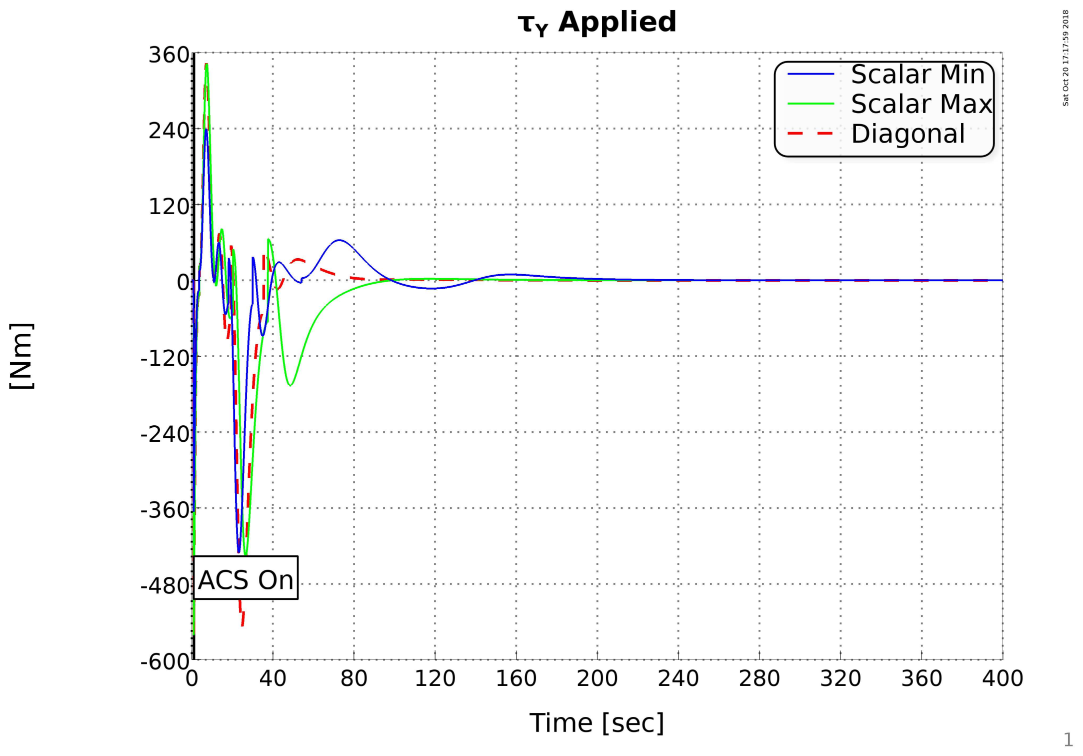


Figure 2.12: *Comparison of gain types and effect on applied Y axis torque. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

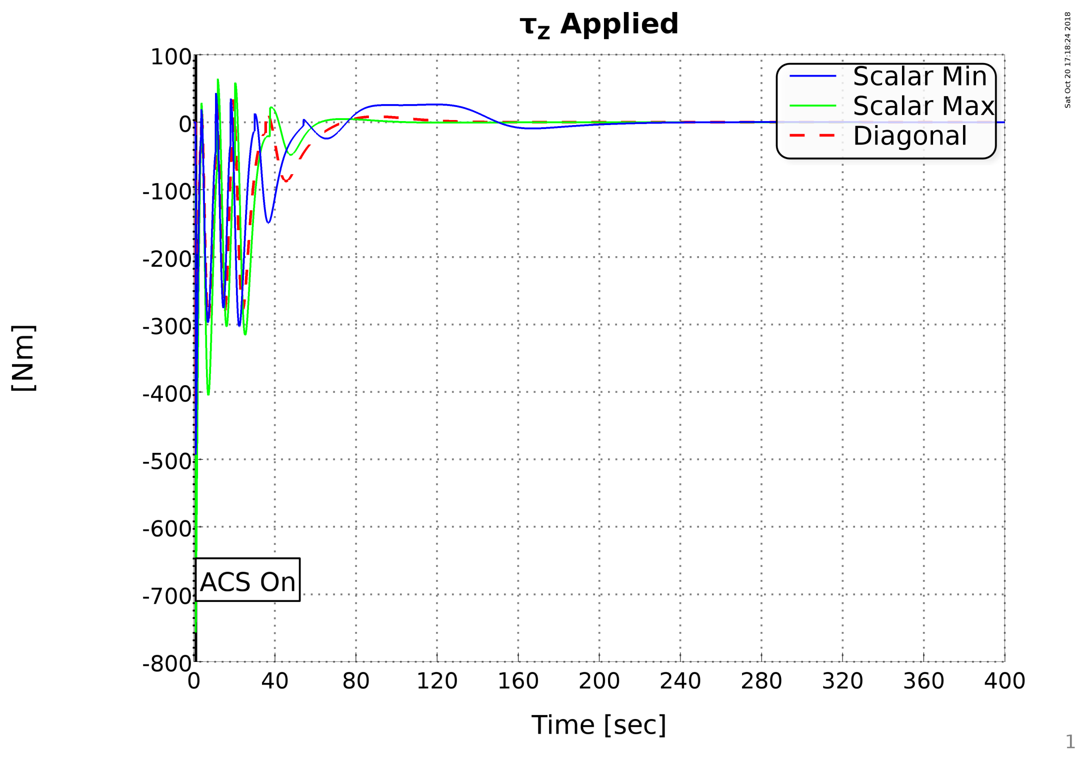


Figure 2.13: *Comparison of gain types and effect on applied Z axis torque. Scalar with minimum inertial scaling, Scalar with maximum inertia scaling, and Diagonal Matrix scaling from Inertial Tensor Diagonal. ACS is enabled 1 second into the simulation.*

Table 2.1: *Convergence times, in seconds, to within 100 arcsecond* (0.028^o) *tolerance of attitude error for scalar vs. diagonal matrix gains. [Note: Simulation maximum duration was 400 seconds. If convergence not achieved before 400.0 seconds, "N/A" is listed.]*

	Scalar Min	**Scalar Max**	**Diagonal Gain**
Err. Eigen Angle	N/A	323.4	338.9
Att. Err. X	N/A	302.9	319.6
Att. Err. Y	353.5	298.7	253.6
Att. Err. Z	389.9	310.4	332.3

Table 2.2: *Convergence times, in seconds, to within 10.0 arcsec/sec* (0.003^o*/sec*) *tolerance of rate error for scalar vs. diagonal matrix gains.*

	Scalar Min	**Scalar Max**	**Diagonal Gain**
Rate Err. RSS	352.2	281.3	296.3
Rate Err. X	199.6	152.2	162.5
Rate Err. Y	200.2	140.7	99.9
Rate Err. Z	208.5	82.4	127.6

Table 2.3: *Convergence times, in seconds, to within 0.10 Nm tolerance of applied torque for scalar vs. diagonal matrix gains.*

	Scalar Min	**Scalar Max**	**Diagonal Gain**
Torque RSS	343.1	240.5	247.0
Torque X	341.7	235.4	246.8
Torque Y	264.5	222.6	184.3
Torque Z	253.9	187.6	148.4

Table 2.4: *Maximum of Attitude Error vs Gain Type: total attitude error in an Eigen Angle/Axis sense; in absolute value, single DOF X-Y-Z Euler angle sense; or 1-Norm of Euler angles sense*

	Scalar Min	**Scalar Max**	**Diagonal Matrix**
Eig.Ang. (Deg)	360.0	304.0	325.7
$\|X\|$ (Deg)	179.7	179.6	179.8
$\|Y\|$ (Deg)	84.4	89.0	80.5
$\|Z\|$ (Deg)	179.7	179.9	179.5
1-Norm (Deg)	333.2	356.5	328.7

Table 2.5: *Mean of Attitude Error vs Gain Type: total attitude error in an Eigen Angle/Axis sense; in absolute value, single DOF X-Y-Z Euler angle sense; or 1-Norm of Euler angles sense*

	Scalar Min	**Scalar Max**	**Diagonal Matrix**
Eig.Ang. (Deg)	303.5	29.2	31.6
$\lvert X \rvert$ (Deg)	33.3	16.3	16.6
$\lvert Y \rvert$ (Deg)	8.4	7.6	6.2
$\lvert Z \rvert$ (Deg)	23.1	11.8	14.0
1-Norm (Deg)	64.9	35.7	36.8

Table 2.6: *Maximum of Rate Error vs Gain Type*

	Scalar Min	**Scalar Max**	**Diagonal Matrix**
RSS (Deg/Sec)	52.2	52.2	52.2
$\lvert X \rvert$ (Deg/Sec)	39.0	37.7	37.9
$\lvert Y \rvert$ (Deg/Sec)	30.0	30.0	30.0
$\lvert Z \rvert$ (Deg/Sec)	37.4	30.4	35.4
1-Norm (Deg/Sec)	90.0	90.0	90.0

Table 2.7: *Mean of Rate Error vs Gain Type*

	Scalar Min	**Scalar Max**	**Diagonal Matrix**
RSS (Deg/Sec)	5.1	3.5	3.7
$\|X\|$ (Deg/Sec)	2.5	1.3	1.7
$\|Y\|$ (Deg/Sec)	2.5	1.9	1.5
$\|Z\|$ (Deg/Sec)	2.9	2.1	2.5
1-Norm (Deg/Sec)	7.9	5.3	5.7

Table 2.8: *Maximum Applied Torque vs Gain Type*

	Scalar Min	**Scalar Max**	**Diagonal Matrix**
RSS (Nm)	850.8	1305.6	1172.2
$\|X\|$ (Nm)	589.0	903.9	903.9
$\|Y\|$ (Nm)	431.0	561.3	560.3
$\|Z\|$ (Nm)	493.0	756.6	493.0
1-Norm (Nm)	1447.8	2221.8	1957.2

Table 2.9: *Mean Applied Torque vs Gain Type*

	Scalar Min	**Scalar Max**	**Diagonal Matrix**
RSS (Nm)	39.3	35.6	34.9
$\|X\|$ (Nm)	23.1	15.4	20.0
$\|Y\|$ (Nm)	16.7	22.4	17.4
$\|Z\|$ (Nm)	20.3	17.1	16.2
1-Norm (Nm)	60.1	54.9	53.6

Figure 2.2 shows the attitude error as the angle of the Eigen Axis / Eigen Angle representation of the error quaternion as a means of capturing the total error across all three axes simultaneously. The domain of the Eigen Angle is $[0^o, 360^o)$, so the scalar mininum case settling to 360^o is equivalent to reaching zero error. Figures 2.3, 2.4, and 2.5 show the attitude error in the X, Y, and Z axes of the servicer, respectively, as X-Y-Z Euler angle representations of the error quaternion. Figure 2.6 shows the body rate error represented as a Root Sum Squared (RSS) (2-Norm) value for all axes. Figures 2.7, 2.8 and 2.9 show the rate errors as measured in the servicer frame on the X, Y, and Z axes respectively. Figure 2.10 shows the applied torque to the servicer as an RSS of all axes. Figures 2.11, 2.12, and 2.13 show the torque applied in the servicer's X, Y and Z axes, respectively. [3] It is not immediately clear from the figures alone, however, if the diagonal gain is an improvement so further metrics are required.

Table 2.1 summarizes how long each method takes to reduce the attitude error to within 100 arcseconds (0.028^o) of desired in an Eigen Angle sense and individually in the X, Y, and Z axes. By these metrics, the diagonal gain case only outperforms the scalar max case in the Y axis, and the scalar min performs worse than both others in all categories. Table 2.2 summarizes how long each method takes to reduce the rate error to within 10 arcsec/sec (0.003 deg/sec) of desired in RSS, and X, Y, and Z axes. Again the diagonal gain

[3]Given initial peak torques, it is clear that not limiting the thruster forces (and resulting body torques) enables all controllers to arrest the tumble (initially 52 deg/sec) rapidly but would require substantial thrusters.

case only outperforms the others in the Y axis and the scalar min case performs worse than the others in all categories. Table 2.3 summarizes how long each method takes to no longer apply any torque above 0.10 Nm in the RSS and X, Y, and Z axes.[4] Here the diagonal gain case reaches this criteria faster than the others in the Y and Z axes, while the scalar min case remains worst performing in all categories.

Table 2.4 summarizes the maximum attitude error over time in the Eigen Angle sense, 1-Norm (of the error quaternion represented as X-Y-Z Euler angles) sense and in the X, Y and Z axes. By this measure, the diagonal gain method outperforms the other methods in three of the five categories: the Y and Z axes, and the 1-Norm. Table 2.5 summarizes the mean (as opposed the the maximum, previously discussed) attitude error over time in each axis as welll as overall in the Eigen Angle sense and 1-Norm sense. By this measure the diagonal gain method is best only in the Y axis. Tables 2.6 and 2.7 summarize the performance of each method in terms of the rate error as the maximum and mean, respectively, over time. Tables 2.8 and 2.9 summarize the performance of each method in terms of the applied torque applied as the maximum and mean, respectively, over time. In terms of both mean and max applied torque or rate error over time, all methods perform very similarly.

Table 2.10 summarizes the sum of the RSS of applied torques over time for the whole simulation, which is analogous to the total power expended. By this measure, the diagonal

[4]Any suitably small torque value can be chosen here. In the design of real spacecraft, it would likely be the torque created by the thrust of the thrusters' minimum on time times either the minimum or maximum arm from thruster to center of mass, or some multiple thereof.

Table 2.10: *Total Torque Applied, as measured by Sum of 2-Norm (analogous to total power expended) and 1-Norm (analogous to total fuel expended) over time, per gain type, .*

Method	$\mathbf{\Sigma\|\cdot\|_2}$ **[Nm]**	$\mathbf{\Sigma\|\cdot\|_1}$ **[Nm]**
Scalar Min	157.0×10^3	240.7×10^3
Scalar Max	142.3×10^3	219.7×10^3
Diagonal Matrix	139.6×10^3	214.4×10^3

gain method performs 1.9% better than the scalar max gain method and 11.0% better than the scalar min gain method. Table 2.10 summarizes the sum of the 1-Norm of applied torques over time for the whole simulation, which is analogous to total fuel expended. This measure also shows that the diagonal gain method performs best; it requires 2.4% less fuel than the scalar max gain method and 10.9% less fuel than the scalar min gain method.

Finally, the experiment above demonstrates that the proposed nonlinear quaternion feedback controller, with diagonally-scaled gains, provides minimal power expended per reorientation, and minimum total fuel expended compared to the scalar methods (Table 2.10). These are metrics which mission designers may care greatly about.

2.4 Conclusion

This chapter has detailed a novel nonlinear quaternion feedback controller. The stability of the controller has been proven in the sense of Lyapunov in the case of static inertia with perfect and imperfect knowledge of that inertia. The performance has been compared to a similar controller and it has been demonstrated to expend less power per reorientation maneuver and expend less fuel. In the next chapter, the development of FFT parameter selection will be discussed.

Chapter 3: Manipulator Controller for Vibration Reduction

3.1 Introduction

Several methods exist to use robotic manipulators to reduce impact energy due to contact or base motion due to manipulator motion. The following chapter will discuss several of these methods and why each is insufficient for the application proposed: detumbling coupled satellites while reducing appendage motion. The desired manipulator control strategy for reducing appendage vibration during a coupled detumble maneuver has the following properties:

1. allows non-zero system base attitude rates (during maneuver),
2. allows an active ACS (non-zero external base torques),
3. allows non-zero initial angular momentum,
4. allows non-zero reaction forces and torques at the end effector;
5. and, provides active system vibration reduction.

A further desire is that the proposed solution requires as little increased hardware as necessary to perform the above task. Several current methods will be discussed below and the deficiencies of each will be highlighted. Finally, a new method is proposed that meets the above requirements.

3.2 Current Manipulator Actuation Methods

3.2.1 Generalized Jacobian

The generalized Jacobian method of coordinated manipulator-base control proposed in K. Yoshida, et al. [45], for use on the ETS-VII, derives a Jacobian suitable for use in resolved acceleration control that accounts for motion of the base caused by the internal torques of the manipulator to resolve motion in inertial space. It starts by writing the equations of motion for the manipulator and satellite base as Equation 3.1.

$$\begin{bmatrix} \mathbf{H}_b & \mathbf{H}_{bm} \\ \mathbf{H}_{bm}^T & \mathbf{H}_m \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{x}}_b \\ \ddot{\boldsymbol{\phi}} \end{bmatrix} + \begin{bmatrix} \mathbf{c}_b \\ \mathbf{c}_m \end{bmatrix} = \begin{bmatrix} \mathcal{F}_b \\ \boldsymbol{\tau} \end{bmatrix} + \begin{bmatrix} \mathbf{J}_b^T \\ \mathbf{J}_m^T \end{bmatrix} \mathcal{F}_h \tag{3.1}$$

Where in Eq. 3.1, $\mathbf{H}_b \in \mathbb{R}^{6\times 6}$ is the inertia matrix of the base satellite; $\mathbf{H}_m \in \mathbb{R}^{n\times n}$ is the inertia matrix for a manipulator with n links; $\mathbf{H}_{bm} \in \mathbb{R}^{6\times n}$ is the coupling inertia matrix; $\mathbf{c}_b \in \mathbb{R}^6$ is the velocity-dependent nonlinear term for the base; $\mathbf{c}_m \in \mathbb{R}^6$ is the velocity-dependent nonlinear term for the manipulator; $\mathcal{F}_b \in \mathbb{R}^6$ is the force and torque exerted on the center of mass of the base; $\mathcal{F}_h \in \mathbb{R}^6$ is the force and torque exerted on the end effector;

$\boldsymbol{\tau} \in \mathbb{R}^6$ are the internal torques on the manipulator joints; $\mathbf{x_b} \in \mathbb{R}^6$ are the inertial base coordinates; and $\boldsymbol{\phi} \in \mathbb{R}^6$ are the joint angles. In the free-flying case, with the ACS inactive, a simplifying assumption is made that the external forces are zero, i.e. $\mathcal{F}_b = \mathbf{0}$ and $\mathcal{F}_h = \mathbf{0}$. If external reactions are zero, then the linear and angular momentum of the system $\left(\mathcal{P}^T, \mathcal{L}^T\right)^T$ remain constant.

$$\begin{bmatrix} \mathcal{P} \\ \\ \mathcal{L} \end{bmatrix} = \mathbf{H}_b \dot{\mathbf{x}}_b + \mathbf{H}_{bm} \dot{\boldsymbol{\phi}} \tag{3.2}$$

Then the velocity of the manipulator hand (end effector) in the inertial frame is given by Equation 3.3, below.

$$\dot{\mathbf{x}}_h = \mathbf{J}_m \dot{\boldsymbol{\phi}} + \mathbf{J}_b \dot{\mathbf{x}}_b \tag{3.3}$$

Equation 3.2 is simplified by the authors assuming that the angular momentum of the system is zero, (which cannot be the case for a tumbling servicer-client system) and reordered to solve for $\dot{\mathbf{x}}_b$, then this is used to simplify Equation 3.3 to Equations 3.4 and 3.5.

$$\dot{\mathbf{x}}_h = \mathbf{J}_g \dot{\boldsymbol{\phi}} \tag{3.4}$$

$$\mathbf{J}_g = \mathbf{J}_m - \mathbf{J}_b \mathbf{H}_b^{-1} \mathbf{H}_{bm} \tag{3.5}$$

The manipulator Jacobian, $\mathbf{J}_m$, base-motion-to-hand-motion Jacobian, $\mathbf{J}_b$, and the coupling inertia matrix, $\mathbf{H}_{bm}$ are all dependent upon current manipulator joint angles. Not clearly stated is the chosen attitude representation for the satellite base in inertial space, other than indicating it was a vector in $\mathbb{R}^3$. The authors conclude with flight experiment data showing minimal absolute inertial disturbance to the manipulator hand while performing straight-path tracking in inertial space, a successful demonstration of this approach to limit unintended motion. The Generalized Jacobian is therefore a method to reduce base motion disturbance introduced by the motion of the arm itself, in the inertial domain, without regard to frequency content, but only with the critical requirements that the system starts with zero initial angular momentum and any ACS is inactive. Therefore it is not useful during detumbling.

3.2.2 Reactionless Null-Space

Reactionless Null-Space (RxNS) controllers seek to design manipulator motion to achieve some tool goal while moving in such a way as to minimize reaction torques applied to the base [46]. They have been demonstrated on the Japanese orbital experiment ETS-VII [45], [47], and studied for use on the Japanese Experimental Module Remote Manipulator System (JEMRMS)/Small Fine Arm (SFA) on the ISS [48], and other systems.

Start with system dynamics as in Equation 3.1, and the constant momentum, Equation 3.2, in the case with no external forces or torques. The momentum equation can be partitioned to include only the angular momentum components (note the tilde on the inertia matrices to indicate partition) such that,

$$\tilde{\mathbf{H}}_b\boldsymbol{\omega}_b + \tilde{\mathbf{H}}_{bm}\dot{\boldsymbol{\phi}} = \boldsymbol{\mathcal{L}}. \tag{3.6}$$

If there is no initial angular momentum, $\boldsymbol{\mathcal{L}} = 0$ and no base disturbance, $\boldsymbol{w}_b = 0$, then we are left with

$$\tilde{\mathbf{H}}_{bm}\dot{\boldsymbol{\phi}} = \mathbf{0}. \tag{3.7}$$

and a null-space solution

$$\dot{\boldsymbol{\phi}} = \left(\mathbf{I} - \tilde{\mathbf{H}}_{bm}^{+}\tilde{\mathbf{H}}_{bm}\right)\dot{\boldsymbol{\zeta}} \tag{3.8}$$

where $^{+}$ above indicates the right pseudo-inverse, and $\dot{\boldsymbol{\zeta}}$ is an arbitrary vector. Also, note that $\left(\mathbf{I} - \tilde{\mathbf{H}}_{bm}^{+}\tilde{\mathbf{H}}_{bm}\right)$ is a projection operation of the input onto the nullspace of the coupling inertia. The number of degrees of freedom for $\dot{\boldsymbol{\zeta}}$ is $n - 3$, where n is the number of degrees of freedom of the manipulator. In the case of a 6 DOF manipulator, either the position or orientation of the end effector may be chosen, while achieving tool motion with no base reactions. The motion of the end effector observed in the satellite body frame can be written as

$$\begin{bmatrix} {}^{b}\boldsymbol{v}_h \\ {}^{b}\boldsymbol{\omega}_h \end{bmatrix} = \begin{bmatrix} \mathbf{J}_v \\ \mathbf{J}_\omega \end{bmatrix} \dot{\boldsymbol{\phi}}. \tag{3.9}$$

Where $\mathbf{J}_v$ and $\mathbf{J}_\omega$ above are the standard translational and rotational matrix Jacobian matrices, respectively. Then the solutions to Equation 3.10 and 3.11, below, provide the paths to reactionless translation or reorientation, respectively.

$$\begin{bmatrix} \tilde{\mathbf{H}}_{bm} \\ \mathbf{J}_v \end{bmatrix} \dot{\boldsymbol{\phi}} = \begin{bmatrix} \mathbf{0} \\ {}^{b}\boldsymbol{v}_h \end{bmatrix} \tag{3.10}$$

$$\begin{bmatrix} \tilde{\mathbf{H}}_{bm} \\ \mathbf{J}_\omega \end{bmatrix} \dot{\boldsymbol{\phi}} = \begin{bmatrix} \mathbf{0} \\ {}^{b}\boldsymbol{\omega}_h \end{bmatrix} \tag{3.11}$$

In [45], Yoshida notes that Equations 3.10 and 3.11 both take the form $\mathbf{M}\dot{\boldsymbol{\phi}} = \mathbf{x}$. To solve these equations, $\mathbf{M}$ must be inverted, yet they have potentially many singularities which prevents inversion. Yoshida suggests the following work-around for square $\mathbf{M}$:

$$\dot{\boldsymbol{\phi}} = k \cdot adj(\mathbf{M})\,\mathbf{x} \tag{3.12}$$

where $adj(\cdot)$ is the adjugate matrix operation, but choose k equal to $1/det(\mathbf{M})$, then the result is the same, but k can be bounded near a singularity.

Previous work by Nenchev et al. [46] built on even earlier work by Lee and Book [49] to include the reduction of vibration in the flexible base by motion of the manipulator. They begin with the system dynamics in Equation 3.13, below. They differ from Eq. 3.1 by the inclusion of the terms $\mathbf{D}_b$, $\mathbf{D}_m$, and $\mathbf{K}_b$, damping of the base motion, damping of the manipulator motion, and stiffness of the base, respectively and the exclusion of the forces experienced at the hand, $\mathcal{F}_h$.

$$\begin{bmatrix} \mathbf{H}_b & \mathbf{H}_{bm} \\ \mathbf{H}_{bm}^T & \mathbf{H}_m \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{x}}_b \\ \ddot{\boldsymbol{\theta}} \end{bmatrix} + \begin{bmatrix} \mathbf{D}_b & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_m \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}_b \\ \dot{\boldsymbol{\theta}} \end{bmatrix} + \begin{bmatrix} \mathbf{K}_b \mathbf{x}_b \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{c}_b \\ \mathbf{c}_m \end{bmatrix} = \begin{bmatrix} \mathcal{F}_b \\ \boldsymbol{\tau} \end{bmatrix} \tag{3.13}$$

The approach of Lee and Book was to assume that the arm is initially stationary, thereby approximating $\mathbf{c}_b$ and $\mathbf{c}_m$ by zero; second, they assumed that deflections would be small, so they approximated the submatrices by the joint variables only, not their transcendental functions. They further simplify the system by ignoring the base damping. With that, they observed that the upper half of the equations of motion can be rewritten as Equation 3.14, below.

$$\mathbf{H}_b \ddot{\mathbf{x}}_b + \mathbf{K}_b \mathbf{x}_b = -\mathbf{H}_{bm} \ddot{\boldsymbol{\theta}} \tag{3.14}$$

Then with the choice of joint accelerations,

$$\ddot{\theta} = \mathbf{H}_{bm}^{+} \mathbf{H}_b \mathbf{G}_b \dot{\mathbf{x}}_b \tag{3.15}$$

where $\mathbf{G}_b$ is a constant gain matrix, and $\mathbf{H}_{bm}^{+} \in \mathbb{R}^{n \times m}$ is the right pseudo-inverse of the inertia coupling matrix, results in vibration damping of the system. While the above method allows base motion and reaction forces and torques at the end effector, it makes no attempt to reduce system motion caused by appendage vibration.

Subsequently, Nenchev et al. proposed a joint-acceleration-based controller that achieves vibration damping, joint motion damping and arbitrary desired motion ($\mathbf{u}$), via Equation 3.16, below.

$$\ddot{\theta} = \mathbf{H}_{bm}^{+} \left(\mathbf{H}_b \mathbf{G}_b \dot{\mathbf{x}}_b - \dot{\mathbf{H}}_{bm} \dot{\boldsymbol{\theta}} \right) + \left(\mathbf{I} - \mathbf{H}_{bm}^{+} \mathbf{H}_{bm} \right) \mathbf{u} - \mathbf{G}_m \dot{\boldsymbol{\theta}} \tag{3.16}$$

Where above, $\mathbf{G}_m$ is the joint damping control gain, and it is assumed that the manipulator provides kinematic redundancy, i.e. $n > 6$. Nenchev et al. also later propose a torque-based vibration suppression control algorithm as Equation 3.17, below.

$$\boldsymbol{\tau} = \mathbf{H}_m \mathbf{H}_{bm}^{+} \mathbf{G}_b \dot{\mathbf{x}}_b + \mathbf{c}_m - \mathbf{H}_m \mathbf{H}_{bm}^{+} \mathbf{c}_b + \mathbf{H}_m \left(\mathbf{I} - \mathbf{H}_{bm}^{+} \mathbf{H}_{bm} \right) \mathbf{u} - \mathbf{G}_m \dot{\boldsymbol{\theta}} \tag{3.17}$$

However, by the derivation of the dymanics of the above Nenchev methods, reaction forces and torques at the end effector are not allowed; disqualifying them from this application.

Washino, et al. [50] used a reactionless nullspace controller to reduce vibrations imparted on a flexible base by a teleoperated manipulator by driving a secondary manipulator attached to the same base with motions designed to cancel the momentum induced by the primary manipulator. I.e. while a human drove the primary manipulator to achieve a desired task, the secondary manipulator was driven by equation 3.18, below.

$$\dot{\boldsymbol{\theta}}_s = -\mathbf{H}_{bms}^{+}\mathbf{H}_{bmp}\dot{\boldsymbol{\theta}}_p \tag{3.18}$$

The requirement of the Washino method of vibration reduction on a secondary manipulator limits the solution space for a coupled satellite problem to where the available inertia of the secondary manipulator is comparable to the system inertia of the client satellite at the end of the primary manipulator. This introduces the need for either a comparatively large secondary manipulator or a reduced client-space for possible inertias, negatively impacting the number of client satellites that could be included.

It is apparent that reactionless nullspace methods have many and varied applications. This method's controller input relies on calculations including $\mathbf{H}_m$ and $\mathbf{H}_{bm}$, however, making its performance dependent upon the accuracy of those quantities. Also, sensing of disturbances is limited to joint measurements, estimates of base state, and ultimately may not be observable.

3.2.3 Robust Impedance Control

Wongratanaphisan and Cole [51] demonstrate vibration suppression of manipulator motion while performing contact maneuvers on a flexible base by inserting a first-order filter into its impedance controller. This filter transfer function is given by Equation 3.19, below.

$$Q(s) = \frac{\tau_1 s + 1}{\tau_2 s + 1} \tag{3.19}$$

Where $\tau_1 < \tau_2$ and τ_2 selected such that $1/\tau_2$ is less than the first natural frequency of the base. This is inserted into the modified impedance control law,

$$U(s) = -\frac{m}{m_d} Q^{-1}(s) K(s) X_m(s) + \left(\frac{m}{m_d} Q^{-1}(s) - 1 \right) F(s) \tag{3.20}$$

where $K(s) = b_d s + k_d$, m_d, c_d and k_d are the desired impedance mass, damping and stiffness, respectively. Further, $X_m(s) = X(s) - X_b(s)$, where $X_m(s)$ is the position state of the manipulator, $X_b(s)$ is that of the base and $X(s)$ is that of the whole system. m is the mass of the whole system. $F(s)$ is the force sensed at the end of the arm with a FTS. $U(s)$ is the resulting actuator response of the manipulator. In this method, the impedance controller can be designed as desired, and the filter design parameters τ_1 and τ_2 can be adjusted until the closed-loop system shows stability via the Popov criterion. Again, this method requires some a priori knowledge of the natural frequencies of concern for filter design, like static structrual filters. However, most disqualifying of all is that the derivation of the dynamics requires that the ACS not be active during this manipulator control phase.

3.2.4 Compliance Control

The Force-Moment Accommodation (FMA) controller of Morimoto, et al. [52] is described by the transfer function in Equation 3.21, below, where K_0 is the product of several controller gains from inner control loops that were part of their design, M is the compliance synthetic mass, C is the compliance synthetic damping, and K is the compliance synthetic stiffness.

$$G_{comp}\left(s\right) = -\frac{K_0\left(Ms^2 + Cs + K\right)}{Ms^2 + Cs + K + K_0} \tag{3.21}$$

whereas a mass-spring-damper system has a transfer function of Equation 3.22, below, expressed as the ratio of force, $F(s)$ and $V(s)$, often referred to as the impedance. [53].

$$Z(s) = F(s)/V(s) = \frac{Ms^2 + Cs + K}{s} \tag{3.22}$$

The FMA controller therefore modifies desired end effector position, relative to its mounting location, based on detected disturbance in an FTS, which is similar to our problem, as the exact position of the end effector relative to the base is less important than preventing contact between vehicles, but it does not taken advantage of the FTS signal to determine the state of what the gripper is in contact with. Furhtermore, this method seeks to reduce to zero the detected forces and torques at the end effector without regard to the forces or torques that might be required during a detumble maneuver.

3.2.5 Summary

Table 3.1 summarizes some of the key aspects of the manipulator control stategies that have been highlighted above. The manipulator control strategy required for a grappled

Table 3.1: *Comparison of manipulator motion damping methods. Darkened cells indicate undesirable properties for the goals of this research. Reactive Control is the novel manipulator control strategy proposed here.*

Method	**Allows Base Att. Rate**	**Allows Active ACS**	**Allows Nonzero Init. Ang. Mom.**	**Allows Reaction At End Effector**	**Reduces System Vibration**	**Secondary Manipulator Not Required**
Generalized Jacobian	No	No	No	No	Yes	Yes
RxNS (Yoshida)	No	No	No	No	Yes	Yes
RxNS (Nenchev)	Yes	No	Yes	No	Yes	Yes
RxNS (Washino)	No	No	No	No	Yes	No
Robust Impedance Control	Yes	No	Yes	Yes	Yes	Yes
FMA Compliance	N/A	N/A	N/A	Yes	No	Yes
Reactive Control	Yes	Yes	Yes	Yes	Yes	Yes

de-tumbling maneuver that reduces system vibration requires that it allows for:

1. non-zero system base attitude rates (during maneuver),
2. an active ACS (non-zero external base torques),
3. non-zero initial angular momentum,
4. non-zero reaction forces and torques at the end effector;
5. and, active system vibration reduction.

An additional desire of the control strategy is a minimal requirement on additional hardware. An FTS does not represent a large design or mass penalty, but, for instance, a redundant or secondary manipulator does represent a potentially large mass penalty (as would be required by Washino's RxNS method). None of the reviewed methods allows for all of these conditions simultaneously, thus necessitating something new. The proposed manipulator control strategy, here described as Reactive Control, will be presented in the next section.

3.3 Proposed Reactive Control

Reactive Control proposed here is based on a catersian end-effector path that is informed by the dominant modal disturbances of the system appendages. The high-level path control takes place in the Arm Control (ARMCTRL) module, while the low-level joint control is enacted in the Manipulator (MANIP) module. Figure 3.1 is a block diagram of the

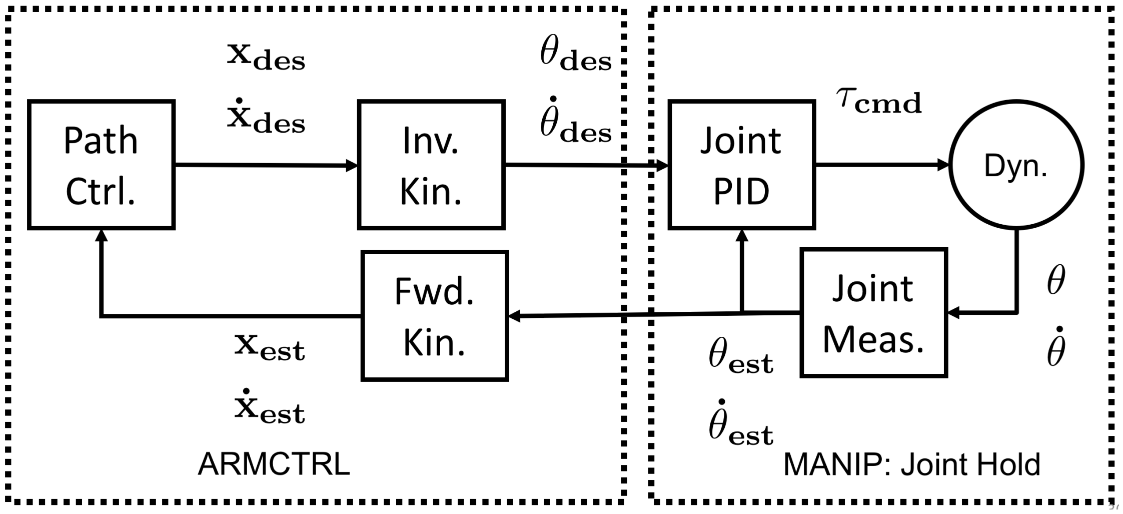


Figure 3.1: *Block diagram of manipulator Reactive Control; cartesian path control is produced based on detected modal frequency and phase reactions to counteract appendage motion.*

data flow between the modules. The path generated in the path control module depends on the detection of dominant appendage modal frequencies, which will be discussed in Chapter 4. In other words, the manipulator tool is used as a tunable damper or antinode to the system disturbances.

3.3.1 Manipulator Trajectory Generation

Once the reaction frequencies, $f_{f,i}$ and $f_{m,i}$, and phases, $\phi_{f,i}$ and $\phi_{m,i}$, have been identified, per the above methods, and the reaction DOF chosen, the reaction forces and/or torques for each DOF $i \in (x, y, z)$ can be constructed in Cartesian manipulator space. For force reactions, the set position of the grapple fixture (end effector or tool) is modified.

The per-DOF modification is computed by Eq. 3.23. The full tool position modification, $\mathbf{p}_{react/des}$, (read as the position change $\mathbf{p}$ from desired, "des", to reactive, "react", pose) is then a composition of each DOF per Eq. 3.24. The final commanded tool position, $\mathbf{p}_{cmd}$, is then the sum of the desired tool position relative to the base and the reaction motion per Eq. 3.25, i.e. $\mathbf{p}_{cmd} = \mathbf{p}_{react/base}$.

$$p_{react/des,fi} = a_{f,i} \sin\left(2\pi f_{react,fi} + \phi_{react,fi}\right) \tag{3.23}$$

$$\mathbf{p}_{react/des} = \begin{bmatrix} p_{react/des,fx} \\ p_{react/des,fy} \\ p_{react/des,fz} \end{bmatrix} \tag{3.24}$$

$$\mathbf{p}_{cmd} = \mathbf{p}_{react/des} + \mathbf{p}_{des/base} \tag{3.25}$$

The torque reactions are modifications of the *desired* tool attitude (as opposed to the current measured tool attitude) with respect to the manipulator base, on a per-DOF basis by Eq. 3.26. The attitude modifications will be small (relative to the entire domain of potential tool orientations), so these per-DOF attitude modifications are converted to a quaternion difference, $\mathbf{q}_{react/des}$, using an Euler-angle to quaternion routine per Eq. 3.27.

$$\theta_{mi} = a_{m,i} \sin\left(2\pi f_{react,mi} + \phi_{react,mi}\right) \tag{3.26}$$

$$\mathbf{q}_{react/des} = \mathit{euler2quat}\left(\theta_{mx}, \theta_{my}, \theta_{mz}, \text{``}\mathit{123}\text{''}\right) \tag{3.27}$$

The new desired tool attitude is computed via quaternion multiplication as in Eq. 3.28, where $\mathbf{q}_{des/base}$ is the attitude change from the base to the initial desired tool location.

$$\mathbf{q}_{cmd} = \mathbf{q}_{react/des} \otimes \mathbf{q}_{des/base} \tag{3.28}$$

Finally, the commanded joint torques for the manipulator, $\boldsymbol{\theta}_{cmd}$, are determined with Eq. 3.29, where $\boldsymbol{\theta}_{current}$ are the current manipulator joint angles and the function *invkin*() performs the inverse kinematics based on the current pose.

$$\boldsymbol{\theta}_{cmd} = invkin\left(\boldsymbol{\theta}_{current}, \mathbf{p}_{cmd}, \mathbf{q}_{cmd}\right) \tag{3.29}$$

The selection of $a_{m,i}$ and $a_{f,i}$ is by linear gain scheduling, driven by the reaction frequency in question. That is, the amplitudes are scheduled on a per-axis and DOF combination, k, basis by $a_k = m_k f_{react,k} + b_k$, where m_k is a negative scalar that decreases the amplitude as the detected frequency to react to increases and b_k is the scalar Y-intercept. This set of commanded joint angles is maintained by the manipulator Joint Hold mode.

3.4 Conclusion

Many manipulator control algorithms have been reviewed which do not meet all of the design criteria for the problem under consideration. A new method has been proposed predicated on detection of appendage motion within the coupled satellite system. Subsequent chapters will discuss how that coupled system appendage modal content can be detected, and further, performance results of the proposed algorithms will be discussed.

Chapter 4: Sliding FFT Window Properties and Automated Peak Detection Method

In order for the Reactive Control manipulator controller to work, the frequency and phase of the desired mdoe to react to must be identified. Frequency identification is done using the discrete FFT. The FFT has many parameters which effect its performance and utility. This chapter presents terminology, rationale for parameter selection and finally a method fast automated peak detection of the output signals that may then be used to drive the manipulator.

4.1 Considerations for FFT sampling of unknown target signals

This section will discuss the ability of an FFT to identify the frequency and phase of a dominant frequency in a noisy signal of multiple frequencies, where the dominant frequency is the frequency that has the largest amplitude. The MATLAB implementation of the FFT algorithm is used here to demonstrate the algorithms capability for this task, whereas in subsequent Freespace simulations the open source GNU Scientific Library (GSL) FFT library is employed to take advantage of the speed of its C-language implementation.[1] For the opening example, a time-series signal is constructed of five additive cosine signals with the following properties:

- frequencies sampled from a uniform distribution of range $[0, 100)$ Hertz;
- phases sampled from a uniform distribution of range $[0, 180)$ Degrees;
- amplitudes of $[25, 20, 15, 10, 5]$ plus a random amplitude sampled from a uniform distribution of range $[0, 2.5)$; and
- guassian noise of an amplitude of one half of the lowest amplitude cosine frequency.

For this discussion, the phrase *target frequency* means the frequency of the dominant sinusoidal component, i.e. of the largest amplitude within the signal. A real-time signal may only be digitally sampled once. A digitized sample, for example a reading from a FTS, is continually sampled at the sampling frequency, F_s. A subset of this data, referred to as a *window*, is then passed into the FFT algorithm. The number of samples between the start of each window k and $k + 1$ that are processed is called the *window stride*. The samples in a window that are passed to the FFT do not need to be consecutive. The spacing of the samples selected from the full data set and placed in the window is called the *element stride*, for example an element stride of 1 would take every adjacent sample from the full data set; an element stride of two would take every other sample, etc.

[1]The Freespace simulation environment will be explained in detail in Chapter 5.

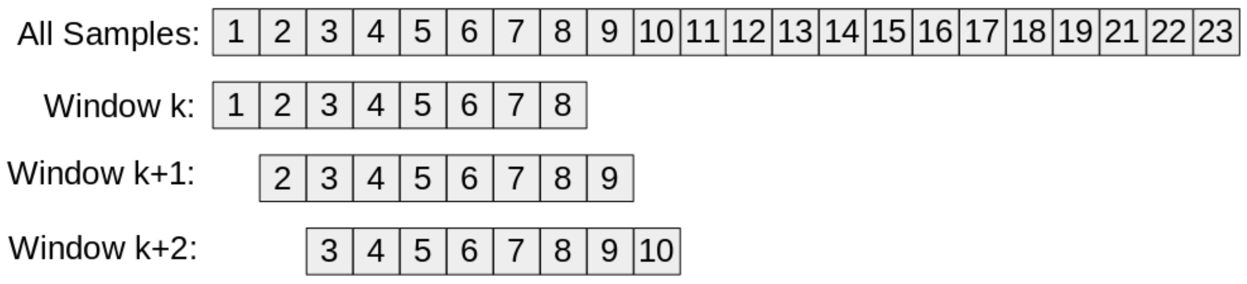


Figure 4.1: *Diagram depicting a sliding window of eight samples, a window stride of one sample and element stride of one sample. There is an overlap of 87.5% between windows at k and $k + 1$.*

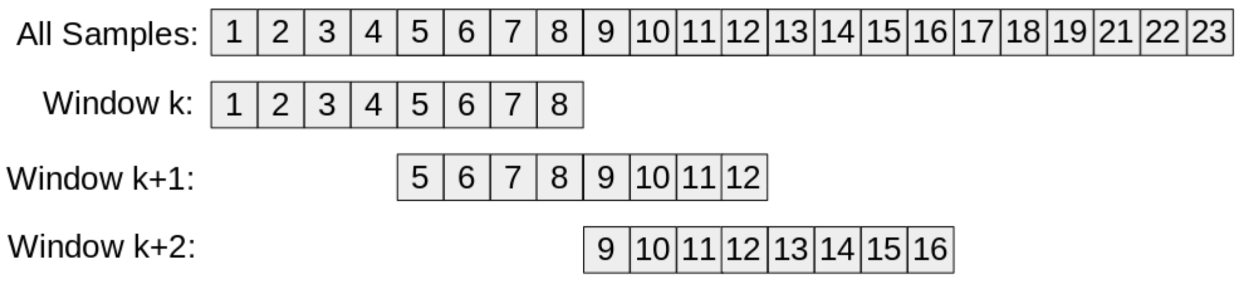


Figure 4.2: *Diagram depicting a sliding window of eight samples, a window stride of four samples and element stride of one sample. There is an overlap of 50% between windows k and $k + 1$.*

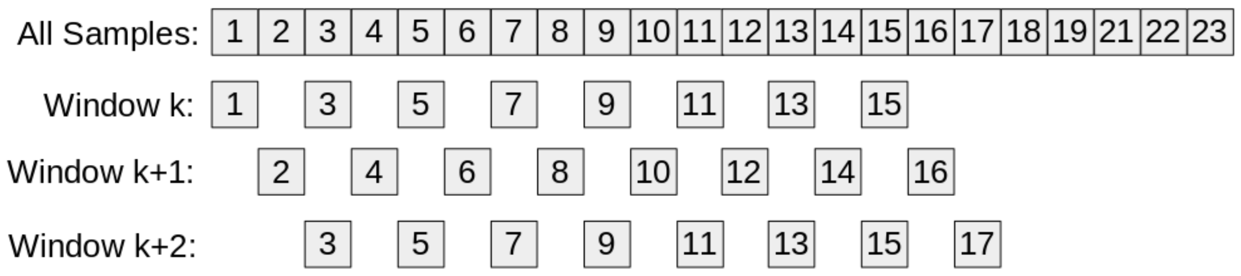


Figure 4.3: *Diagram depicting a sliding window of eight samples, a window stride of one sample and element stride of two samples. There is an overlap of 87.5% between windows k and $k + 1$ in time, but none of the samples are the same.*

Given a large buffer of digitallly sampled data, a subset of that data is called a *window*. If that window were to move across that buffer as processing progress that is called a *sliding window*. [54]. Figures 4.1, 4.2 and 4.3 illustrate the different effects of window stride and element stride on a sliding window. Figure 4.1 depicts a sliding window of eight samples, a window stride of one sample and element stride of one sample. There is an overlap of 87.5% in time between windows at k and $k+1$. The sampling frequency of the windows, $F_{s,win}$, in this example is the same as the sampling frequency of the original set of all samples, F_s. Figure 4.2 depicts a sliding window of eight samples, a window stride of four samples and element stride of one sample. There is an overlap of 50% between windows k and $k+1$. Again, the sampling frequency of the windows, $F_{s,win}$, in this example is the same as the sampling frequency of the original set of all samples, F_s. Figure 4.3 depicts a sliding window of eight samples, a window stride of one sample and element stride of two samples. There is an overlap of 87.5% between windows k and $k+1$ in time, but none of the samples are the same in any two subsequent windows; however, element overlap between windows k and $k+2$ is 87.5%. In this third example, the effective sampling frequency of each window is now half that of the full set of samples, i.e. $F_{s,win} = F_s/s_{elem}$, where s_{elem} is the element stride of the window.

The detection of the phase of a signal using an FFT is improved by knowing the proper sampling frequency and oversampling by powers of two, e.g. using a sampling frequency several power-of-two multiples that of the frequency of intersest, as noted in Oppenheim [54] and Lyons [55]. The following experiment however, demonstrates that the detection of the phase of the target frequency is not improved by selecting the element stride to more closely sample the data at a frequency that is a power-of-two multiple of the detected target frequency, given a previously digitally sampled signal. Here are the steps in this experiment:

1. Construct a time-series signal with a dominant signal of frequency f_{target} and phase ϕ_{target}, as discussed above.
2. Sample the signal at very high frequency, F_s, that exceeds potential frequencies of interest by several orders of magnitude. This will be referred to as stage 1 sampling.
3. Construct the stage 1 window of four seconds duration and $s_{elem1} = 1$.
4. Perform FFT to determine dominant signal frequency, f_1 and phase, ϕ_1.
5. Select an element stride, s_{elem2}, such that the stage 2 window's effective sampling frequency is close to a power of two multiple of the target frequency, i.e. $s_{elem2} = round(F_s/f_1)$ and $F_{s,win} = F_s/s_{elem2}$.
6. Construct the second window with s_{elem2} from the same four second buffer and perform an FFT once again to determine frequency and phase of the dominant frequency, f_2 and ϕ_2, respectively. This will be referred to as stage 2 sampling.
7. Compare the the error between f_1 and f_2 versus f_{target}, and ϕ_1 and ϕ_2 versus ϕ_{target}.

The minimum frequency detection error is one half of the FFT bin size. The size of each bin is a function of the sampling frequency and number of samples in a window.

The size of the stage 1 bins is $b_1 = F_s/n_{win1}$ in Hz and the size of stage 2 bins is $b_2 = (F_s/s_{elem})/floor(n_{win1}/s_{elem})$. Therefore, except for small n_{win1} where the effect of the floor function is noticable, the bin sizes of stages 1 and 2 are nearly identical.

Figure 4.4 shows the input phase and frequency of the dominant signal in the randomly generated signals for one thousand randomly sampled runs. This is included to illustrate the diversity of input cases. The results of their processing will be discussed below.

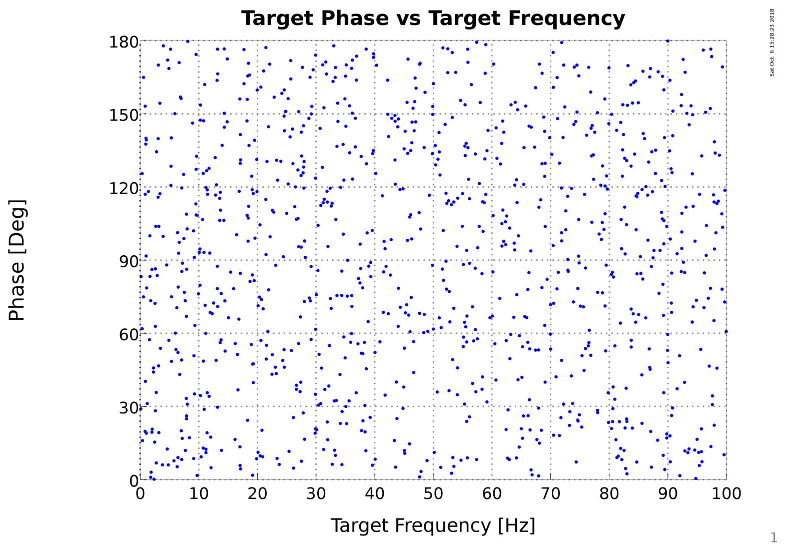


Figure 4.4: *Target phase vs target frequency of the input signals for one thousand randomly generated signals.*

Figures 4.5 and 4.6 show the detected frequency and phase error, respectively, versus the target frequency for stages 1 and 2. This demonstrates that the error is not related to the target frequency of the signal.

Figures 4.7 and 4.8 show the deteced frequency and phase error, respectively, versus

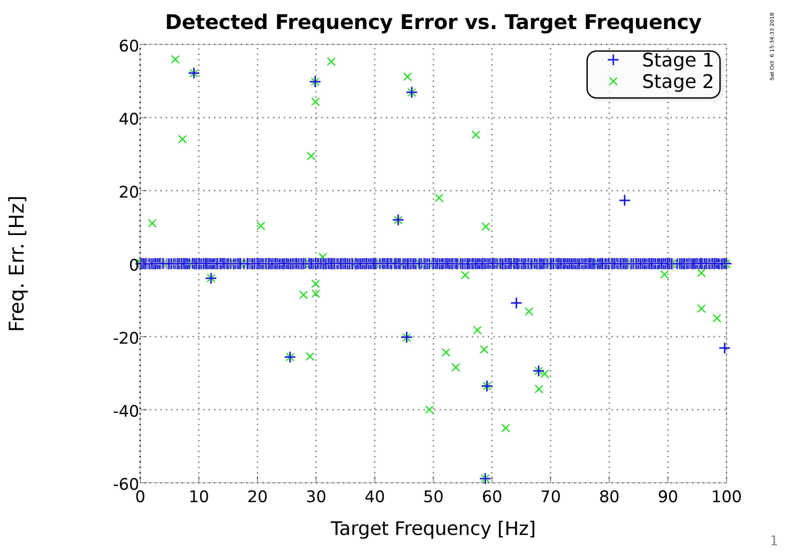


Figure 4.5: *Detected frequency error vs target frequency for stages 1 and 2. One thousand randomly sampled signals.*

target phase for stages 1 and 2. From these it can be seen that the error is not related to target phase of the signal.

Figures 4.9 and 4.10 show detected frequency and phase error, respectively, versus signal noise as a percentage of the maximum signal. One can conclude from these that the error is not related to signal noise.

Figures 4.11 and 4.12 show the detected frequency error and detected phase error, respectively, for stage 1 versus stage 2. If both stages detected the target frequency with minimal error, Figure 4.11 would show an nearly circular ellipse with radii of one-half the bin width of each stage and Figure 4.12 would be a point at the origin. It is clear from Figure 4.11 that this method has a detrimental impact on frequency detection performance and

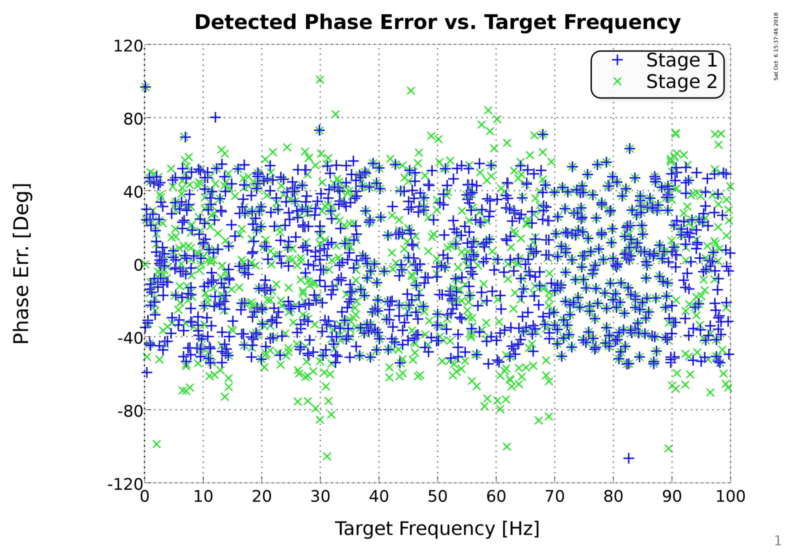


Figure 4.6: *Detected phase error vs target frequency for stages 1 and 2. One thousand randomly sampled signals.*

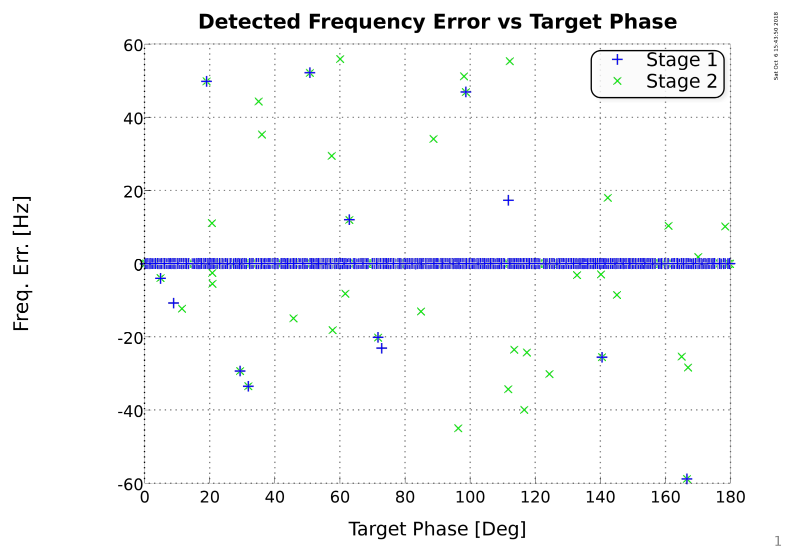


Figure 4.7: *Detected frequency error vs target phase. One thousand randomly sampled signals.*

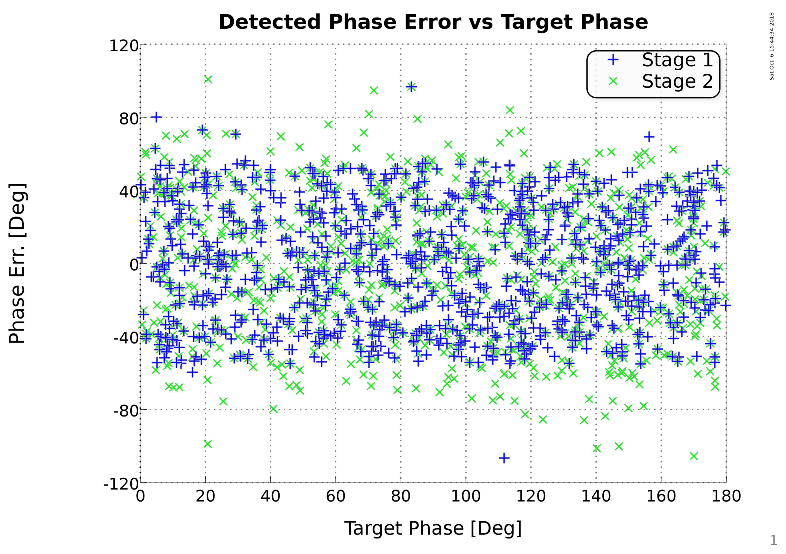


Figure 4.8: *Detected phase error vs target phase. One thousand randomly sampled signals.*

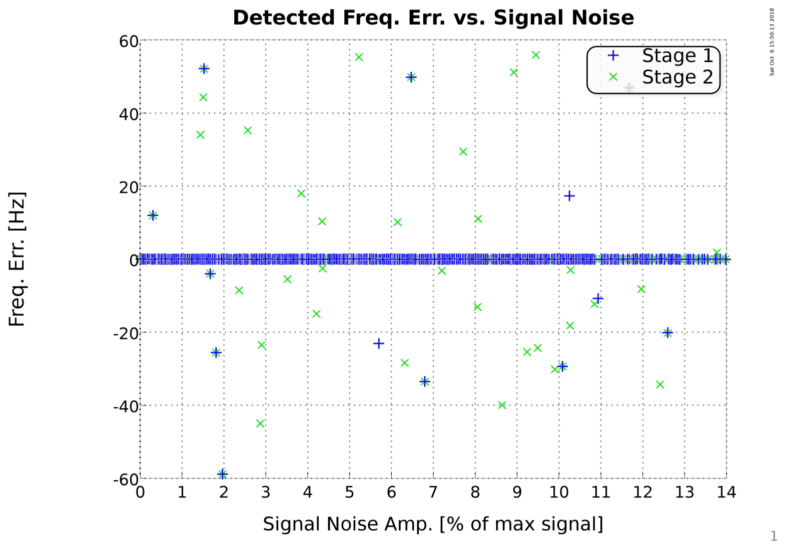


Figure 4.9: *Error in detected dominant signal frequency vs signal noise as a percentage of dominant signal amplitude.*

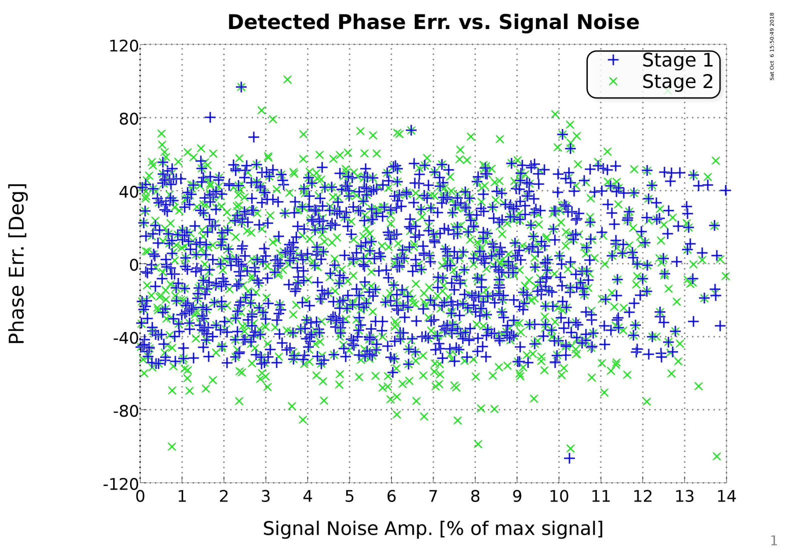


Figure 4.10: *Error in detected dominant signal phase vs signal noise as a percentage of dominant signal amplitude.*

from Figure 4.12 that it has no discernable positive impact on phase detection performance.

Another way to view the frequency and phase detection error of stage 2 detection is as a percentage of the stage 1 error. Let f_{s1} be the detected frequency in stage 1; let f_{s2} be the detected frequency of stage 2; and, let f_{target} be the target frequency. Then the stage 2 detection error as a perctage of stage 1 error is $e_{freq} = \frac{abs(f_{s2} - f_{target})}{abs(f_{s1} - f_{target})}$. The same is applied to the phase detection error, e_{ϕ}. Figures 4.13 and 4.14 show histograms for the stage 2 detection errors of frequency and phase, respectively, for the above described experiment when the elements of the stage 2 window are all within the stage 1 window length, i.e. the stage 2 window is a subset of the stage 1 window. Several outliers with stage 1 error near zero were not included in the histogram because these skewed the histograms to include

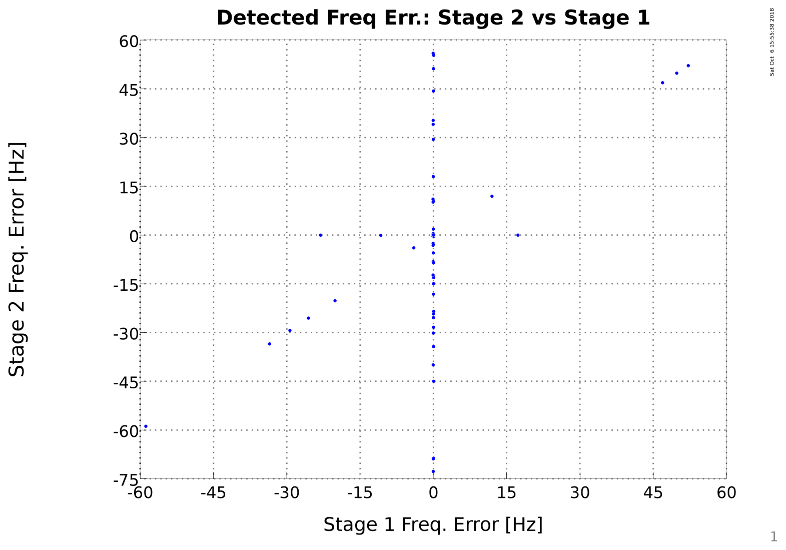


Figure 4.11: *Detected frequency error of stage 1 vs stage 2. One thousand randomly sampled runs. No outliers removed.*

error increases of thousands. These outliers are safe to ignore in the histogram because their percentage of 1,000 runs is $< 1\%$. What is telling is that more than 40% of solutions of stage 2 have a frequency identification error of 10% or less. In other words, the frequency identification is equal or better for stage 2 for all runs. However, roughly two thirds of stage 2 phase detection errors saw a decrease, and about one third of runs saw an increase in the phase detection error. Phase detection error must be less than 180^o to not create constructively oscillatory signals in respone.

It is possible to determine the impact of the number of samples in the stage 2 window. If a buffer of the high-rate stage 1 samples is available that exceeds the length of the stage 1 window, it is possible to perform the stage 2 FFT over a window with the same number of

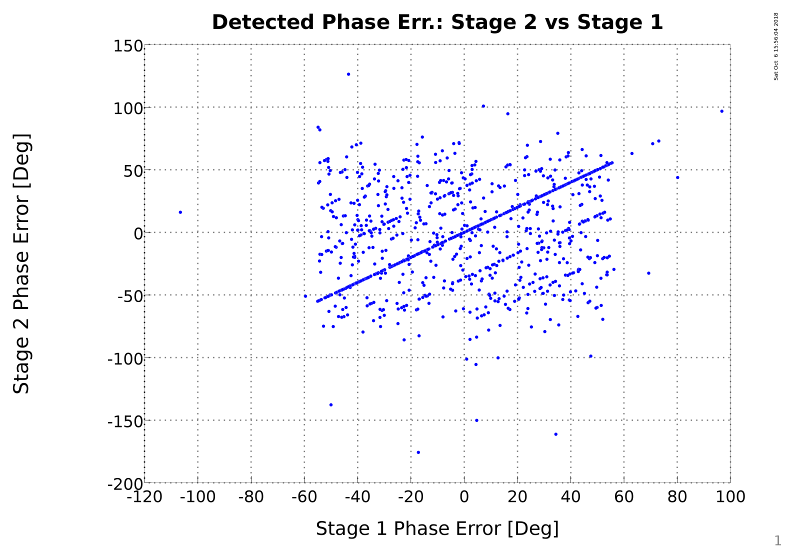


Figure 4.12: *Detected phase error of stage 1 vs stage 2. One thousand randomly sampled runs. No outliers removed.*

samples as stage 1 (assuming a sufficiently large buffer to sample from) with the necessary element stride. The histogram of the results of this subsequent experiment are show in Figures 4.15 and 4.16 for frequency and phase error detection, respectively. Again, some results with stage 1 error are so near zero as to cause stage 2 error increases over 200% were removed for clarity. Those figures show the results of a separate set of one thousand randomly sampled signals per the above specifications but bear remarkably similar results. Frequency detection error is decreased across the board, but phase detection error increases for one third of cases. Performance is not markedly improved but the cost of re-processing with the larger stage 2 sample set is that of larger memory requirements, and as the length of the buffers increases, the center-time of the window also increases, introducing lag into

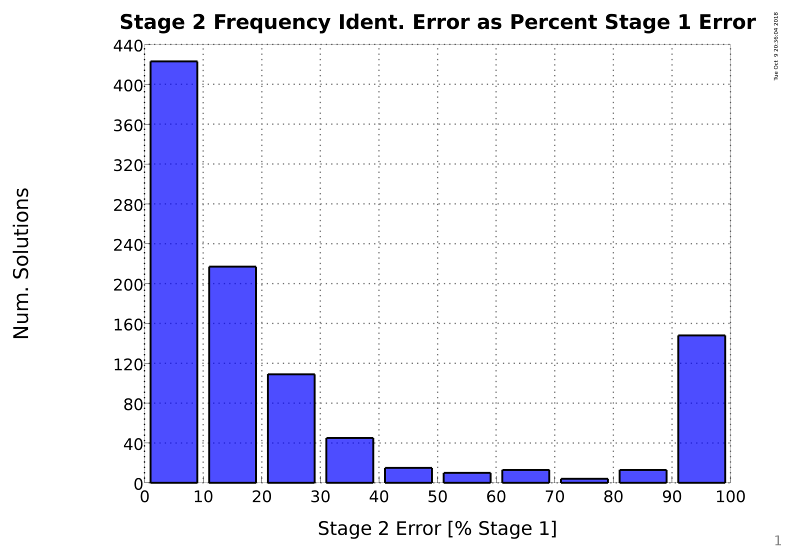


Figure 4.13: *Histogram of the stage 2 relative frequency detection error as a percentage of the stage 1 error when stage 2 windows are always sub-sets of the stage 1 window elements. One thousand randomly sampled runs. [Three runs* (0.3% *of runs) have been removed because of an increase in error greater than* 200% *for clarity.]*

the results.

Therefore, in the absence of fore-knowledge of the target frequencies, it is concluded that sampling the signal at ultra-high frequencies to better gauge the proper subsampling frequency has no positive effect on the detection of the dominant frequency or phase signal in a noisy measurement of multiple sinusoids. Whereas the choice for the sampling frequency of the commanded Propulsion (PROP) torques or MANIP joint torques will be driven by controller bandwidth requirements and system dynamics, the choice of the FTS sampling frequency is a free parameter. Given the results discussed above, it will therefore not be

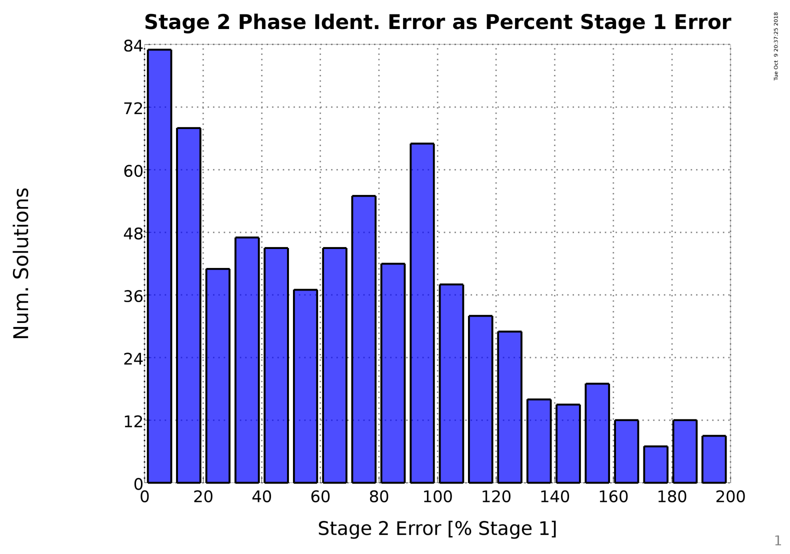


Figure 4.14: *Histogram of the stage 2 phase detection error as a percentage of the stage 1 error when stage 2 windows are always sub-sets of the stage 1 window elements. One thousand randomly sampled runs.*

necessary to over-sample the FTS measurements, and a lower rate will be sufficient.

4.2 Series Length Selection

This section illustrates the importance of the length of the acfft window on detection of dominant frequencies. The length of the FFT window, N_{fft}, determines the width of the output bins and also dictates the time it takes to record a signal at a given sampling frequency. The length of the FFT window series must be chosen but there are competing factors in its selection. A shorter window allows for a faster response to potentially changing signals, while, on the other hand, a longer window provides greater noise rejection and finer

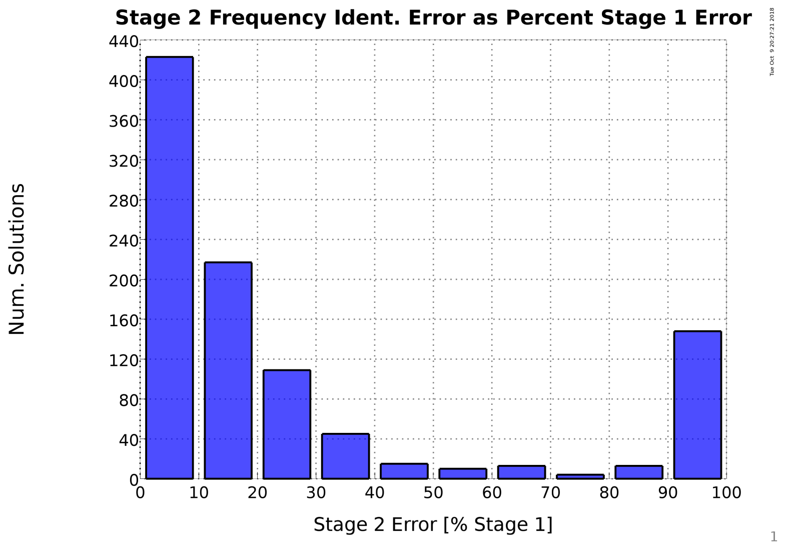


Figure 4.15: *Histogram of stage 2 frequency detection error as a percentage of stage 1 error when stage 2 windows all have the same number of samples as the stage 1 windows. One thousand randomly sampled runs. Outliers above 200% removed.*

frequency resolution because the number of frequency bins is equal to $\frac{N_{FFT}}{2} + 1$ and the resolution of each bin is equal to $\left(\frac{F_s}{2}\right) / \left(\frac{N_{FFT}}{2} + 1\right)$.

To investigate the impact of the FFT series window length, a data series with sinusoids of four frequencies, listed below, and additive zero-mean Gaussian noise was generated with F_s of 500 Hz.

1. $f_1 = 200$ Hz, amplitude $a_1 = 10$, noise $p_1 = 0.75$
2. $f_2 = 100$ Hz, amplitude $a_2 = 7$, noise $p_2 = 0.75$
3. $f_3 = 33$ Hz, amplitude $a_3 = 5$, noise $p_3 = 0.75$

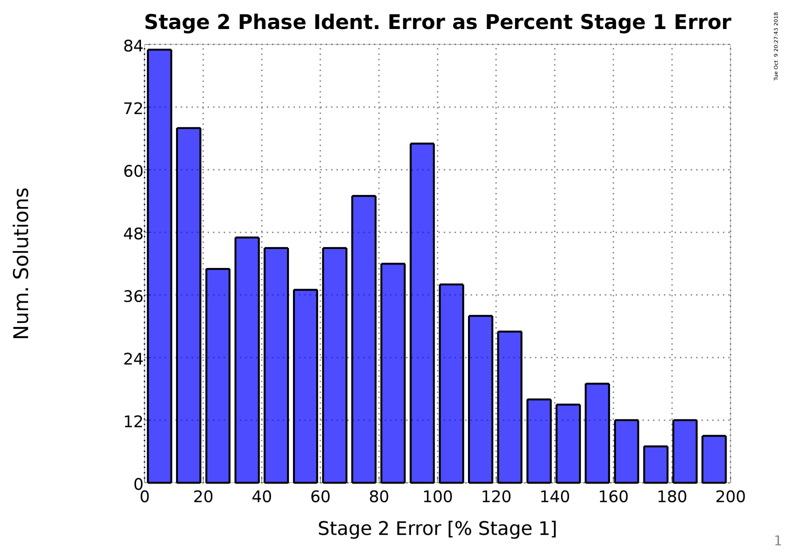


Figure 4.16: *Histogram of stage 2 phase detection error as a percentage of the stage 1 error when stage 2 windows all have the same number of samples as the stage 1 windows. One thousand randomly sampled runs. Outliers above 200% removed for clarity.*

4. $f_4 = 10$ Hz, amplitude $a_4 = 2$, noise $p_4 = 0.75$

$$s = \sum_i a_i \left(\sin\left(f_i\right) + p_i\, \mathcal{N}\left(0, 1\right)\right) \tag{4.1}$$

Window lengths of 128 (i.e. 2^7) samples through 262,144 (i.e. 2^{18}) samples, equivalent, at this sampling frequency, to windows of 0.3 seconds through 8.7 minutes are processed below. Figure 4.17 shows the impact of the series length when processed by MATLAB's FFT routine. The strongest frequency is discernable from the noise with windows as small as 128 samples (0.3 seconds), but to discern the smallest (by amplitude) frequency signal, at 20% of the greatest signal, requires over 4 seconds of data, or 2048 samples. If the noise of each signal is reduced to 25% of their amplitudes, then the smallest signal, by amplitude, can be discerned above the noise with a series of just 256 samples (0.512 seconds at 500 Hz), as Figure 4.18 shows.

The final choice of the FFT series window length will depend on the expected detector noise levels and the dynamic range of the signals required to be detected for each subsystem.

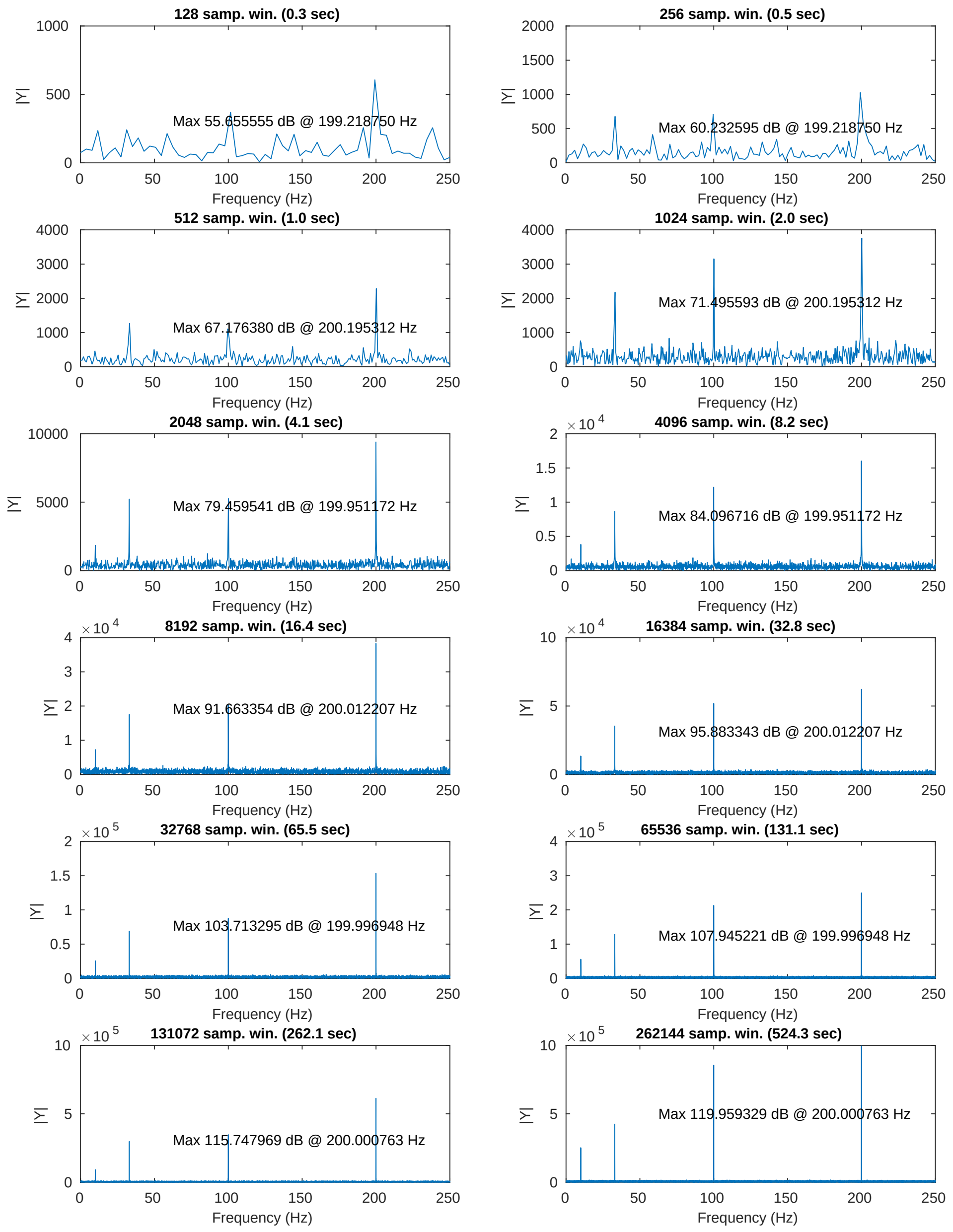


Figure 4.17: *A comparison of FFT series length performance; 4 sinusoidal signals each with 75% noise.*

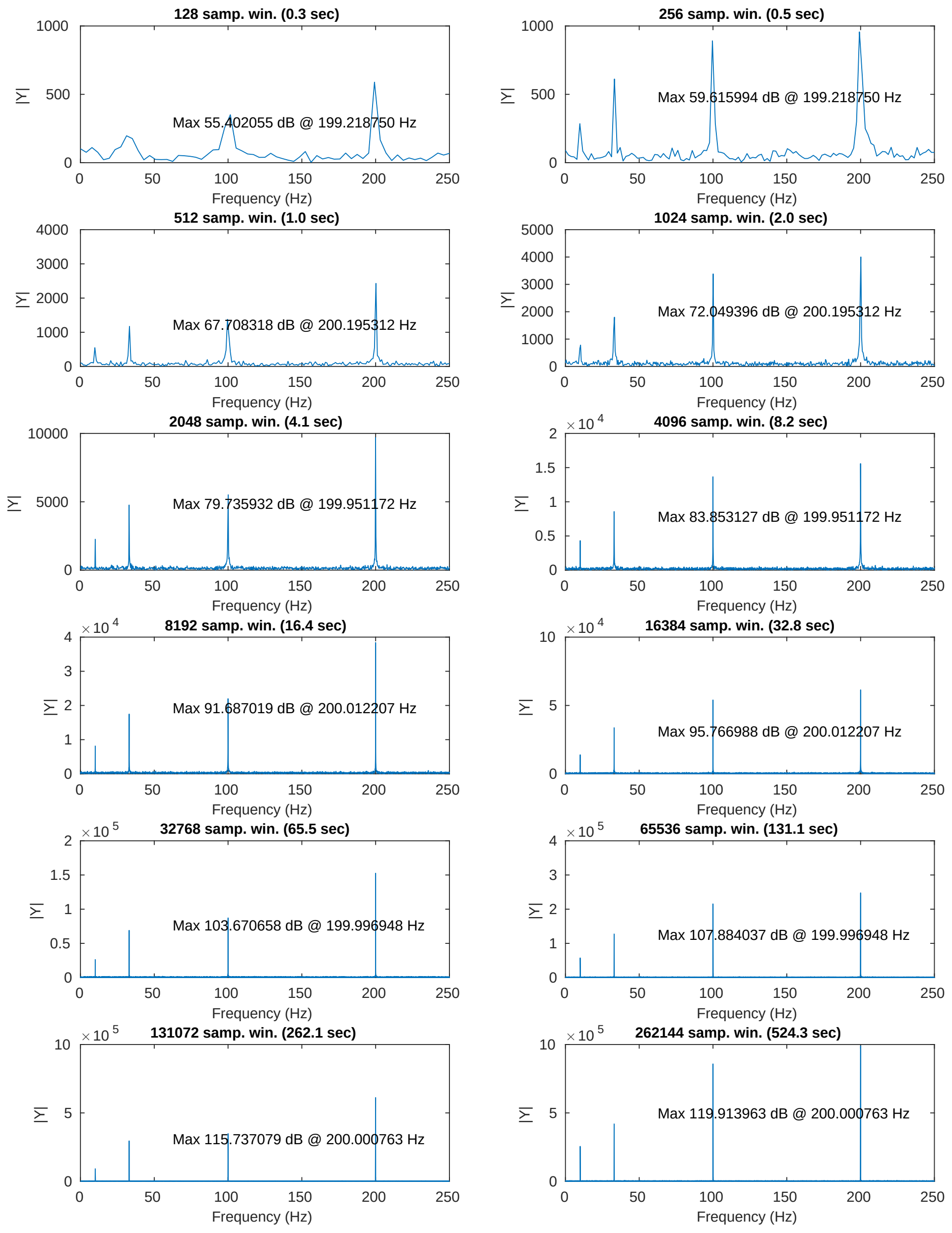


Figure 4.18: *A comparison of FFT series length performance; 4 sinusoidal signals each with 25% noise.*

4.3 Automated Peak Detection

The output of an FFT is a signal in the frequency domain where the relative strengths of each sinusoidal component of the time-series signal can be discerned by their relative amplitude in the frequency domain. It is easy for the human eye to distinguish peaks in this data, but determining them numerically requires several steps. The approach utilized here to detect peaks in the FFT output is detailed in Algorithm 1 and graphically as flowchart in Figure 4.19. Depending on the signal source, before being processed by the FFT algorithm, the mean of the time series data may or may not be subtracted from the data. First, the FFT output is normalized by its maximum value to put the amplitudes in the domain of zero to one. This is helpful for comparisons between frequency domain data of differing units, e.g. an FTS senses force in Newtons but torques in Newton-meters.

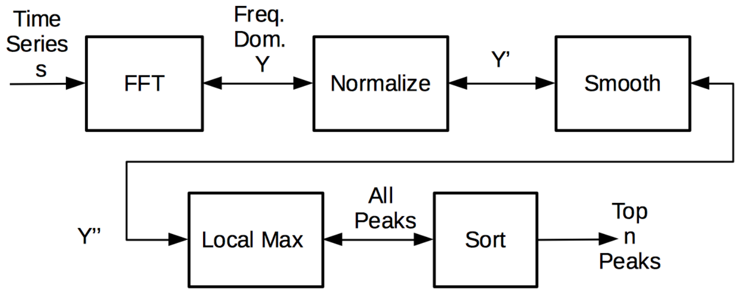


Figure 4.19: *The peak detection process flow.*

After normalization, a sliding-window averaging smoother is used to reduce the noise in the frequency domain signal. The choice of the span of the sliding window, n_{smooth}, depends

```
**Data:** Sliding window raw signal, $s$; number of peaks to return, $n_p$; FFT bin center
        frequency labels, $f_{bin}$; sliding window smoother span, $n_{smooth}$; single-sided
        inflection point search neighborhood, $n_{infl}$; single-sided local maximum
        search neighborhood, $n_{lmax}$; sampling frequency, $f_s$
**Result:** $n_p$ or less peaks are identified
1  $Y =$fft$(s, f_s)$;
2  $absY =$abs$(Y)$/max(abs$(Y)$);
3  $phi =$angle$(Y)$;
4  $[sY, sf_{bin}] =$smooth$(absY, n_{smooth})$;
5  $[infIdx] =$inflectSearch$(sY, n_{infl})$;
6  $[mIdx] =$localMaxSearch$(sY, infIdx, n_{lmax})$;
7  $[cIdx] =$topSearch$(sY, mIdx, n_p)$;
8  $peakFreqs = sf_{bin}(cidx)$;
9  $peakAmps = sY(cIdx)$;
10 $peakPhases = phi(cIdx)$;
11 $n_{found} =$length$(cidx)$;
```
Algorithm 1: FFT peak identification algorithm

on the number of frequency bins in the signal. The smoothing routine is detailed below in Algorithm 2. Next, peaks are determined by sign changes in the slope of the signal. The slope is calculated with a simple back-difference of the frequency domain signal, as described in Algorithm 3. The indexing here is chosen so that for a given index of the back-difference it is the slope to the *right* of the same index in the original signal. That is, given a frequency domain signal x and its back-difference dx, the slope to the left of $x(k)$ is $dx(k-1)$ and the slope to the right of $x(k)$ is $dx(k)$.

```
**Function** $[sY, sf_{bin}]$ =**smooth**$(Y, f_{bin}, n_{smooth})$;
$tail = (n_{smooth} - 1)/2$;
$len$ =**length**$(Y)$;
**for** $k = (tail + 1)$ *to* $(len - tail)$ **do**
    $sum = 0$;
    **for** $j = 1$ *to* $n_{smooth}$ **do**
        $sum = sum + Y(j)$;
    **end**
    $sY(k - tail) = sum/n_{smooth}$;
    $sf_{bin}(k - tail) = f_{bin}(k)$;
**end**
```

Algorithm 2: Sliding-window smoothing algorithm (note: first array index is 1)

```
Function[$d$] =backDiff($sig, len$);
for $k = 1$ to $len - 1$ do
    $d(k) = sig(k+1) - sig(k)$;
end
```
Algorithm 3: Back-difference algorithm (note: first array index is 1)

The inflection point search inspects the slopes within a neighborhood, n_{infl}, of a given point in order to select that point as an inflection point, i.e. n_{infl} slopes to the left of the point must be positive and n_{infl} slopes to the right must be negative. Here the term *neighborhood* of size n around sample k means all samples between sample $k - n$ and $k + n$. See Algorithms 4, 5 and 6, below.

Further, given the choice of n_{infl}, and the noisiness of the signal (even post-smoothing), there may be several inflection points in close proximity to each other in the frequency domain. The human eye might group these together into one growing peak, but numerically peaks within a larger growing peak can be filtered out by ensuring that a given inflection point is also a local maximum within some neighborhood. For each inflection point found, a local maximum search is performed to check that the given point is the maximum within n_{lmax} points to the left and right. Note that in the case of two adjacent points having the same amplitude in the frequency domain, in this formulation the slope will be zero for the left-most point of the pair and that point is considered to be a local maximum, whereas the

```
Function [$inflectFound$]= leftSearch($rightslope, len, n_{infl}, k$);
$inflectFound = 1$;
for $j = 1$ to $n_{infl}$ do
    if $k - j < 0$ then
        if $n==1$ then
            $inflectFound = 0$;
            break;
        end
    else
        if $rightSlope(k - j) < 0$ then
            $inflectFound = 0$;
        end
    end
end
```

Algorithm 4: Search slopes to the left of the point of interest to ensure that they are rising.

```
1  Function[$inflectFound$] =rightSearch($rightSlope, len, n_{infl}, k$);
2  $inflectFound = 1$;
3  for $j = 0$ to $(n_{infl} - 1)$ do
4      if $(k + j) > (len - 2)$ then
5          break;
6      end
7      if $rightSlope(k + j) > 0$ then
8          $inflectFound = 0$;
9      end
10 end
```

Algorithm 5: Search slopes to the right of the point of interest to ensure that they are falling.

right-most point in the pair will not. This prevents missing a peak due to a plateau of two equal points and prevents anomolous detection of two adjacent points as two independent peaks.

Finally, the relative amplitude of the peaks that survived the local maximum search are sorted and the top n_p are returned. This is described in Algorithm 8. The solution of the sorting of each remaining peak includes the center frequency of the matching FFT output bin, the normalized amplitude and the phase.

To illustrate the steps of the peak identification process, two examples are included below. The first example is a window of samples from the actuated forces from the propulsion module during a detumble maneuver. The time series signal included 128 samples collected at the module frequency of 10 Hz. Figure 4.20 shows the logarithm of the FFT output at the beginning of the peak identification process. Figure 4.21 shows the same signal after it has been normalized by its maximum value. Figure 4.22 shows this signal after normalization and smoothing with a sliding window of 5 samples. Figure 4.23 shows this signal after normalization and smoothing and identifies preliminary peak candidates with red circles after an inflection point search with a neighborhood of 1 sample to either side. Many candidate peaks exist and the DC peak is ignored, as desired. Figure 4.24 shows this signal after normalization and smoothing with the peaks identified after removing several peaks utilizing the local maximum search in a neighborhood of 2 samples to either side. The five highest relative amplitude peaks are summarized in Table 4.1.

```
Function[$infIdx$] =inflectSearch($sY, n_{infl}$);
$nFound = 0$;
$rightSlope = backDiff(sY,$length$(sY))$;
$k = 1$;
while $k <$lenght$(rightSlope)$ do
    if leftSearch$(rightSlope(k), n_{infl})$ AND rightSearch$(rightSlope(k), n_{infl})$ then
        $nFound = nFound + 1$;
        $infIdx(nFound) = k$;
        $k = k + n_{infl}$;
    else
        $k = k + 1$;
    end
end
```

Algorithm 6: Inflection-point search algorithm (note: first array index is 1)

```
Function[$mIdx$] = localMaxSearch($sY, idx, n_{lmax}$);
$nFound = 0$;
for $k = 1$ to $length(idx)$ do
    $isMax = 1$;
    for $p = 1$ to $n_{lmax}$ do
        if $((idx(k) - p) > 0)$ AND $(sY(idx(k) - p) >= sY(idx(k)))$ then
            $isMax = 0$;
            break;
        end
    end
    for $p = 1$ to $n_{lmax}$ do
        if $((idx(k) + p) <= length(sY))$ AND $(sY(idx(k) + p) > sY(idx(k)))$ then
            $isMax = 0$;
            break;
        end
    end
    $nFound = nFound + 1$;
    $mIdx(nFound) = k$;
end
```

Algorithm 7: Search to check if a given point is the local maximum within the neighborhood of n_{lmax} samples.

```
1 Function[$cIdx$] =topSearch($sY, mIdx, n_p$);
2 [$decsY, dIdx$] =sort($sY(mIdx)$,'descending');
3 $idx = 1 : n_p$;
4 $cIdx = mIdx(dIdx(idx))$;
```

Algorithm 8: Find the indices of the top n_p values of the subset $mIdx$ of signal sY.

Table 4.1: *The frequency, amplitude and phase of the five largest peaks as identified in the PROP actuated X force signal.*

Rank	Bin	Freq. [Hz]	Rel. Amp.	Phase [Deg]
1	26	2.1	-1.53	-132.3^o
2	41	3.3	-1.58	84.8^o
3	13	1.1	-1.61	119.8^o
4	52	4.1	-1.67	-93.2^o
5	56	4.5	-1.77	-14.2^o

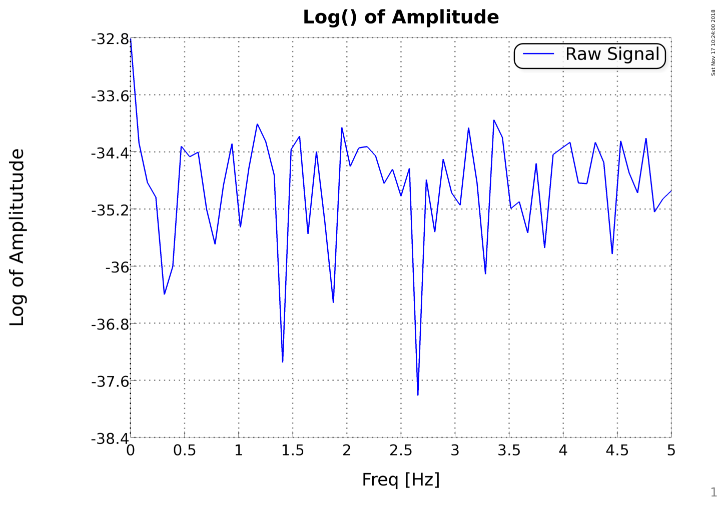


Figure 4.20: *Log of the FFT output raw signal of the actuated forces in the X axis from the PROP module during a detumble maneuver.*

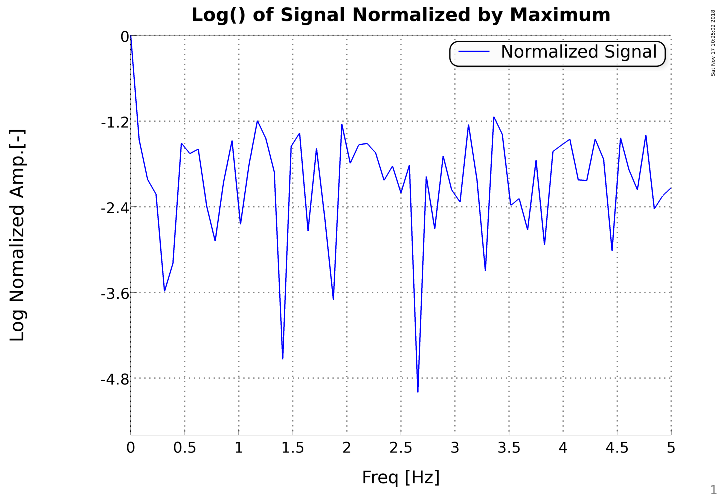


Figure 4.21: *Log of the FFT output signal of the actuated forces in the X axis from the PROP module during a detumble maneuver, normalized by its maximum value.*

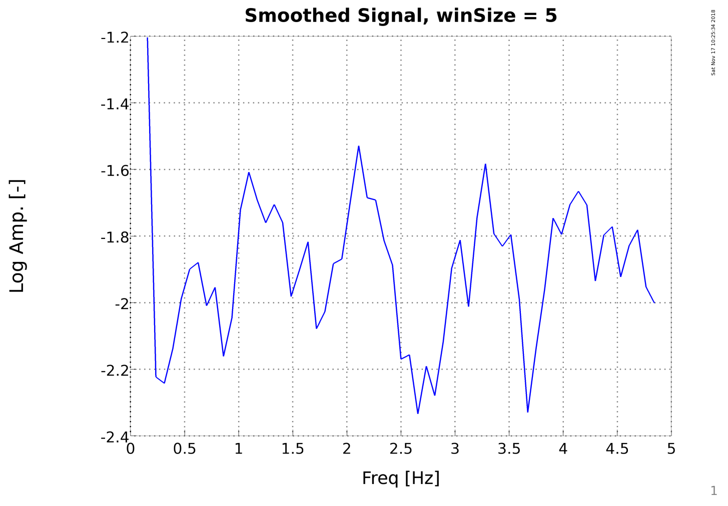


Figure 4.22: *Log of the FFT output signal of the actuated forces in the X axis from the PROP module during a detumble maneuver, after normalization and smoothing with a sliding window of 5 points.*

The second example is a window of samples from the measured forces from the FTS module during a detumble maneuver. The time series signal included 4096 samples collected at the module frequency of 500 Hz, a 8.192 second window. Figure 4.25 shows the logarithm of the raw FFT output at the beginning of the peak identification process. Figure 4.26 shows the same signal after it has been normalized by its maximum value. Figure 4.27 shows this signal after normalization and smoothing with a sliding window of 7 samples. Figure 4.28 shows this signal after normalization and smoothing and identifies preliminary peak candidates with red circles after an inflection point search with a neighborhood of 1 sample to either side. Figure 4.29 shows this signal after normalization and smoothing with

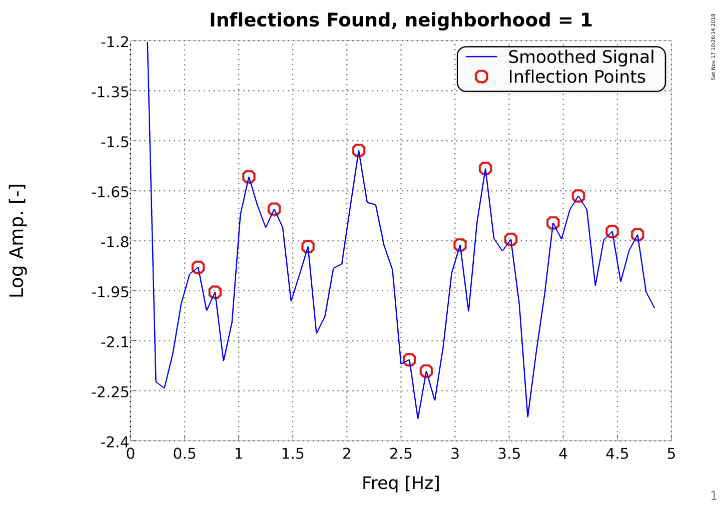


Figure 4.23: *Log of the FFT output signal of the actuated forces in the X axis from the PROP module during a detumble maneuver, after normalization, smoothing and inflection point identification with a inflection neighborhood of 1 sample to either side.*

the peaks identified after removing several with the local maximum search in a neighborhood of 12 samples to either side. The five highest relative amplitude peaks are summarized in Table 4.2.

4.4 Conclusion

This chapter has demonstrated three topics related to automated signal detection in time-series data using an FFT. First, phase detection is not improved by re-processing digital signals at varying window parameters, indicating the driving parameters for dominant frequency detection are the length of the processing window and the sample rate. Secondly,

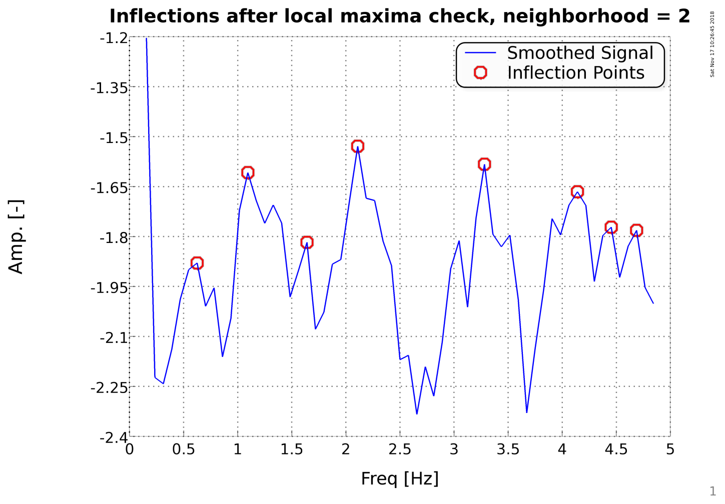


Figure 4.24: *Log of the FFT output signal of the actuated forces in the X axis from the PROP module during a detumble maneuver. Peaks are identified with red circles after normalization, smoothing, an inflection search and a local maximum search with a neighborhood of 2 samples to either side.*

Table 4.2: *The frequency, amplitude and phase of the five largest peaks as identified in the FTS measured Z axis force signal.*

Rank	Bin	Freq. [Hz]	Rel. Amp.	Phase [Deg]
1	26	3.4	-1.5	151.9^o
2	3	0.6	-1.8	70.3^o
3	223	27.5	-5.5	106.3^o
4	251	30.9	-6.6	95.0^o
5	1886	230.5	-8.2	172.9^o

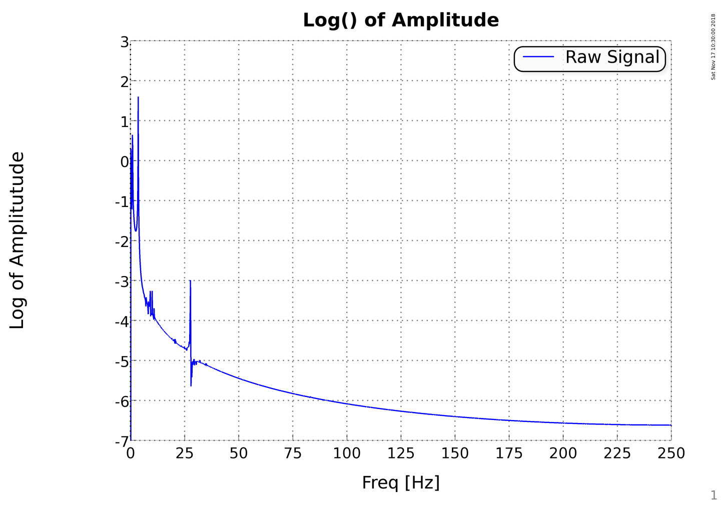


Figure 4.25: *Logarithm of the FFT output raw signal from the measured FTS forces in its Z direction during a detumble maneuver.*

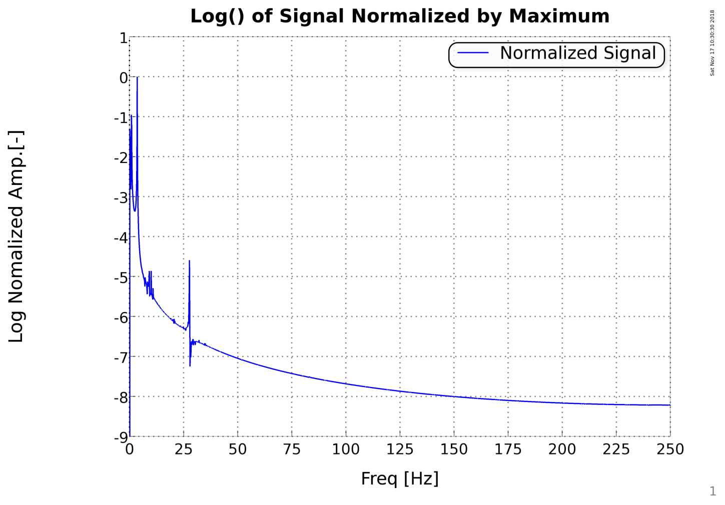


Figure 4.26: *Log of the FFT output signal from the measured FTS forces in its Z axis during a detumble maneuver, normalized by its maximum value.*

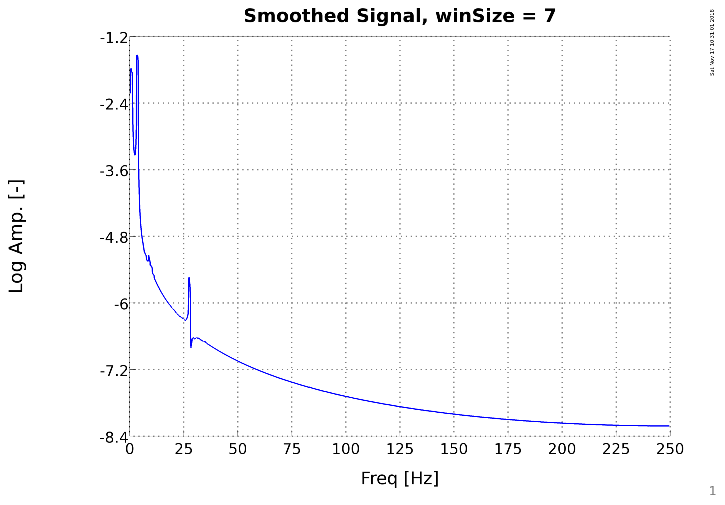


Figure 4.27: *Log of the FFT output signal from the measured FTS forces in its Z axis during a detumble maneuver, after normalization and smoothing with a sliding window of 7 points.*

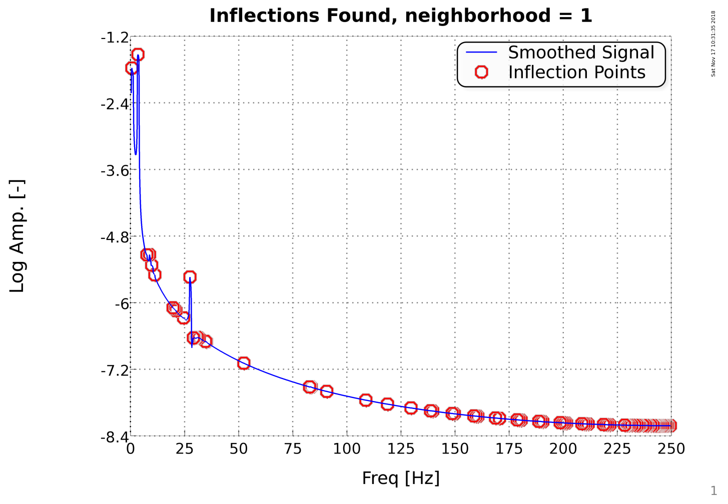


Figure 4.28: *Log of the FFT output signal from the measured FTS forces in its Z axis during a detumble maneuver, after normalization, smoothing and inflection point identification with a inflection neighborhood of 1 sample to either side.*

it determined the levels of noise that are allowable depending on the length of the processing window. Finally, a method of automated multiple peak detection in the presence of noise was presented. In the next chapter, the application of the above FFT processing and the above nonlinear attitude controller will be used to demonstrate their combined ability to reduce vibrations in a client's appendage while performing a detumble maneuver.

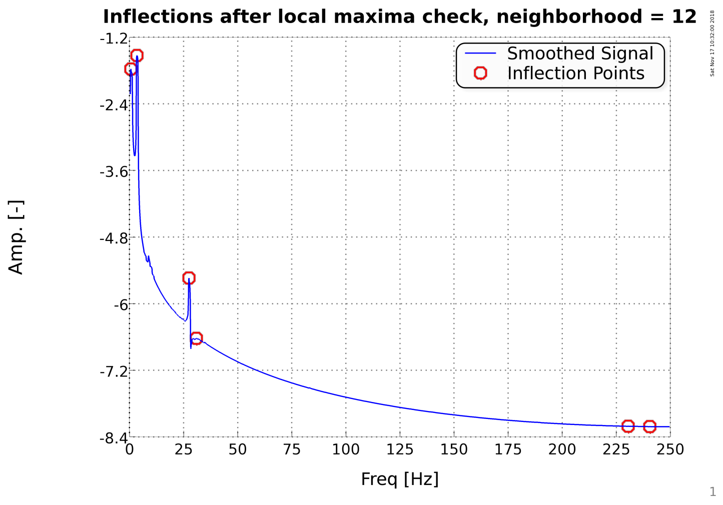


Figure 4.29: *Log of the FFT output signal from the measured FTS forces in its Z axis during a detumble maneuver. Peaks are identified with red circles after normalization, smoothing, an inflection search and a local maximum search with a neighborhood of 12 samples to either side.*

Chapter 5: 6-DOF Orbital Simulation

This chapter will outline the components of the 6-DOF orbital simulation incorporating the previously discussed innovations and present how they can be utilized to reduce vibrations in client appendages during detumbling. This chapter will also discuss how the target frequencies are identified, and how manipulator trajectories are created. Then the resulting reduction in client appendage motion will be compared to detumbling maneuvers without this intervention. First, the Freespace simulation environemnt will be described.

5.1 Freespace Simulation Environment Overview

Full 6-DOF simulations of detumbling grappled satellites were done in the Freespace simulation environment, an orbital and environmental dynamics simulation created by the NASA's Goddard Space Flight Center (GSFC). Freespace has been in development and use at GSFC since prior to 2005. [56] Freespace was used for the design and development of National Aeronautics and Space Administration (NASA)'s 2009 Relative Navigation System (RNS) experiment on STS-125 for HST Servicing Mission 4 (SM4). [57] Freespace was used in

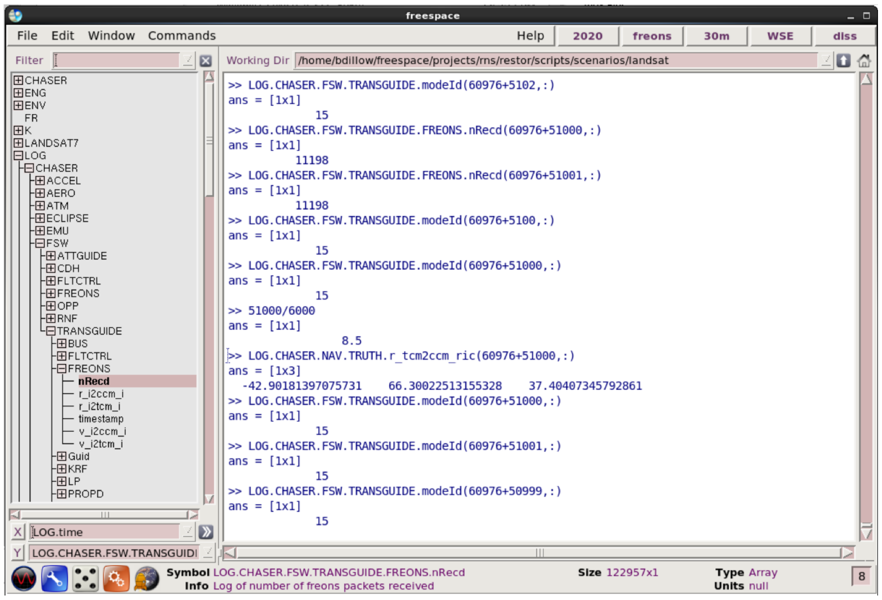
Hardware In the Loop (HIL) testing for the NASA's Satellite Servicing Projects Directorate (SSPD) Argon test campaign, where it performed guidance, navigation and control functions; integrated the dynamics; and issued commands to a Fanuc robotic manipulator that was holding the computer vision sensor package. [58] It was also used in the development of NASA's Magnetosphere Multiscale Mission (MMS) mission [59] [60] [61], launched in 2015, and several other missions yet to publish.

Freespace is a high-fidelity mission simulation and design tool. It includes an integrator for the processing of multi-body system dynamics (i.e. kinematic chains of bodies and / or gravitational effects of large celestial objects). It also features a scripting interface similar to MATLAB for simulation configuration and subsequent processing of logged simulation results, i.e. a scripting interface with MATLAB compatible syntax, navigable data tree structures and robust 2D plotting tools. It also features a visualization component so that 3D CAD models of systems involved can be represented in their dynamic and relative states, i.e. the user can see how a multi-body system moves with respect to other bodies within that system in OpenGL environments. Figure 5.1 shows the main Freespace GUI interface with its data tree browser on the left and the scripting console on the right. Figure 5.2 shows the GUI of the Freespace Engine which is the front-end to the integrator. Some figures within this dissertation are captured from within Freespace's visualizer, as indicated. Furthermore, flight-like software written in C can be interfaced directly with the integrator to test algorithms before integration to spacecraft. A module written for Freespace affects a simulation through two categories of interaction: its Real Time (RT) function and its Discrete Real Time (DRT) function. Effects like the forces and torques of gravity, SRP or aerodynamic drag, as well as actuation forces and torques of thrusters or reaction wheels are realized through the RT functions as the integrator processes the simulation. The DRT functions are most easily thought of as the discrete digital logic that is performed to decide how much force or torque to apply in control, or when an estimator or filter will update. The DRT functions are called at constants rates to process data, whereas the RT functions will be called at varying (small) timesteps that the integrator deems necessary to achieve accurate integration results. This is mentioned to clarify the distribution of labor (and computation) that can be seen in the source code of the appendices: digital logic within DRT functions; application of forces and torques within the RT logic.

5.2 Freespace Setup

In this study, we have the following standard modules active in Freespace:

- SOLSYS : it calculates locations of necessary celestial bodies;
- SUN : it calculates the location of the sun relative to Earth;
- GRAV : it calculates the gravitational impact on each body of the multi-body system; and,



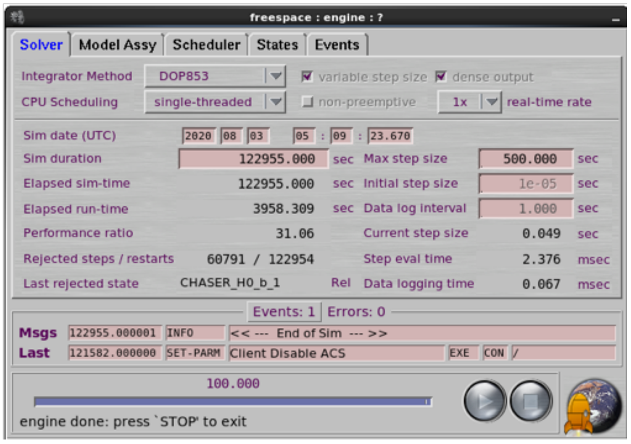
Working Dir /home/bdillow/freespace/projects/rns/restor/scripts/scenarios/landsat

```
>> LOG.CHASER.FSW.TRANSGUIDE.modeId(60976+5102,:)
ans = [1x1]
             15
>> LOG.CHASER.FSW.TRANSGUIDE.FREONS.nRecd(60976+51000,:)
ans = [1x1]
          11198
>> LOG.CHASER.FSW.TRANSGUIDE.FREONS.nRecd(60976+51001,:)
ans = [1x1]
          11198
>> LOG.CHASER.FSW.TRANSGUIDE.modeId(60976+5100,:)
ans = [1x1]
             15
>> LOG.CHASER.FSW.TRANSGUIDE.modeId(60976+51000,:)
ans = [1x1]
             15
>> 51000/6000
ans = [1x1]
                 8.5
>> LOG.CHASER.NAV.TRUTH.r_tcm2ccm_ric(60976+51000,:)
ans = [1x3]
  -42.90181397075731    66.30022513155328    37.40407345792861
>> LOG.CHASER.FSW.TRANSGUIDE.modeId(60976+51000,:)
ans = [1x1]
             15
>> LOG.CHASER.FSW.TRANSGUIDE.modeId(60976+51001,:)
ans = [1x1]
             15
>> LOG.CHASER.FSW.TRANSGUIDE.modeId(60976+50999,:)
ans = [1x1]
             15
```

X LOG.time
Y LOG.CHASER.FSW.TRANSGUIDI

Symbol LOG.CHASER.FSW.TRANSGUIDE.FREONS.nRecd   Size 122957x1   Type Array
Info Log of number of freons packets received   Units null   8

Figure 5.1: *The Freespace workspace GUI showing variable tree browser (top left) and scripting interface console (right).*



Integrator Method DOP853   [✓] variable step size   [✓] dense output
CPU Scheduling single-threaded   ☐ non-preemptive   1x real-time rate

Sim date (UTC) 2020 08 03 05 : 09 : 23.670
Sim duration 122955.000 sec   Max step size 500.000 sec
Elapsed sim-time 122955.000 sec   Initial step size 1e-05 sec
Elapsed run-time 3958.309 sec   Data log interval 1.000 sec
Performance ratio 31.06   Current step size 0.049 sec
Rejected steps / restarts 60791 / 122954   Step eval time 2.376 msec
Last rejected state CHASER_H0_b_1 Rel   Data logging time 0.067 msec

Events: 1 Errors: 0
Msgs 122955.000001 INFO << --- End of Sim --- >>
Last 121582.000000 SET-PARM Client Disable ACS EXE CON /

100.000
engine done: press `STOP' to exit

Figure 5.2: *The Freespace Engine GUI.*

- VSD : it calculates the velocity state dynamics of each body in the multi-body systems as they are perturbed by environmental forces/torques and actuated upon by control forces/torques, as well as the forces and torques between bodies as they react to one another in the DOF that are not free to rotate or translate.

In addition, the following modules were created for this research:

- SADA : it is the controller for the solar array drive actuator on the servicing spacecraft to maintain pointing at the sun, when active;
- ARMCTRL : it is the high-level controller for the manipulators that implements the cartesian position and velocity control, as well as interpretting the FTS measurements;
- MANIP : it is the joint-level manipulator controller;
- ACS : it is the attitude control system that converts error between current sensed attitude and body rates and the desired attitude and body rates into torque commands;
- PROP : it is the module to convert desired servicer-body torque commands to individual thruster-firing on-times; and,
- FTS : it is the Force-Torque Sensor module for capturing reaction forces and torques measurements within the gripper.

Each module implements its algorithms in the C programming language and is configured at run-time with scripts written in a language similar to MATLAB and Octave. These configuration scripts use the file extension ".m", and are therefore sometimes referred to as M-scripts.

The nominal DRT rates of each module are listed in Table 5.1.

Table 5.1: *Nominal Simulation Module Rates (∞ indicates RT only modules, without periodic DRT calculations)*

Module	**Rate [Hz]**	**Module**	**Rate [Hz]**
SOLSYS	∞	SUN	∞
GRAV	∞	VSD	∞
FTS	500	MANIP	500
ARMCTRL	500	ACS	10
PROP	10	SADA	1

5.2.1 Built-in Modules

The Solar System (SOLSYS) module calculates the position and velocity of selected heavenly bodys in the solar system within the simulation. The SUN module calculates the unit vector from epoch J2000 (2000 Jan. 1.5 TD) Earth-Centered Inertial (ECI) origin to the sun for a given Julian day. It is accurate to within approximately 0.01 degrees. The algorithm is based on Jean Meeus' *Astronomical Algorithms* [62] (1998, pages 163-176).

The Gravity (GRAV) module calculates the gravitational potential experienced by each spacecraft body. The algorithm is based on the NASA GSFC and National Imagery and Mapping Agency (NIMA) Joint Geopotential Model (EGM96) with shared sperical harmonic evaluations, as described by Lemoine et al. in [63]. The Velocity State Dynamics (VSD) module computes the accelerations and velocities of the bodies within the simulation as well as any constraint forces and torques between two joined bodies, imparted by other modules such as GRAV or PROP. It supports multiple open branching kinematic chains of bodies. Connections between bodies can be single or three DOF translational (prismatic) or rotational joints, as well as fully rigid. Its derivation is similar to that in Stoneking's Newton-Euler Dynamics example for a spacecraft [64].

5.2.2 Custom Modules

5.2.2.1 FTS, MANIP, and ARMCTRL Modules

The FTS module reads the constraint forces and torques on a rigid joint between two bodies between the final actively controlled joint of the manipulator, the seventh, and the grapple fixture. Within the Velocity State Dynamics (VSD), the FTS joint is the eighth joint of the kinematic chain between the servicer's base body and the next outward body in the manipulator's kinematic chain; the FTS joint has zero DOF and therefore all reaction forces and torques required to maintain its initial relative pose between inward and outward bodies are calculated. The grapple fixture is the ninth joint in the kinematic chain from the servicer's base to the grappled client body, and its DOF vary depending on scenario configuration. The force and torque measurements are saved to a buffer and a sliding-window FFT is performed, as discussed in Chapter 4. This data is subsequently passed to the ARMCTRL module for further processing and reaction.

The MANIP module is a joint-level controller for the manipulator. It can either perform joint-space angle and rate Proportional-Integral-Derivative (PID) control, or directly execute commanded joint-space desired torques from the ARMCTRL module. In joint-space PID control, anti-windup protection are included to prevent excessive integral term reactons; the joint torque, τ_j, is computed based on deviations from commanded joint angle and rate as in Eq. 5.1, where θ_j and $\dot{\theta}_j$ are the angle and rate of joint j, respectively; and T_{MANIP} is the period of the MANIP DRT control cycle. The gains $K_{p,j}$, $K_{i,j}$, $K_{d,j}$ are the proportional, integral and derivative gains, respectively, for each joint j and are chosen on a per-joint basis. The MANIP module can also follow direct torque commands on a per-joint basis. In Joint-Hold mode, Eq. 5.1 is used to calculate the joint torques for a static desired manipulator pose, with each $\dot{\theta}_{cmd,j}$ set to zero.

$$\begin{aligned}
\theta_{err,j} &= \theta_j - \theta_{cmd,j} \\
\dot{\theta}_{err,j} &= \dot{\theta}_j - \dot{\theta}_{cmd,j} \\
\Sigma_j &= \Sigma_{j-1} + \theta_{err,j} T_{MANIP} \\
\Sigma_j &= \text{sign}(\Sigma_j)\text{min}(|\Sigma_j|, \Sigma_{max,j}) \\
\tau_j &= -K_{p,j}\theta_{err,j} - K_{i,j}\Sigma_j - K_{d,j}\dot{\theta}_{err,j}
\end{aligned} \tag{5.1}$$

The ARMCTRL module is a high-level manipulator control module that can compute cartesian-space trajectories or other reactive measures. ARMCTRL has several modes but the most important to the focus of this research is the Reactive Velocity Control mode which monitors the FFT output of the commanded forces and torques from PROP, the commanded joint torques from MANIP (or internally calculated), and sensed reaction forces and torques in the manipulator at the grapple point from the FTS module. The down-selection of FFT peaks of each DOF within each monitored module is illustrated by Figure 5.5.

5.2.2.2 ACS and PROP Modules

The ACS module contains the nonlinear attitude controller, as described in Chapter 2, that calculates servicer body torques based on attitude and rate errors. It runs at 10 Hz. The torques that it calculates to correct the attitude errors are in the servicer body frame. They are passed to the PROP.

The PROP module converts servicer body-frame torque commands to individual thruster on-time allocations. The mapping of body torques to thruster on-times introduces control errors and nonlinearities which are of interest to spacecraft control due to thruster minimum on times and thrust saturation. The PROP module may also be configured for perfect application of body torques for studies which require it. There are many ways to map body torques to thruster firing times. The simplest forms are direct mapping which directly allocate all body-frame torques in a given axis to a pre-defined set of thrusters and scale the firing based on the torque requested [40]. This method requires detailed pre-analysis of commands and a (nearly) unchanging spacecraft inertia to perform well, and is therefore not suitable for a spacecraft with reconfigurable mass properties such as in a coupled satellite configuration, where the barycenter of the coupled system is shifting more rapidly than a traditional spacecraft with fuel depletion, OR a system with high inertia uncertainties, such as when grappling a derelict client after a break-up event. At the other end of the complexity spectrum, Mixed-Integer Linear Programming (MILP) is a method that provides the ability to limit fuel usage considering multiple constraints, such as minimum thruster on times and individual duty cycle limitations [65], [66], but it can be computationally expensive and the efficiency it provides versus fuel usage and other metrics is not the focus of this research. A balance is struck with the use of pseudo-inverse thrust mapping and null-space boosting to meet minimum on time requirements, as in [67]. This method is similar to torque allocations of redundant reaction wheel spacecraft as discussed in Markley [40], but applied to uni-directional thrusters. First, the concurrent force and torque commands in the spacecraft body frame are combined into a single 6-DOF vector, as in Eq. 5.2. In the following simulations, where no ΔV will be performed and the spacecraft are purely under attitude control, the force commands, $\mathbf{f}^b_{cmd}$, will be all zeros. The body torque commands, $\boldsymbol{\tau}^b_{cmd}$, come from the ACS module.

$$\boldsymbol{\mathcal{F}}_{cmd} = \begin{bmatrix} \mathbf{f}^b_{cmd} \\ \\ \boldsymbol{\tau}^b_{cmd} \end{bmatrix} \tag{5.2}$$

The thrust mapping matrix, $\mathbf{M}$, is composed of the direction and moment arms of each thruster, with the concatenated direction, $\mathbf{d}^b_n$ and moment arm from the center of mass for each thruster, $\mathbf{a}^b_n = \mathbf{d}^b_n \times \mathbf{r}^b_{n/cm}$, comprising one comlumn, as described in Eq. 5.3, for each of n thrusters.

$$\mathbf{M} = \begin{bmatrix} \mathbf{d}^b_1 & \cdots & \mathbf{d}^b_n \\ \\ \mathbf{a}^b_1 & \cdots & \mathbf{a}^b_n \end{bmatrix} \tag{5.3}$$

Once $\mathbf{M}$ is constructed as an 6-by-n matrix, the individual thruster on times are calculated with Eq. 5.4, where the † operation again indicates the Moore-Penrose pseudo-inverse.

$$\boldsymbol{\tau}_{des} = (\mathbf{M})^\dagger \, \boldsymbol{\mathcal{F}}^b_{des} \tag{5.4}$$

The deficiency of this calculation is that it may produce individual components of $\boldsymbol{\tau}_{des}$ which are below the minimum thrust of a given thruster or even negative (an impossible command for uni-directional thrusters). To correct this limitation, null-space boosting is employed. Given any vector $\mathbf{v}_{null} \in NULL(\mathbf{M})$, $\mathbf{M}\mathbf{v}_{null} = \mathbf{0}$. $\boldsymbol{\mathcal{F}}_{actuated} = \mathbf{M}\boldsymbol{\tau}_{cmd}$, any relative variation of components of $\boldsymbol{\tau}_{cmd}$ that remain within the null-space of $\mathbf{M}$ will have zero effect on $\boldsymbol{\mathcal{F}}_{actuated}$. This provides the flexibility to modify $\boldsymbol{\tau}_{cmd}$ such that the entire vector is changed without changing the overall effect on the forces and torques produced on the spacecraft body. First, the null-space of the distributrion matrix is found using the SVD algorithm and then the first column-vector is used. (Any choice of column would be equally unimpactful on the actauted force and torque on the servicer body. If certain thrusters were undesirable to be used additionally, the null-space vector could be selected differently.) The selected null-space vector is designated $\mathbf{v}_{null}$. The thruster that is farthest from the minimum on thrust is designated τ_{low}. Then the final thruster allocation is defined by Eq. 5.5.

$$\boldsymbol{\tau}_{cmd} = \boldsymbol{\tau}_{des} + \mathbf{v}_{null}\left(\tau_{min} - \tau_{low}\right) \tag{5.5}$$

The resulting modified command produces the same resultant desired reaction:

$$\begin{aligned}
\boldsymbol{\mathcal{F}}^{b}_{actuated} &= \mathbf{M}\boldsymbol{\tau}_{cmd}\\
\boldsymbol{\mathcal{F}}^{b}_{actuated} &= \mathbf{M}\left(\boldsymbol{\tau}_{des} + \mathbf{v}_{null}\left(\tau_{min} - \tau_{low}\right)\right)\\
\boldsymbol{\mathcal{F}}^{b}_{actuated} &= \mathbf{M}\boldsymbol{\tau}_{des} + \mathbf{M}\mathbf{v}_{null}\left(\tau_{min} - \tau_{low}\right)\\
\boldsymbol{\mathcal{F}}^{b}_{actuated} &= \mathbf{M}\boldsymbol{\tau}_{des} + \mathbf{0}
\end{aligned}$$

Once all thrusters are assured to be above the minimum thrust requirements or not firing, the maximum thrust condition is checked. If any individual thruster's thrust exceeds the maximum capability, the command is clipped and the difference between desired command and actuated command is absorbed as actuation error. The impact of this clipping actuation error can be limited by ensuring the commanded torques and forces in the body frame are limited to remain within the total capability of the whole system of thrusters and the commands' directions preserved, i.e. if $\|\mathbf{f}^b_{des}\| > f_{max}$ or $\|\boldsymbol{\tau}^b_{des}\| > \tau_{max}$, the following scaling is applied:

$$\mathbf{f}^b_{cmd} = \frac{\mathbf{f}^b_{des}}{\|\mathbf{f}^b_{des}\|} f_{max}$$
$$\boldsymbol{\tau}^b_{cmd} = \frac{\boldsymbol{\tau}^b_{des}}{\|\boldsymbol{\tau}^b_{des}\|} \tau_{max}$$

In this example spacecraft, twenty four thrusters were employed, four per servicer face. Nominal thruster positions are listed in Table 5.2 and nominal thruster directions are listed in Table 5.3.

Number	**ID**	**Pos. X**	**Pos. Y**	**Pos. Z**
1	ZP1	-0.95	0.95	-3.0
2	ZP2	-0.95	-0.95	-3.0
3	ZP3	0.95	-0.95	-3.0

Number	**ID**	**Pos. X**	**Pos. Y**	**Pos. Z**
4	ZP4	0.95	0.95	-3.0
5	ZM1	-0.95	0.95	0
6	ZM2	0.95	0.95	0
7	ZM3	0.95	-0.95	0
8	ZM4	-0.95	-0.95	0
9	YM1	-0.95	1.0	-0.05
10	YM2	0.95	1.0	-0.05
11	YM3	0.95	1.0	-2.95
12	YM4	-0.95	1.0	-2.95
13	YP1	-0.95	-1.0	-0.05
14	YP2	0.95	-1.0	-0.05
15	YP3	0.95	-1.0	-2.95
16	YP4	-0.95	-1.0	-2.95
17	XM1	1.0	-0.95	-0.05
18	XM2	1.0	0.95	-0.05

Number	**ID**	**Pos. X**	**Pos. Y**	**Pos. Z**
19	XM3	1.0	0.95	-2.95
20	XM4	1.0	-0.95	-2.95
21	XP1	-1.0	-0.95	-0.05
22	XP2	-1.0	0.95	-0.05
23	XP3	-1.0	0.95	-2.9
24	XP4	-1.0	-0.95	-2.95

Table 5.2: *Nominal thruster positions on servicer spacecraft. All positions are in meters.*

Number	**ID**	**Dir. X**	**Dir. Y**	**Dir. Z**
1	ZP1	0	0	0
2	ZP2	0	0	0
3	ZP3	0	0	0
4	ZP4	0	0	0
5	ZM1	180	0	0
6	ZM2	180	0	0

Number	**ID**	**Dir. X**	**Dir. Y**	**Dir. Z**
7	ZM3	180	0	0
8	ZM4	180	0	0
9	YM1	0	-90	90
10	YM2	0	-90	90
11	YM3	0	-90	90
12	YM4	0	-90	90
13	YP1	0	-90	-90
14	YP2	0	-90	-90
15	YP3	0	-90	-90
16	YP4	0	-90	-90
17	XM1	0	90	0
18	XM2	0	90	0
19	XM3	0	90	0
20	XM4	0	90	0
21	XP1	0	-90	0

Number	ID	Dir. X	Dir. Y	Dir. Z
22	XP2	0	-90	0
23	XP3	0	-90	0
24	XP4	0	-90	0

Table 5.3: *Nominal thruster directions. All directions are Euler angles about the body axes in Degrees.*

The individual thrusters modeled here are throttled to the appropriate duty-cycle over the command interval between their minimum and maximum thrust per command interval, as opposed to modeling the duty cycle as 100% of the maximum thrust for some portion of the command cycle between the minimum on time and the command cycle period. After the commanded torques are mapped to individual thrusters based on the knowledge of the thruster directions and moment arms, commanded per-thruster thrusts are passed back through true the distribution matrix (i.e. without knowledge error), $\mathbf{M}$, to calculate the final actuated forces and torques for that control cycle, including the impact of that knowledge error. These actuated forces and torques are stored to a circular buffer and a sliding window FFT is used to determine the (at most) top five non-DC peak frequencies in each DOF. The parameters for the PROP module FFT are detailed in Table 5.4. The sampling frequency is a function of the element stride and the frequency of the PROP module itself, as discussed in Chapter 4. These parameters are derived heuristically. The parameters for all six DOF are chosen to be the same.

Table 5.4: *PROP module FFT parameters.*

Parameter	**Value**
Sampling Frequency, F_s	10 Hz
Window Length, n_{win}	64
Window Stride, s_{win}	10
Element Stride, s_{elem}	1
Max Num. Peaks, n_{peaks}	5
Smoothing Span, s_{smooth}	3
Inflection Point Neighborhood, n_{infl}	1
Local Maxima Neighborhood, n_{locMax}	3

The impact of the thruster modeling can be seen in the individual thruster duty cycles. The example below will be examined in more detail further down in Chapter 5, described as the MMS-like tumble scenario. Initally, the coupled system is tumbling a rate of $(0, 0, 18)$ degrees per second and the rate nulling is attempted. Figure 5.3 shows the thruster on-times as a percentage per command cycle on the range of 0 (fully off) to 1 (fully on). As described above, in the Servicer body frame,

- thrusters 1 through 4 apply +Z force,
- thrusters 5 through 8 apply -Z force,
- thrusters 9 through 12 apply -Y force,
- thruster 13 through 16 apply +Y force,
- thruster 17 through 20 apply -X force, and,
- thrusters 21 through 24 apply +X force.

It is clear from figure 5.3 that both the -Z and +Z thrusters are on 100% for almost the first 90 seconds of the detumble maneuver. This is an artifact of the thruster mapping algorithm not trying to be fuel efficient. Pairs of thrusters that have equivalent maximum thrust but face in opposite directions produce zero net force or torque when fired simultaneously at the same duty cycle. Another way to look at the impact of the thruster mapping algorithm is to look at the histogram of thruster duty cycles. Figure 5.4 shows the histogram of the thruster duty cycles for the same simulation as figure 5.3, with duty cycles broken down in to bin increments of 10%. This shows the number of command cycles a given thruster was commanded in that particular duty cycle range. For the given configuration of this example, the minimum on time fell in the range between 10 and 20 % of a command cycle; therefore, all commands in the 0 to 10 % range represented fully off command periods. From this it is clear that the vast majority of commands for the pseudo-inverse thruster mapping with

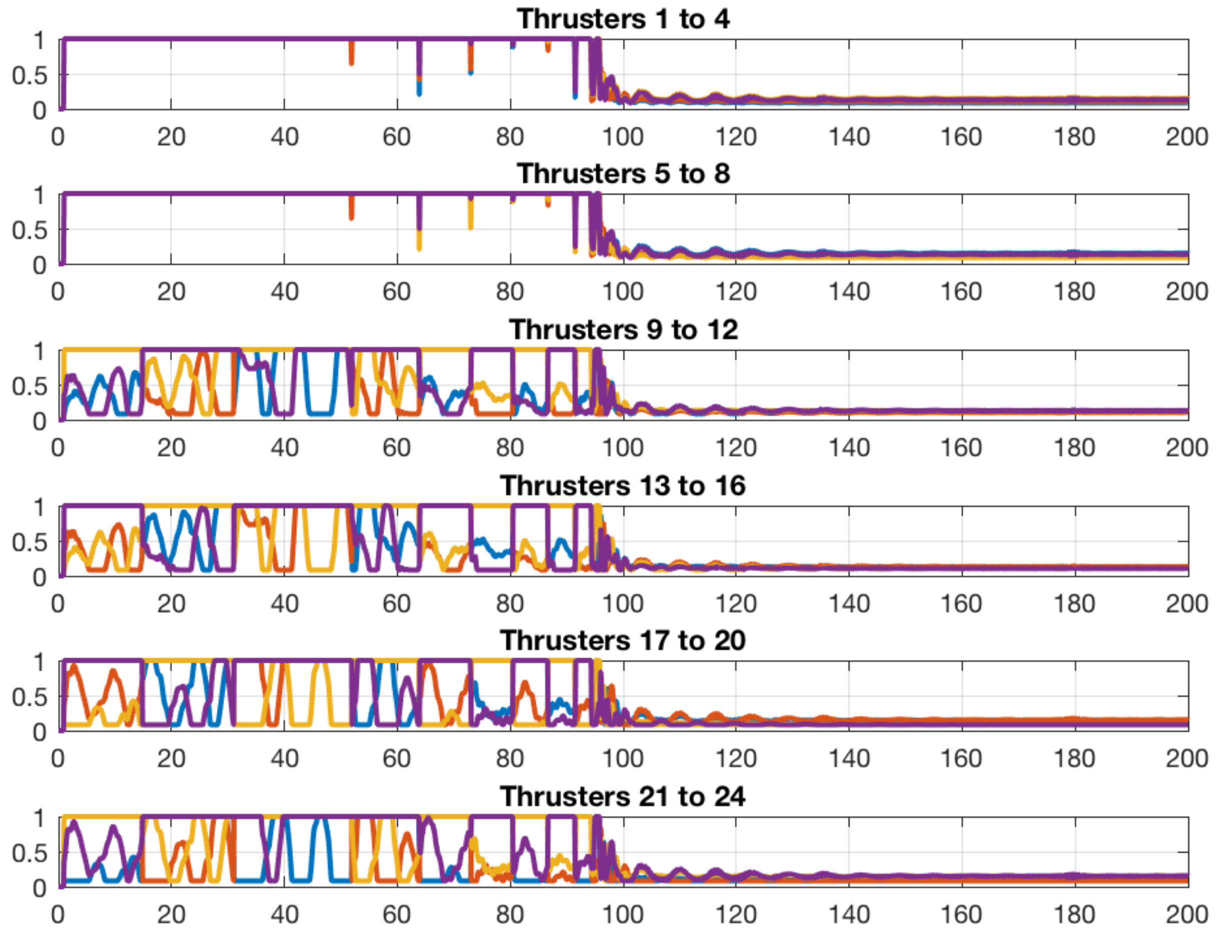


Figure 5.3: *Thruster duty cycle vs. time for MMS-like detumble example.*

null-space boosting end up being either at the minimum on time, or at 100 % duty cycle. Other mapping algorithms will produce different firing profiles, and therefore different disturbance profiles. (An observation which is mentioned again in the further research section of Chapter 6.)

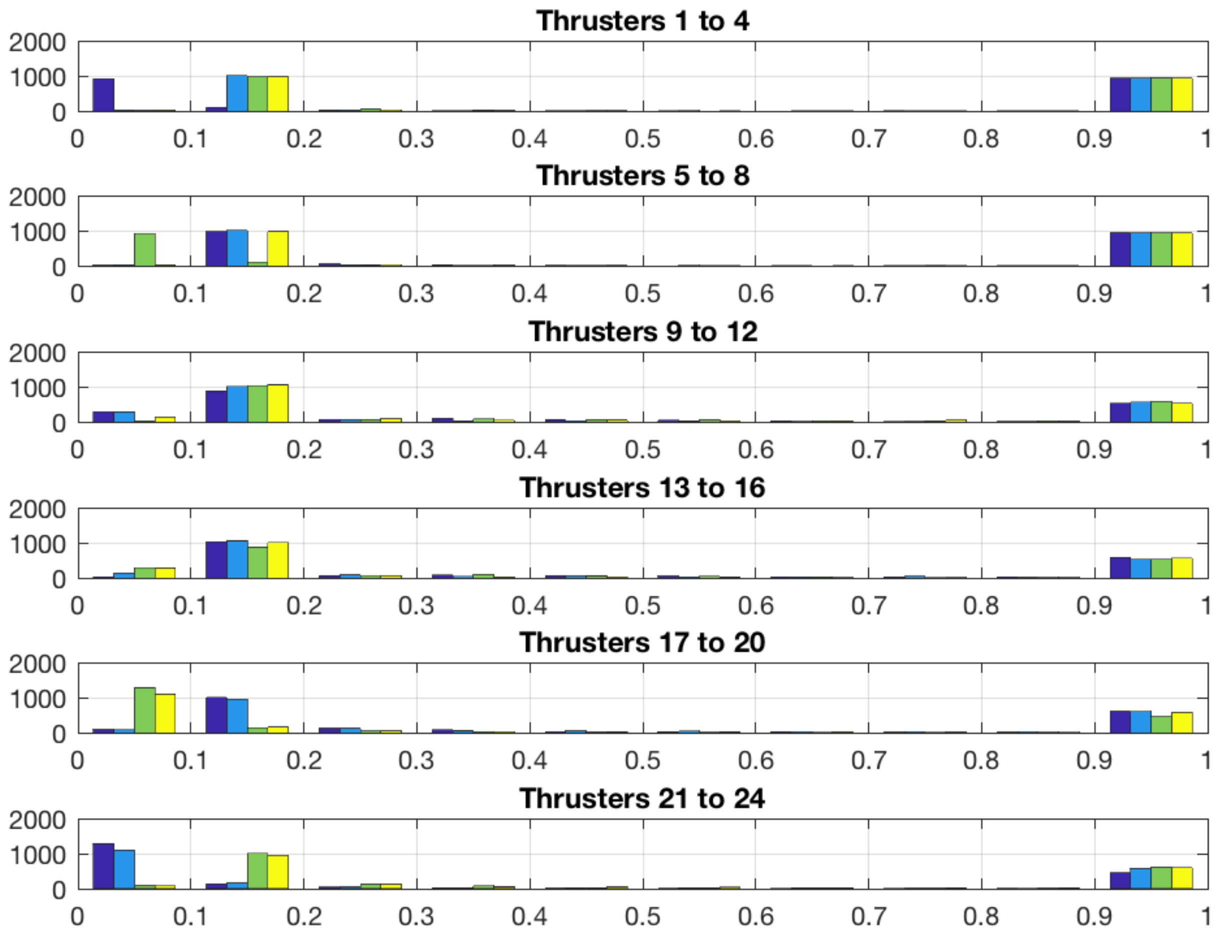


Figure 5.4: *Histogram of thruster duty cycles for MMS-like detumble example.*

5.3 Reaction Frequency Identification

Reaction frequency identification requires identification of frequencies induced by Servicer actuation from the ACS and the MANIP so that those frequencies may be removed from the signals detected in the FTS module. Each FFT result is reported as a relative signal magnitude in frequency bins whose width (and center frequency) is a function of sampling

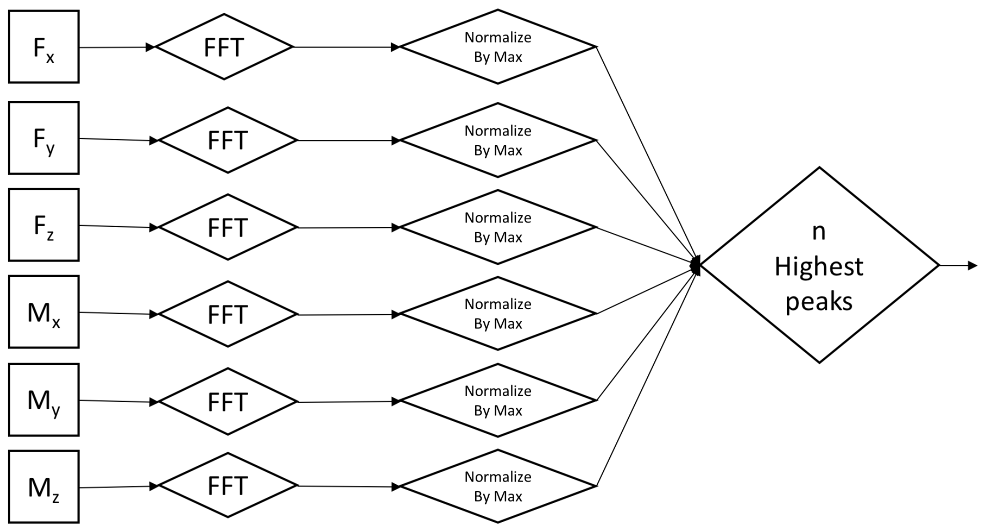


Figure 5.5: *Per-module FFT peak down-selection.*

frequency and number of time-series samples per FFT window, as discussed in Chapter 3. When combinbing these disparate sources, of potentially differing bin sizes, the coarseness of the destination bins becomes a design parameter to be chosen. Each bin (source or destination) has three frequencies of merit: the bin center frequency, f_c; the bin left frequency, $f_l = f_c - f_{width}/2$; and, the bin right center frequency, $f_r = f_c + f_{width}/2$. If the source FFT output has a bin width which differs from the destination bin width, four different situations can arise, illustrated in Figure 5.6:

A) The source peak's center frequency falls between a destination bin left and right frequencies but the source left and right frequencies fall outside of the destination bin, i.e. the source bin is much wider than the destination bin. The source peak's impact is spread across the bins both to the left (lower in frequency) and to the right (higher in frequency), as well as the current bin.

B) The peak's left source bin frequency falls outside of the destination bin, while the peak's right source bin frequency remains within. The source power is spread proportionally between the bin to the left of the destination (lower frequency) and the current destination bin.

C) Both the peak's left and the right source bin frequencies fall within the destination bin. This occurs when the width of the source bin is less than the width of the destination bins and the center frequency of the source peak is closer to the destination bin center than either edge.

D) The peak's left source frequency falls within the destination bin, while the peak's right source bin frequency falls outside. The peak's source center frequency lands within the destination bin range, but it is spread across the current destination bin and the one to the right (higher frequency).

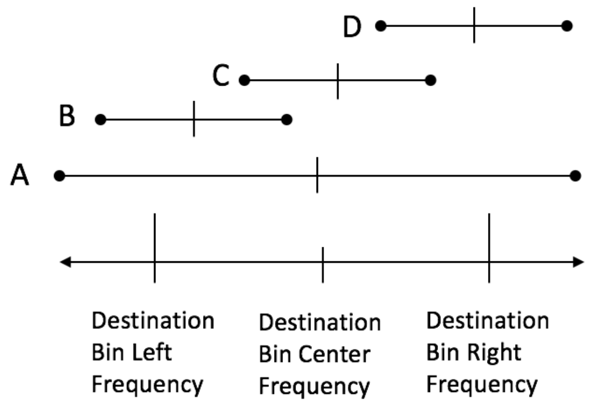

D
C
B
A
Destination Bin Left Frequency
Destination Bin Center Frequency
Destination Bin Right Frequency

Figure 5.6: *Four cases of handling FFT persistence allotment when source and destination bin widths differ: A) source width covers three destination bins; B) source partially covers current bin and one to the left; C) source completely within current bin, and; D) source partially covers current bin and on to the right.*

5.4 Client Appendage Motion Reduction

The goal of this research is a reduction of the vibrations within client appendages during detumble maneuvers. In the following example, a client is constructed with a single solar array appendage. Figure 5.7 labels the client body axes and the axes of the solar array appendage as displayed in its zero-angle position. The origin of the client body frame is at the center of the marman ring that attached it to its launch vehicle at the plane of contact. The Z axis is positive up through the body of the client spacecraft and colored red. The Y axis is positive in the direction away from the solar array. The X axis completes the right-hand rule system. The solar array consists of a single active drive assembly operating in the -Y axis of the client body frame and four solar array panels which are folded upon launch then deployed in orbit. Once the deploy activity is complete, these four joints lock into position and become passive flexible joints. When the active joint is at its zero-angle position (as shown in Fig. 5.7), each passive joint rotates in the client's Z axis.

The example below has the coupled client and servicer system with an initial attitude rate of $(1, 2, 3)$ degrees per second as expressed in the servicer body frame. The deflection of each joint in response to the detumbling maneuvers is shown in Figure 5.8. Each passive deployment joint was given the same natural frequency and damping dynamics. It can be seen in Figure 5.8 that this leads to similar angle waveforms but the amplitudes differ based on total inertias experienced outboard of each joint.

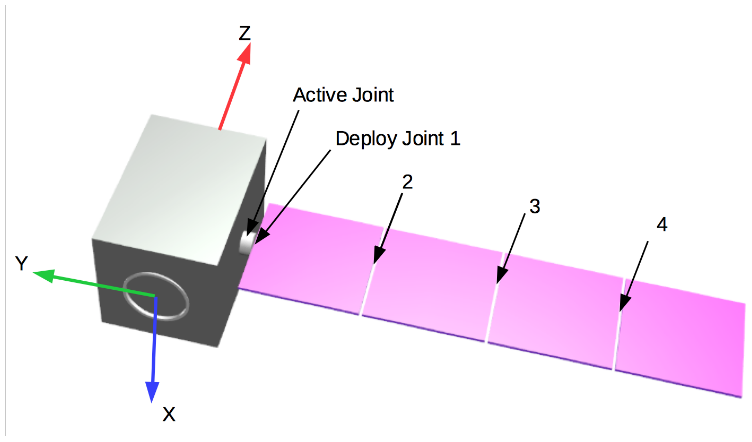


Figure 5.7: *Client satellite with body axes, with the actively controlled joint and passive deployment joints (1 through 4) labeled. Array shown at its zero angle position.*

The simulation's truth of the angle of each passive deployment joint (directly sampled from the dynamics) is processed via a spectrogram below in Figures 5.9 through 5.12. The angles are sampled at 100 Hz from the simulation dynamics; no sampling noise was inserted to recreate any form of joint angle measurement. The samples are windowed with a rectangular window. Each window of is 4096 samples. The stride between each window is 1 sample. With a sampling rate of 100 Hz, the center frequency of the highest bin of the spectrogram is about 50 Hz. Reviewing Figures 5.9 through 5.12, it is clear that the interesting frequencies for these appendages are much smaller than 50 Hz, even less than 1 Hz. Given the very similar time-series response of the four joints in Fig. 5.8, it is not surprising that the spectrograms of each indicates very similar responses in the Frequency domain over time.

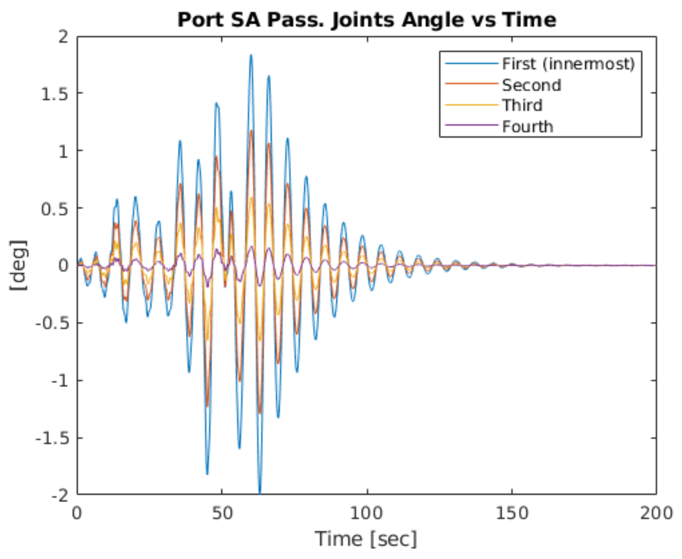


Figure 5.8: *Client solar array passive joint angles between individual panels during detumble. The first joint is the innermost, closest to the spacecraft body.*

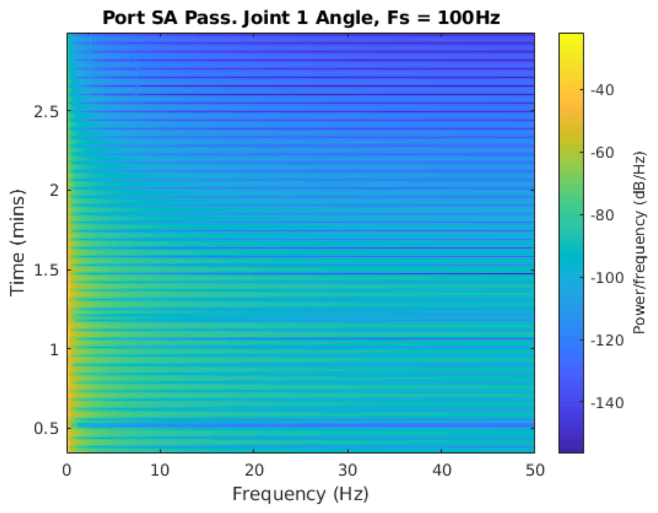


Figure 5.9: *First (innermost) client solar array passive joint spectrogram over all frequencies given 100 Hz sampling rate.*

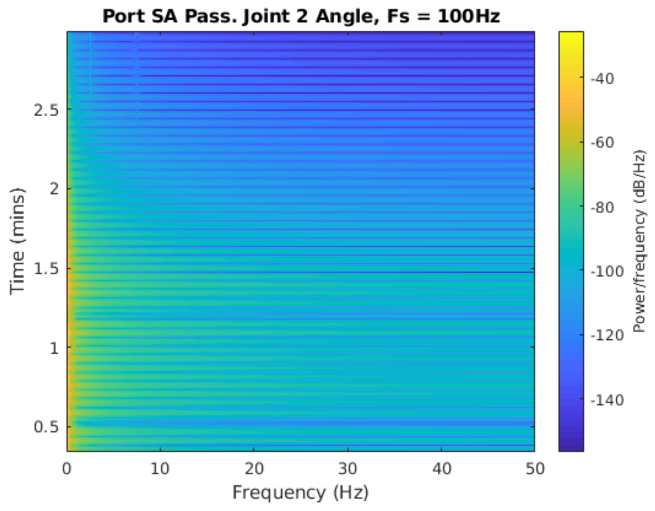


Figure 5.10: *Second client solar array passive joint spectrogram over all frequencies given a 100 Hz sampling rate.*

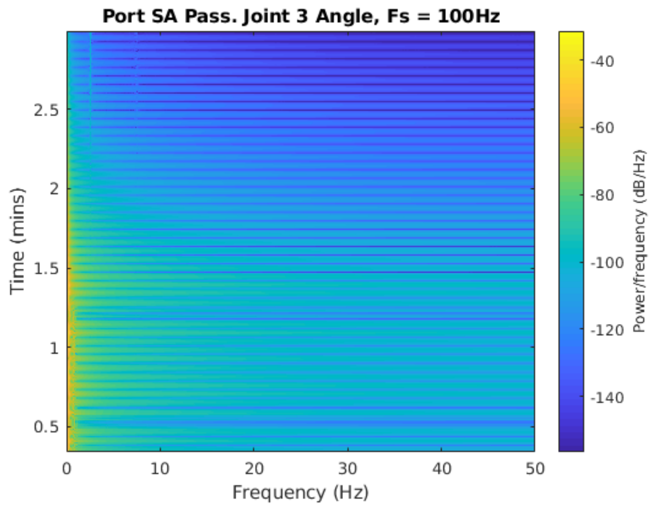


Figure 5.11: *Third client solar array passive joint spectrogram over all frequencies given a 100 Hz sampling rate.*

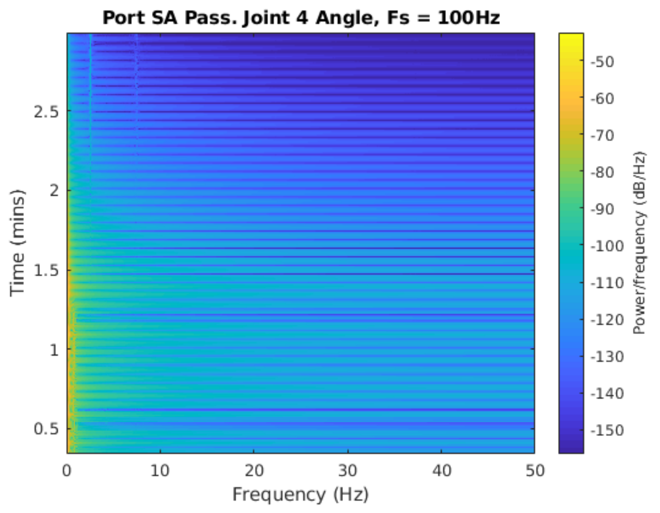


Figure 5.12: *Fourth (outermost) client solar array passive joint spectrogram over all frequencies given a 100 Hz sampling rate.*

Figures 5.13 through 5.16 zoom in on the frequency response in the spectrograms that are 5 Hz and under. In each of the four passive deployment joints, the dominant frequency response is at the frequency bin centered on 0.17 Hz. To appropriately respond to this frequency of dynamics, the FTS will need to process a sufficient number of measurements at a sufficient sampling frequency in each FFT window to detect this frequency and its phase.

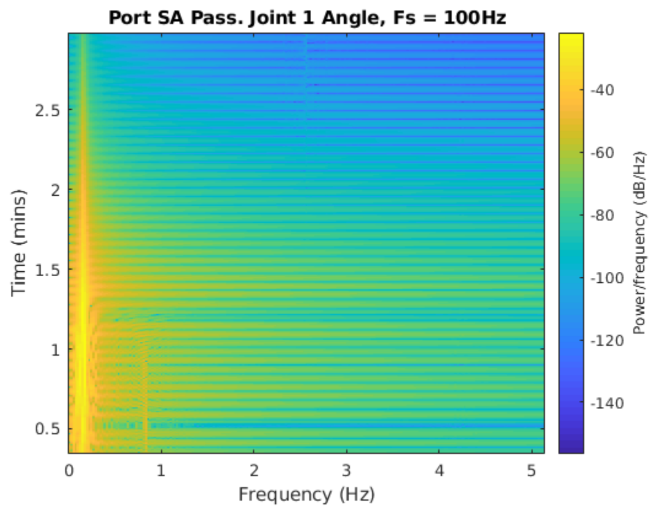


Figure 5.13: *First (innermost) client solar array passive joint spectrogram, zoomed in to clarify response below 5 Hz.*

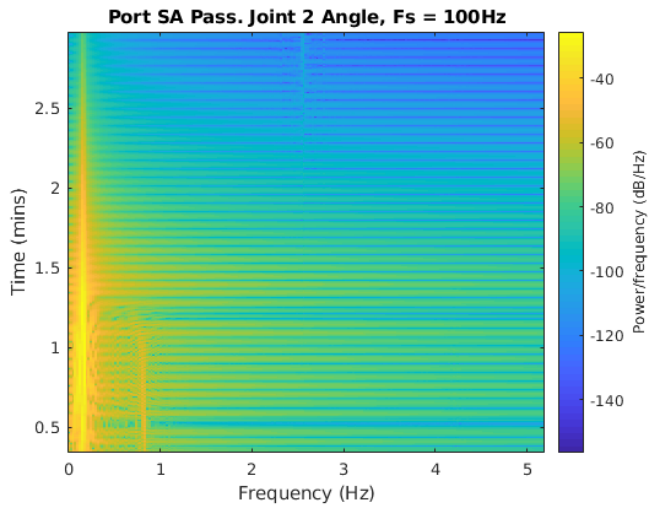


Figure 5.14: *Second client solar array passive joint spectrogram, zoomed in to clarify response below 5 Hz.*

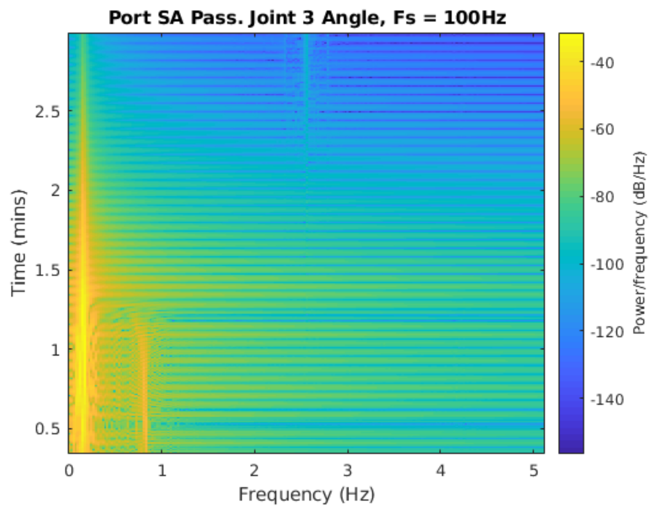


Figure 5.15: *Third client solar array passive joint spectrogram, zoomed in to clarify response below 5 Hz.*

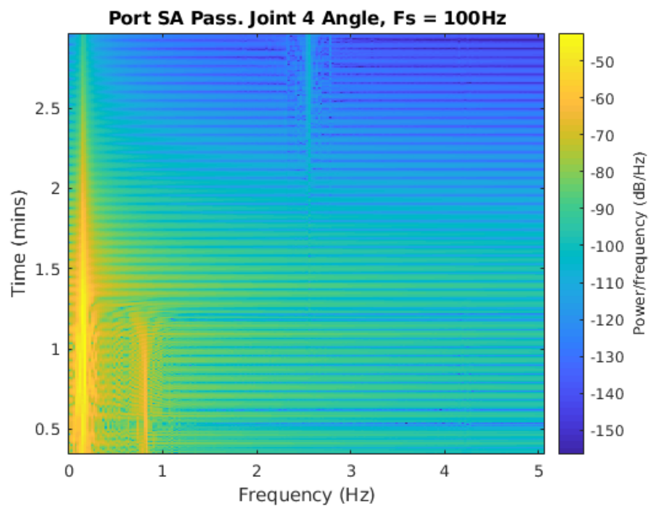


Figure 5.16: *Fourth (outermost) client solar array passive joint spectrogram, zoomed in to clarify response below 5 Hz.*

Next, the FTS measurements will be inspected to see if that is possible. Figure 5.17

shows the time-series measurements of the FTS, in the FTS's own coordinate frame, during the grappled detumble as described above. This shows how the FTS senses the ACS reactions up until the attitude errors are damped out around 60 seconds into the simulation, but the reactive forces of the client appendages continue to be felt as motion damps out.

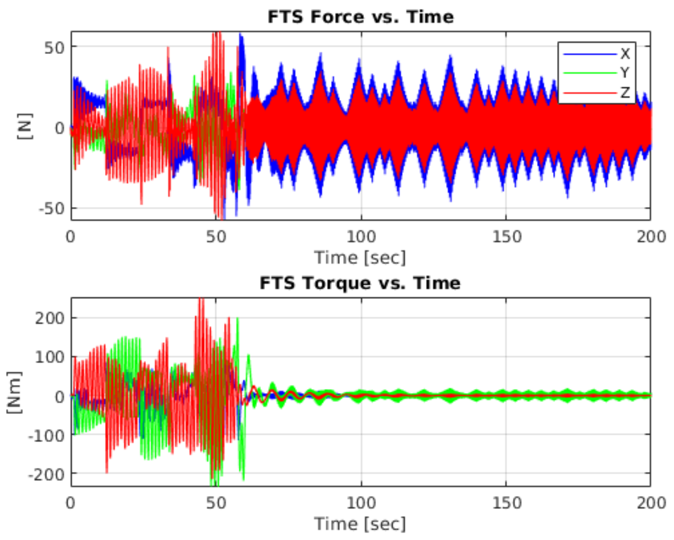


Figure 5.17: *Forces and Torques sensed at the FTS during detumble with the manipulator in joint hold mode, prior to reactive forces/torques generation.*

Spectrograms for the FTS-sensed forces and torques show similar spectral response given the full sampling rates of the sensor. For example, Figure 5.18 shows the response in the X force dimension out to 125 Hz. The interesting content is down within 10 Hz.

Figure 5.19 shows the sensed X force spectrogram for 5 Hz and under.

In Figure 5.19, the same peaks at roughly 0.75 Hz can be seen until the ACS system stops reacting, and the natural frequency reaction of the client solar array can be seen to

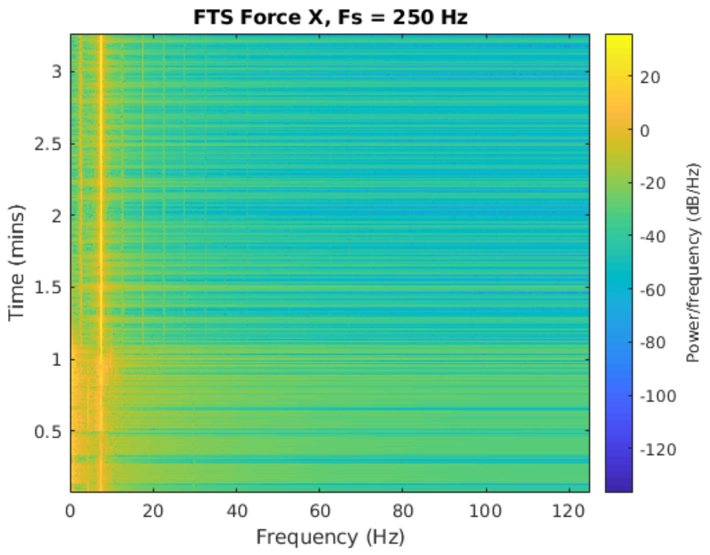


Figure 5.18: *Spectrogram of FTS Force X during detumble up to 125Hz.*

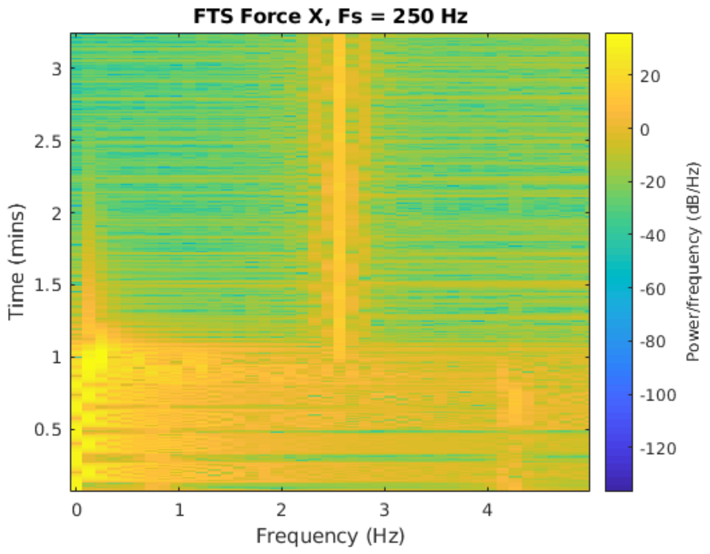


Figure 5.19: *Spectrogram of FTS Force x during detumble for 5Hz and under.*

continue to respond as it dampens out between 0.15 and 0.2 Hz. This mode can be similarly sensed in the Y and Z Forces (Figures 5.20 and 5.21), as well as the X through Z torques at the FTS.

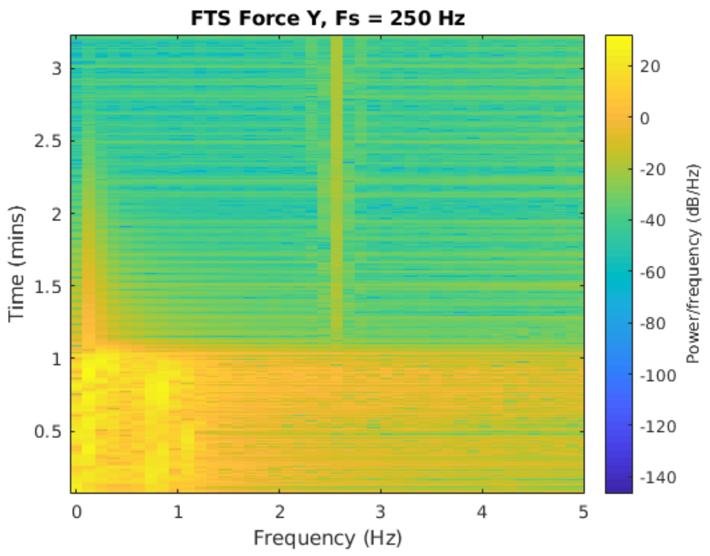


Figure 5.20: *Spectrogram of FTS Force Y during detumble for 5 Hz and under.*

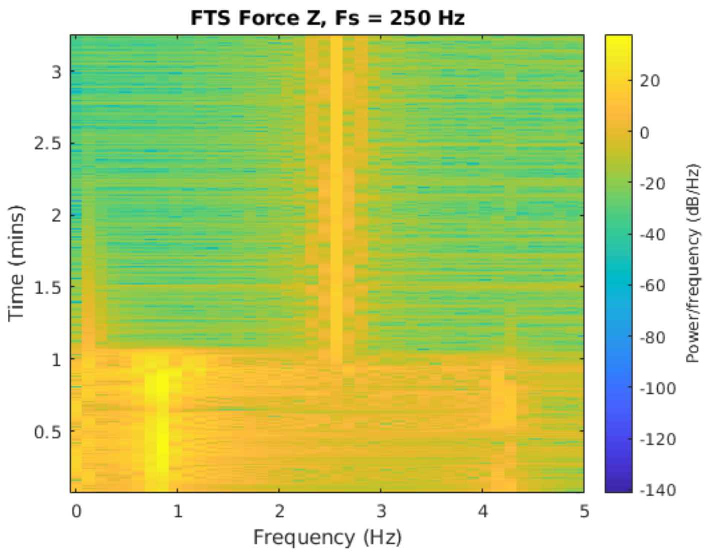


Figure 5.21: *Spectrogram of FTS Force Z during detumble for 5 Hz and under.*

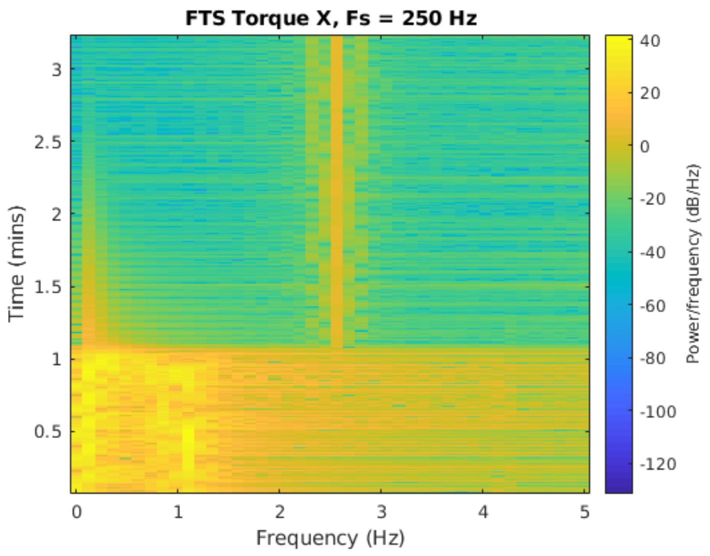


Figure 5.22: *Spectrogram of FTS Torque X during detumble for 5 Hz and under.*

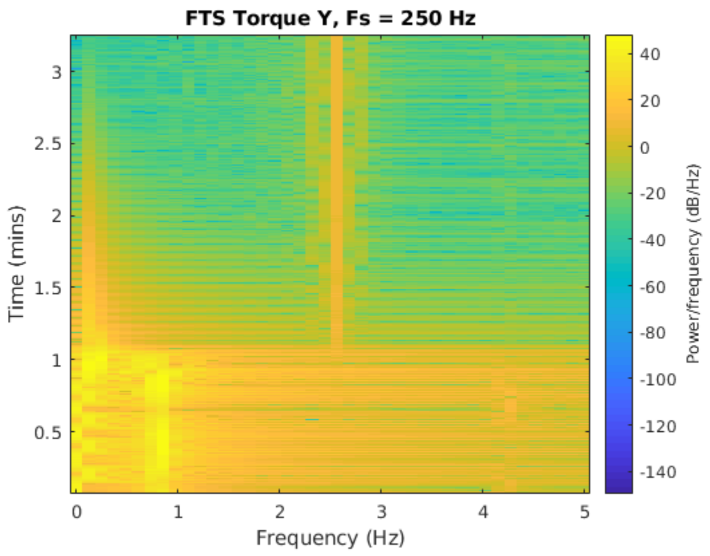


Figure 5.23: *Spectrogram of FTS Torque Y during detumble for 5 Hz and under.*

This establishes how the motion of the client appendages is detectable in the grapple manipulator's FTS.

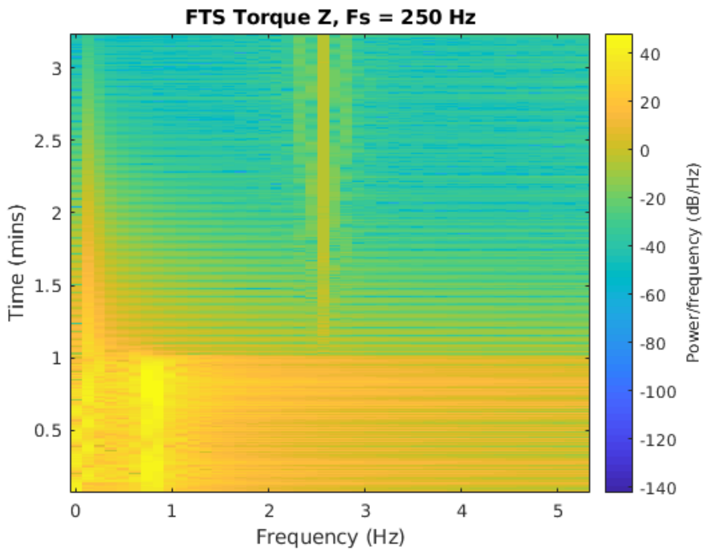


Figure 5.24: *Spectrogram of FTS Torque Z during detumble for 5 Hz and under.*

5.5 Scenario 1: Single-Axis Detumble

Spin-stabilized spacecraft have the majority of their momentum in a single axis by design, for example MMS. The next scenario is a coupled system of a servicer and a client satellite that start with a single-axis tumble rate similar to MMS: $(0, 0, 18)$ deg/sec. [60]. The example client spacecraft is notably different from a member of the MMS constellation in that it is an assymetric spacecraft with a deployable solar array, as shown in Figure 5.25, whereas the MMS spacecraft have their solar cells on the main body of spacecraft and featuring multiple symmetric radial and axial boom antennae.[1] However, the spin rate of

[1]It is much harder to detect the appendage motion of client spacecraft which are symmetric via only the FTS at the grapple point due to the equal and opposite reactions of the appendage joints through the

the example spacecraft is representative of an MMS member spacecraft. Further, the modal responses of the client appendages that dominate the detectable response are due to the joints between the solar array panels, as opposed to potential higher frequency modes in the torsional or off-axis bending directions.

As in prior scenarios, the servicer control system has no knowledge or measurement of the client's appendage joint angles. The only source of information it has about the client motion is measured through the FTS and detected via the sliding window FFT peak detection methods described above. Figure 5.26 shows how the client's solar array appendage joints react to the detubmling with the Reactive Control algorithm disabled, as recorded by the simulation's dynamics module. In this figure, the robotic manipulator that couples the two spacecraft is under active control but remains only in the Joint Hold mode and does not make use of the data from the FTS. Here it is only the natural damping of the solar array joints that reduces joint motion over time. The goal of the Reactive Control algorithm is to decrease the time required to damp joint deflections and limit the maximum deflections experienced while Reactive Control is active.

Figure 5.27 shows how the client appendage joints respond when the Reactive Control is engaged, activating at 40 seconds into the simulation. Prior to the activation of the Reactive Control system, the manipulator is effectively in the Joint Hold mode. In this scenario, the active degree of freedom for the Reactive Control algorithm is Z translation.

motion. The assymetry prevents the natural cancellation and therefore allows the detection.

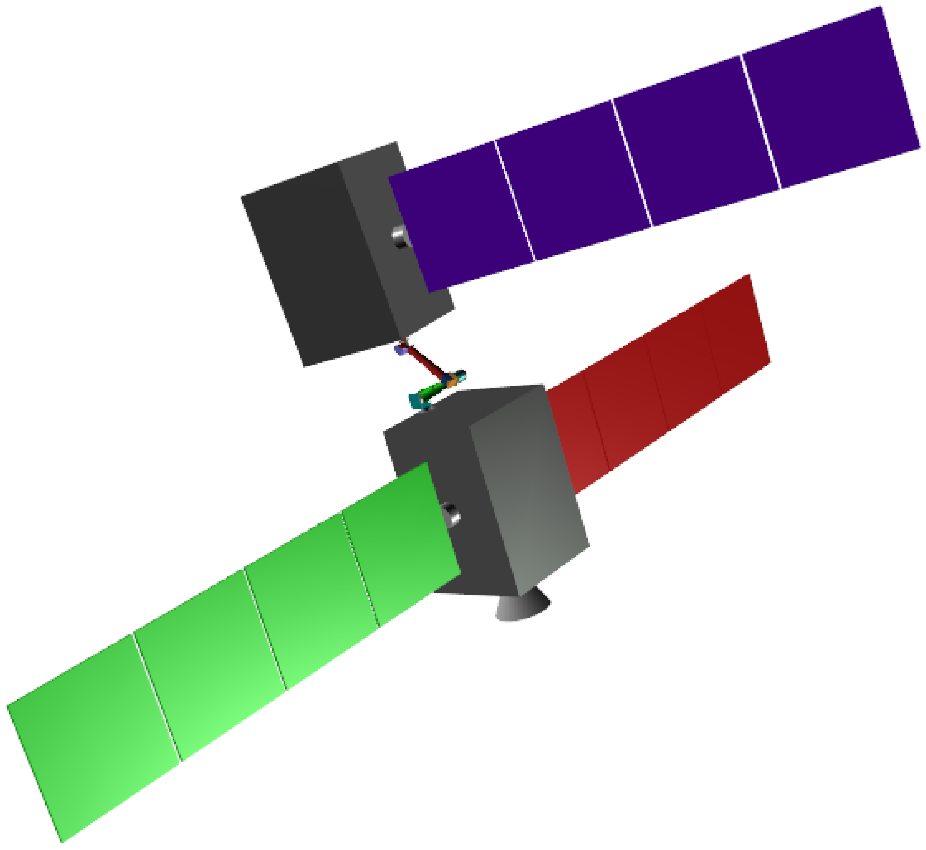


Figure 5.25: *Servicer (with red and green solar arrays) and client (with purple solar array) configuration at start of detumble operations. The sevicer arrays are modeled as rigid here. The purple client solar array is modeled with flexible joints between each of the four array panels.*

Note the decreased peak displacement once Reactive Control is enabled and faster motion dissipation, compared to Fig. 5.26. Figure 5.28 shows the change in the quaternion error over time during this detumble with the Reactive Control enabled, proving that the system attitude can be stabilized while also reducing client appendage motion.

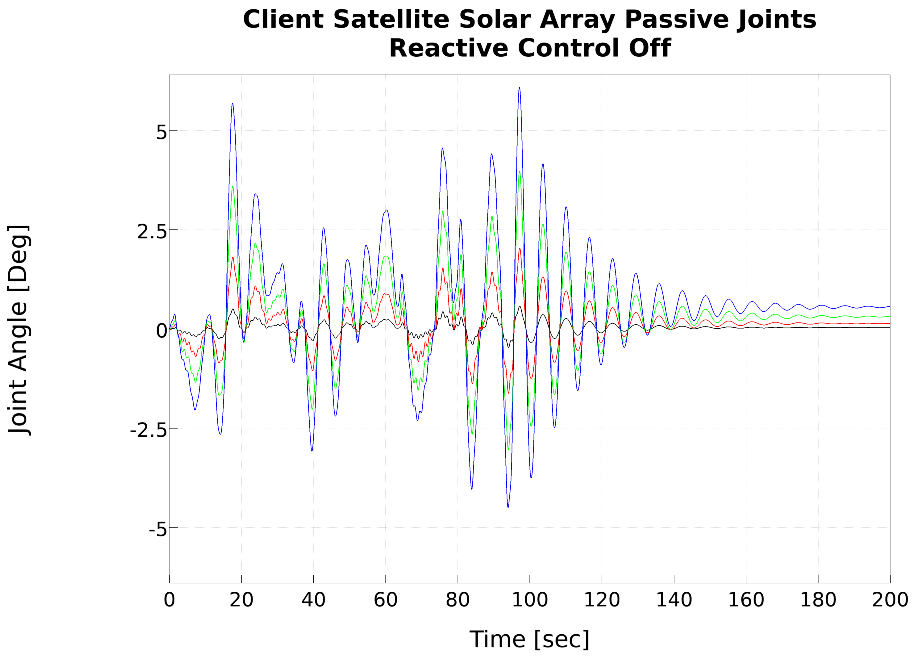


Figure 5.26: *Client appendage joint response during detumble without Reactive Control activation. The blue trace is the innermost passive array joint; the black trace is the outermost passive array joint.*

Figures 5.30 through 5.32 compare the motion of the client's appendage joints during detumble when the Reactive Control is on and off. The Y axis scale of each figure is different to highlight the effect of the Reactive Control at reducing the passive joint motion. If the Y axes were held the same for all four figures, the effect would be difficult to observe in the outer joints.

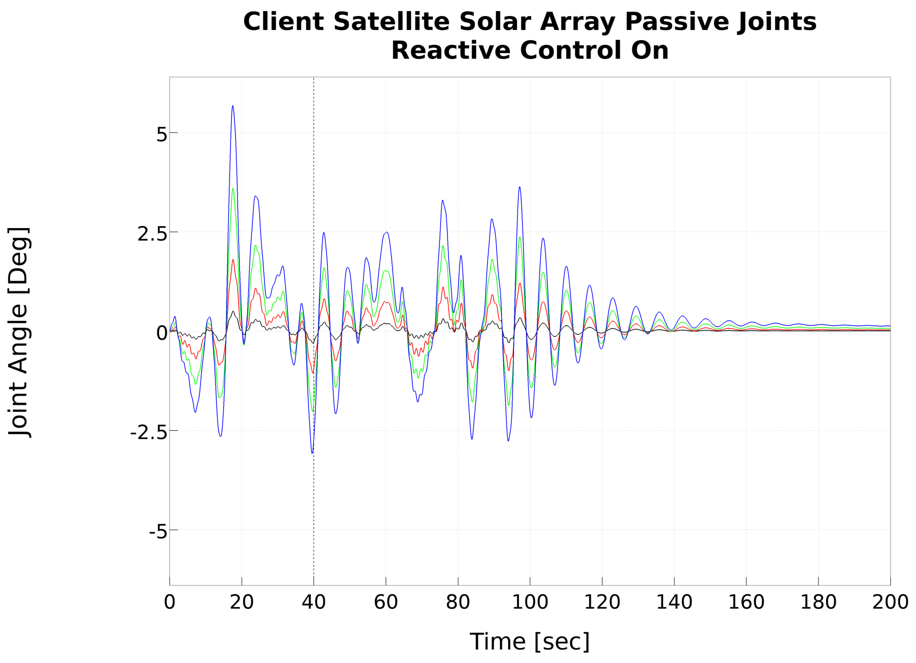


Figure 5.27: *Client appendage joint response during detumble with Reactive Control iniated at 40 seconds into the simulation. The blue trace is the innermost passive array joint; the black trace is the outermost passive array joint.*

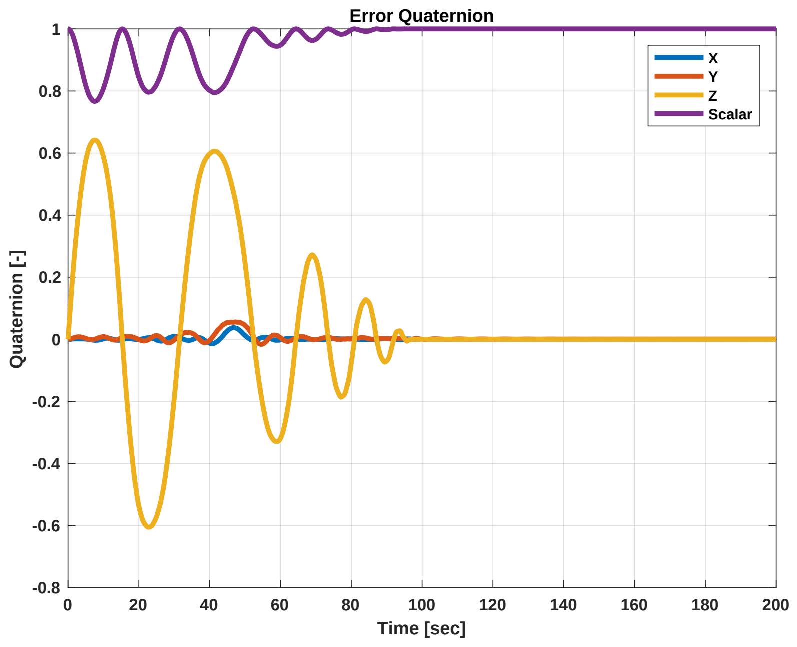


Figure 5.28: *MMS-like client detumble error quaternion with reactive control enabled at 40 seconds into the simulation.*

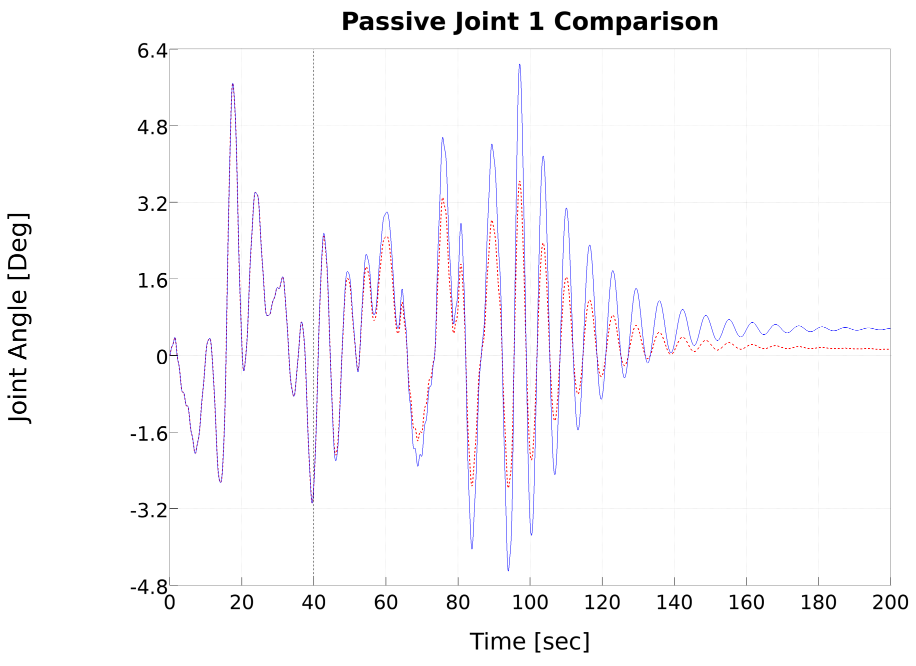


Figure 5.29: *Client's innermost solar array joint during an MMS-like detumble comparison between Reactive Control disabled and enabled. The blue trace is without Reactive Control; the red dashed trace is with Reactive Control enabled at 40 seconds.*

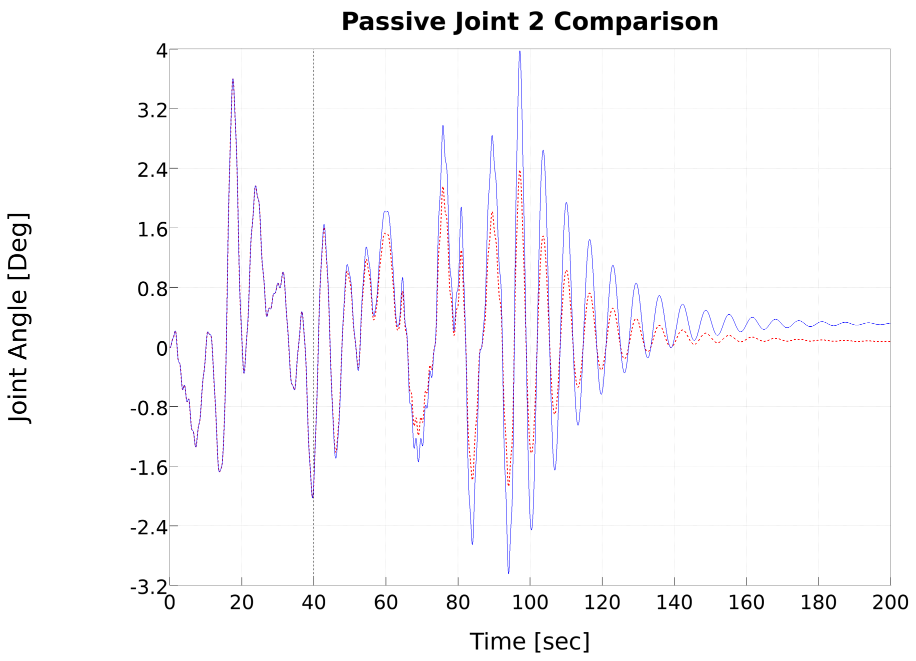


Figure 5.30: *Client's second solar array joint during an MMS-like detumble comparison between Reactive Control disabled and enabled. The blue trace is without Reactive Control; the red dashed trace is with Reactive Control enabled at 40 seconds.*

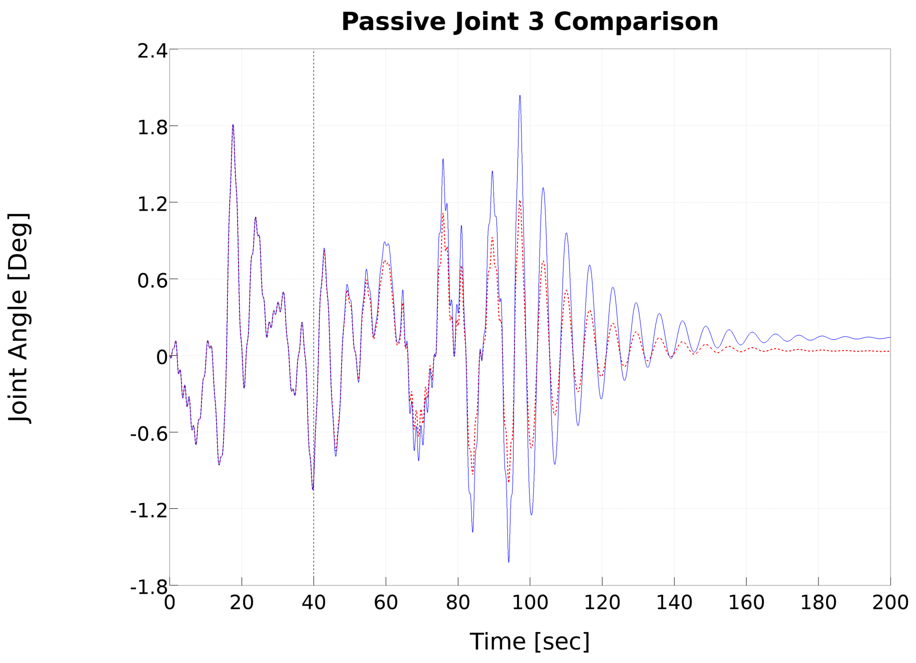


Figure 5.31: *Client's third solar array joint during an MMS-like detumble comparison between Reactive Control disabled and enabled. The blue trace is without Reactive Control; the red dashed trace is with Reactive Control enabled at 40 seconds.*

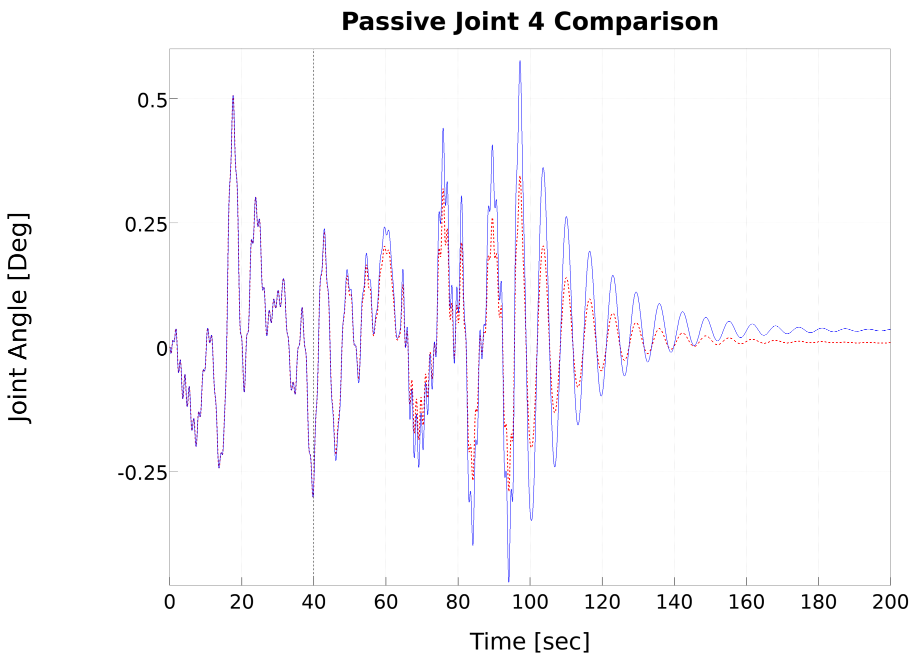


Figure 5.32: *Client's outermost solar array joint during an MMS-like detumble comparison between Reactive Control disabled and enabled. The blue trace is without Reactive Control; the red dashed trace is with Reactive Control enabled at 40 seconds.*

5.6 Scenario 2: Multi-Axis Detumble

The next scenario demonstrates the ability of these methods to bring a client/servicer system, similar to the previous example, under control with an initial multi-axis tumble of $(2, 20, 5)$ deg/sec (as measured in the servicer body frame). As in prevous scenarios, the ACS is enabled at 1 second into the simulation. Figure 5.33 shows the ACS response for the system with the manipulator's reactive control disabled; instead the manipulator is in joint hold mode for the entire detumble to show client appendage response without the reactive control effects. As in prior examples, the innermost solar array joint is represented by a blue trace and shows the greatest displacement. Here, also, the active DOF for the reactive control is the Z translational axis.

Once Reactive Control was enabled, figure 5.34 shows the body rate errors of the servicer during the detumble maneuvers. Also, figure 5.35 shows the attitude error as a quaternion during the detumble maneuvers. Not surprisingly, the attitude and rate errors in the servicer's Y axis (green trace) dominate both figures until there is sufficient time to damp the motion. Figure 5.36 shows the response of the client appendage joints during the same multi-axis detumble with Reactive Control enabled at 40 seconds into the simulation. The overall deflection of the client appendage joints is reduced by the active manipulator reactions.

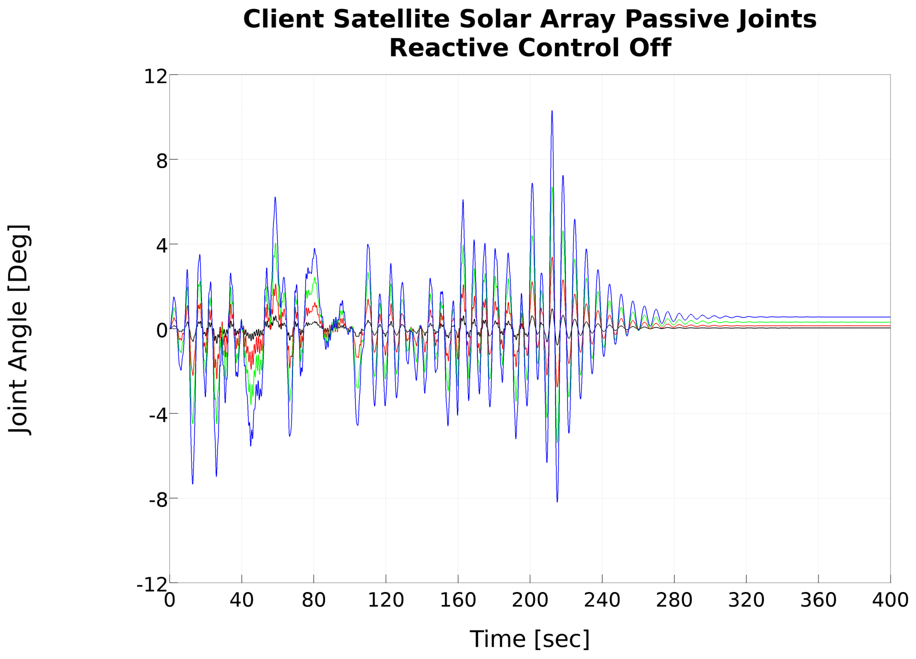


Figure 5.33: *Multi-axis detumble with the manipulator in joint hold mode (reactive control disabled).*

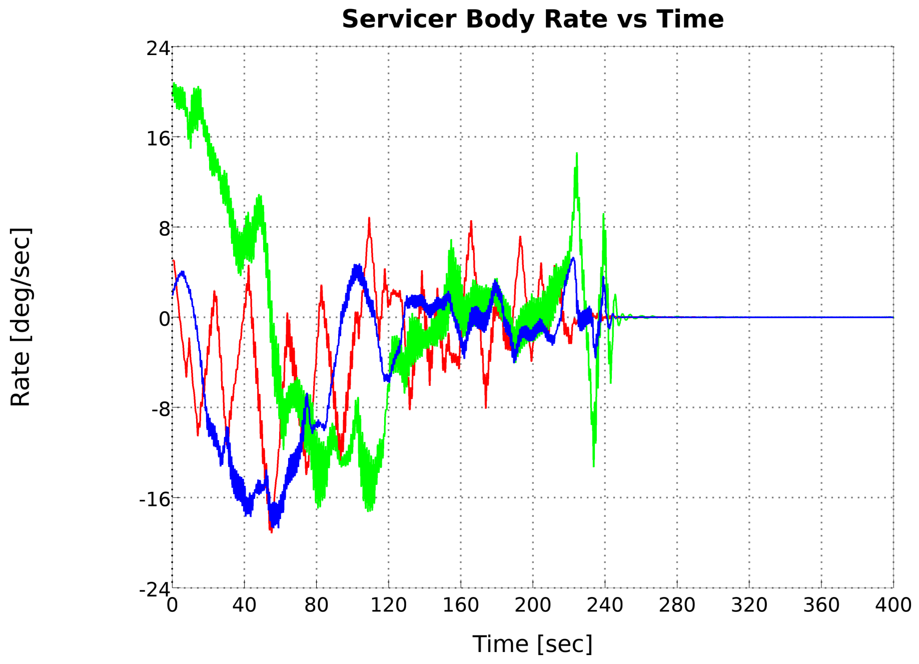


Figure 5.34: *System body rate vs. time during multi-axis detumble. The blue trace is the servicer X axis rates. THe green trace is the servicer Y axis body rates. The red trace is the servicer Z axis body rates.*

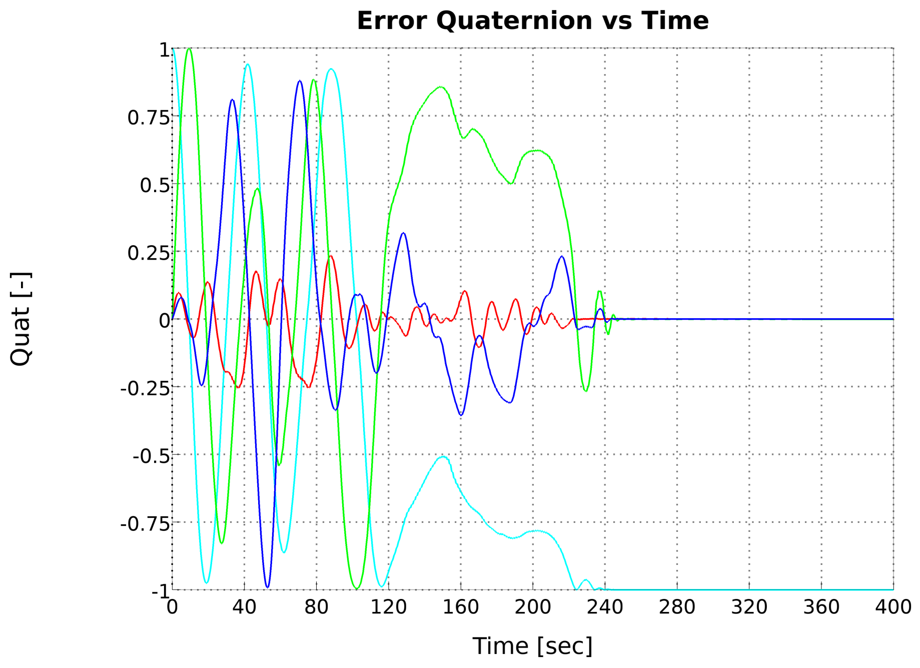


Figure 5.35: *System attitude error quaternion vs time during multi-axis detumble.*

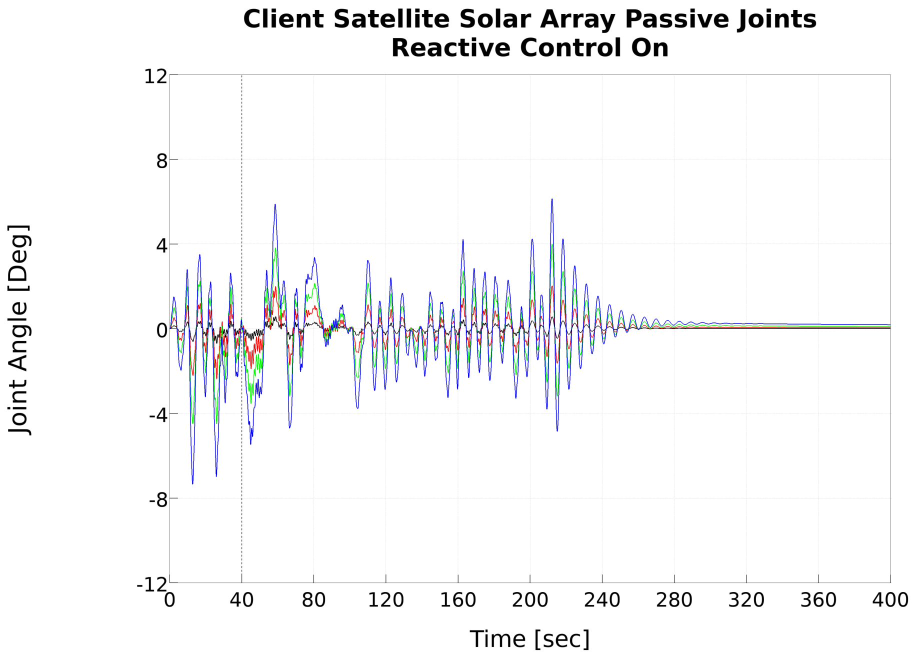


Figure 5.36: *Multi-axis detumble, client appendage joint deflections with Reactive Control engaged at 40 seconds.*

Figure 5.37 through 5.40 compare the client appendage joint deflections between reactive control being disabled and enabled. Here it can be seen how the reactive control is able to reduce the client appendage reduction.

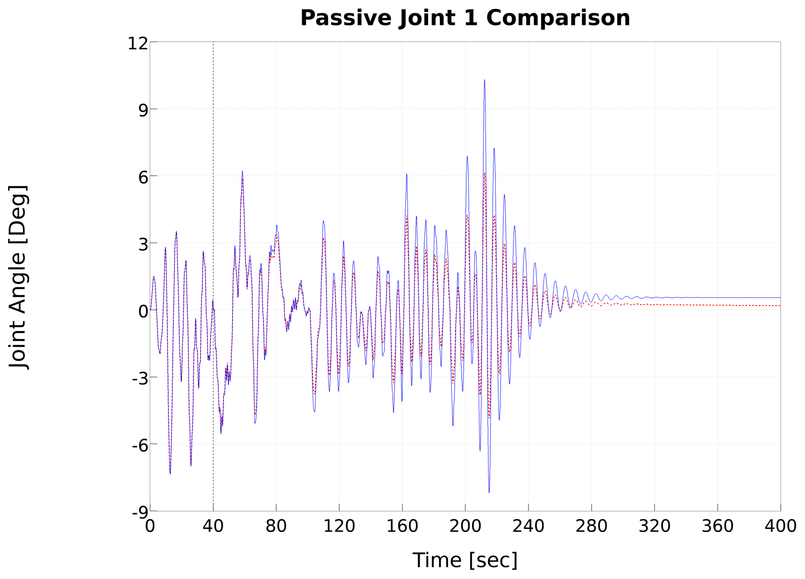


Figure 5.37: *Multi-axis detumble, client appendage innermost joint deflections. Blue trace is with the reactive control disabled; red-dash trace is with the reactive control enabled.*

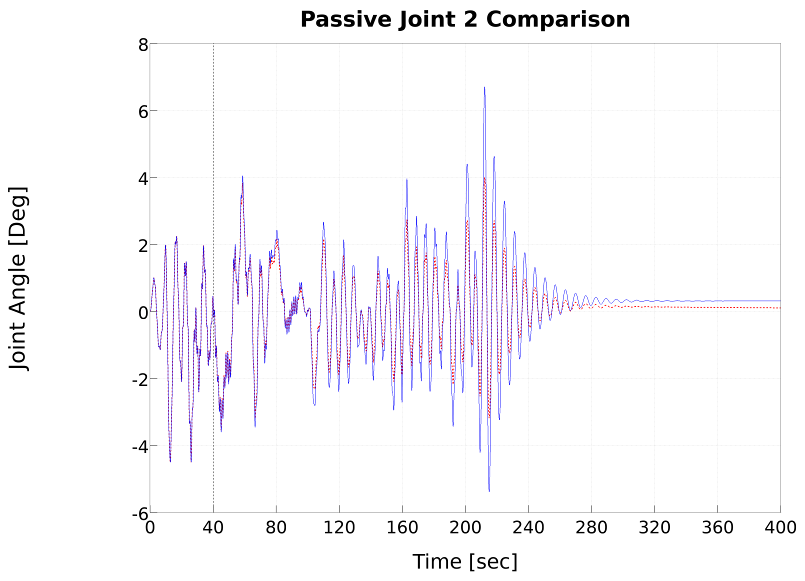


Figure 5.38: *Multi-axis detumble, client appendage second joint deflections. Blue trace is with the reactive control disabled; red-dash trace is with the reactive control enabled.*

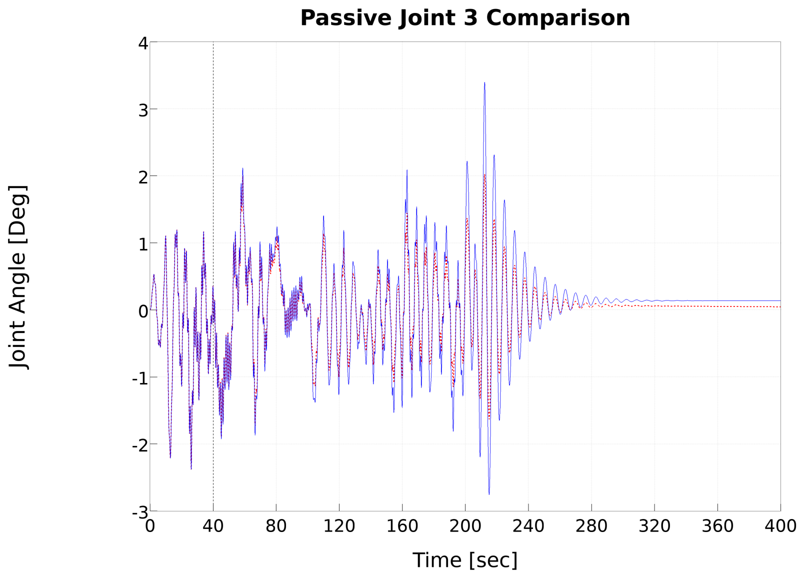


Figure 5.39: *Multi-axis detumble, client appendage third joint deflections. Blue trace is with the reactive control disabled; red-dash trace is with the reactive control enabled.*

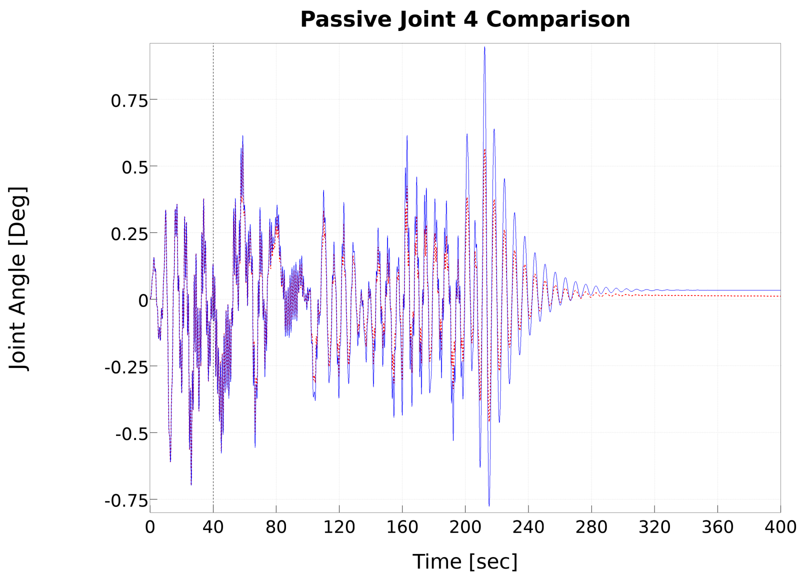


Figure 5.40: *Multi-axis detumble, client appendage fourth (outtermost) joint deflections. Blue trace is with the reactive control disabled; red-dash trace is with the reactive control enabled.*

5.7 Conclusion

This chapter has presented the simulation environment which pulled together the implementation of all three contributions of this work in a 6-DOF Earth-orbit environment: the nonlinear attitude controller, the client appendage modal detection with a combination of FTS and FFT signal processing; and a Cartesian-space reactive tool contoller to reduce client appendage motion. These efforts combined show the effectiveness at reducing the client appendage motion during a detumble maneuver.

Chapter 6: Conclusion

6.1 Contributions

The contributions of this work have been three-fold. First, a novel quaternion feedback controller with a dense matrix gain was presented that outperforms its scalar counterparts in several desirable ways. Most importantly, the stability of this attitude controller was proven in the presence of inertia knowledge errors. Second, a method of automated modal disturbance detection with an force-torque sensor via the FFT in the presense of sensor noise was presented which enables the third contribution. Finally, a manipulator control algorithm that takes advantage of the second contribution was presented to reduce the disturbances contributed by a client's appendages by modifying the end effector Cartesian path reaction based on the modal response of the client, in a system where a client spacecraft is grappled by a servicer spacecraft with a robotic manipulator.

6.2 Proposed Further Research

One limitation that was observed of this method was the lack of ability to detect client appendage motion in the cases of symmetric client appendages during the detumble. This has been attributed to a natural cancellation of disturbance torques caused by appendage deflection in the symmetric case that masks the actual appendage joint motion from the FTS. One future avenue of research for improving this method would be the inclusion of a computer vision algorithm to identify appendage joint motion so that it may be fused with the FTS information to create a better estimate of the client's motion. A computer vision implementation would present a non-intrusive, non-contact method of estimating the joint appendage motion that would be well suited to the non-cooperative client model, at the risk of adding lighting requirements to the concept of operations.

The current method relies heavily on the ability of the FFT to detect the phase of the modes of interest. Robustness would be improved by inlcusion of additional methods of the initial detection of modal phases or the subsequent verfication that the dominant mode phase hasn't changed.

Another avenue of improvement for this research would be additional research into other methods of modal identification from the FTS data such as adaptive notch filters that might be able to perform the frequency and phase identification without the cumbersome sliding-window FFT and its multiple associated configuration parameters.

Finally, the reactive control method proposed above is a Cartesian-space reaction. Another possible method for reactive control is to keep the trajectory in the velocity realm and feed forward the ACS reactions to perform the detumble to the manipulator's tool tip and generate reaction forces and torques for the manipulator based on the desired response at the grapple point.

Appendix A: Servicer Manipulator

The servicer in this research uses a 7-DOF manipulator. The manipulator's Denavit-Hartenberg (DH) parameters, with all of the active joints at their zero positions, are listed in Table A.1. These parameters follow the convention defined in Craig [68]. The axes and d_i parameters are depicted on its CAD model in Fig (A.1). The palm of the gripper is 0.15 meters from the origin of joint 7.

Table A.1: *Modified DH Parameters of the servicer's manipulator, with all arm joints at zero.*

Link i	$\boldsymbol{\alpha_{i-1}}$ **(rad)**	$\mathbf{a_{i-1}}$ **(m)**	$\mathbf{d_i}$ **(m)**	$\boldsymbol{\theta_i}$ **(rad)**
1	0	0	0.255	0
2	$\pi/2$	0	0	0
3	$-\pi/2$	0	0.97	0
4	$-\pi/2$	0	0.215	0
5	$\pi/2$	0	1.02	0
6	$\pi/2$	0	0	0
7	$-\pi/2$	0	0	0

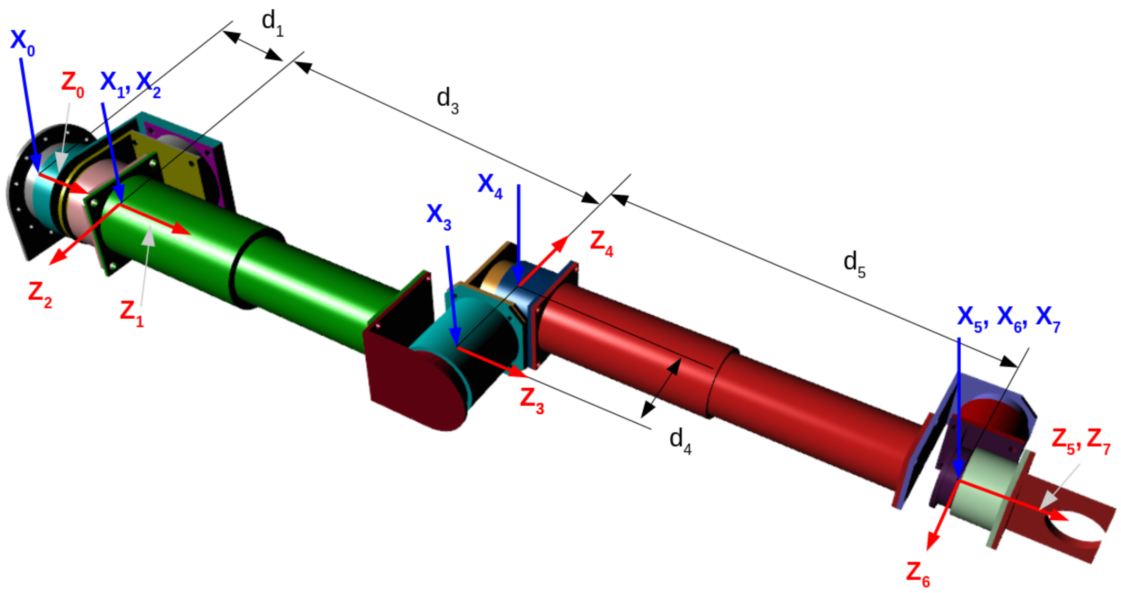


Figure A.1: *Servicer manipulator DH axes and parameters visualized. All joint angles are zero degrees. Y axes are not shown for clarity, but each completes the right hand rule for that frame.*

Table A.2: *Link mass and position of center of mass of each link, i, from the origin of link i, in meters, to the nearest millimeter, expressed in that link's mechanical frame for the servicer manipulator.*

Link	**Mass [kg]**	**Center of Mass**		
		$\mathbf{X_{Li}}$	$\mathbf{Y_{Li}}$	$\mathbf{Z_{Li}}$
1	10.35	0	0.157	0.103
2	6.93	0	-0.035	0.044
3	18.37	0	-0.013	0.420
4	6.93	0	0.008	-0.001
5	17.34	0	0.033	0.448
6	4.82	0	-0.001	0.039
7	2.73	0	-1.080	0.007

Table A.3: *Moment of Inertia for each link, i, in kgm^2, at the center of mass of the link, expressed in the link's mechanical coodrinate frame.*

Link	$\mathbf{I_{XX}}$	$\mathbf{I_{YY}}$	$\mathbf{I_{ZZ}}$	$\mathbf{I_{XY}}$	$\mathbf{I_{XZ}}$	$\mathbf{I_{YZ}}$
1	0.167	0.132	0.127	0	0	-0.048
2	0.091	0.087	0.066	0	0	0.022
3	2.160	2.143	0.158	0	0	0.129
4	0.087	0.090	0.042	0	0	-0.006
5	2.246	2.171	0.154	0	0	-0.265
6	0.022	0.023	0.012	0	0	0
7	0.013	0.014	0.009	0	0	0

Appendix B: Simulation Source Code

B.1 fft.h

B.1.1 struct fft_persistence_params_s

Listing B.1: *Peak persistence search parameter structure*

```
typedef struct fft_persistence_params_struct {
  /* number of bins in destination FFT */
   unsigned int dst_nBins;
   /* destination FFT delta-frequency per bin */
   double       dst_dF;
   /* number of bins in source FFT */
   unsigned int src_nBins;
   /* source FFT delta-frequency per bin */
   double       src_dF;
```

```
    /* time between source samples, sec */
    double        src_dt;
    /* multiplicative amplitude degradation factor [0:1]*/
    double        degradeFactor;
    /* bin size percentage epsilon for smearing check */
    double        binEpsilon;
    /* nonzero if the source peaks will be smeared across mult dest bins */
    unsigned int doSmear;
} fft_persist_params_s;
```

B.1.2 struct fft_persistence_data_s

Listing B.2: *Peak persistence search data structure*

```
typedef struct fft_persistence_data_struct {
    /* number of bins in persistence output */
    unsigned int n;
     /* persistent amplitude (n  length) */
    double *amp;
    /* persistent time (n length) */
    double *time;
    /* change in phase of this bin's persistent phase (n length)*/
    double *deltaPhase;
    /* phase in this bin from previous sample (n length)*/
    double *prevPhase;
    /* rank of peak (1 being largest peak) of bins (n length)*/
    double *rank;
} fft_persist_data_s;
```

B.1.3 struct fft_peaks_s

Listing B.3: *FFT peaks data structure*

```
typedef struct fft_peaks_struct {
    double  *nPeaks;
    double  *freqs;
    double  *amps;
    double  *phases;
} fft_peaks_s;
```

B.1.4 struct fft_output_s

Listing B.4: *FFT output data structure*

```
typedef struct fft_output_struct {
   /* inputs/outputs */
   double     *newMeas;    /* nonzero to indicate the data has changed */

   /* parameters that won't change */
   int         nBins;
   double      *Fs;
   double      *df;
} fft_out_s;
```

B.1.5 struct fft_s

Listing B.5: *FFT top-most data structure*

```
typedef struct fft_struct {
   /* parameters */
```

```
double Fs;            /* sampling frequency [Hz] */
unsigned winLen;      /* # samples in each FFT window [-] */
unsigned winStride;   /* # samples to skip between executing
                         FFT window [-] */
unsigned elemStride;  /* # samples to skip within a window [-] */
unsigned maxNpeaks;   /* max # of fft peaks to report */
unsigned smoothSpan;  /* number of samples per smoothing window
                         (total span) */
unsigned inflectN;    /* # samples to either side of point to be
                         considered for inflection point calculations */
unsigned locMaxN;     /* # samples to either side of point for local
                         maximum searches */
unsigned remMean;     /* nonzero to remove FFT mean before peak search */
double      * df;       /* delta-frequency per bin (resolution) [Hz] */
fft_persist_params_s persistParams;

/* outputs / telemetry */
double *freqBins;     /* frequency labels for FFT output bins [Hz] */
double *fftMag;       /* magnitude of FFT output [?] */
double *fftPhase;     /* phase of FFT output [rads] */
```

```
    long      compTime_ns; /* time required for FFT computation,
                               including copies */
    int       fftRtn;      /* FFT return status */
    int       peaksRtn;    /* findPeaks return status */
    double   *newMeas;   /* boolean, nonzero when new FFT output is present */
    fft_out_s out;       /* structure of output to other modules */
    fft_peaks_s peaks;
    fft_persist_data_s persistData;

    /* private */
    char name[FFT_NAME_MAX_LEN];

    /* circular buffer components */
    /* memory for in-place FFT calculations:
       time-series samples preFFT, half-complex unpack FFT data post*/
    double *fftWin;

    unsigned winCnt; /* number of samples taken since
                          last FFT window calculation */

    double   *buf;        /* circular buffer memory */
    double   *bufHead;    /* newest sample in buffer window */
    double   *bufTail;    /* oldest sample in buffer window */
    double   *bufEnd;     /* end address of buffer after allocation */
    unsigned  nSamples;   /* # samples actually in the buffer */
    unsigned  nBins;      /* # freq. bins in output */

} fft_s;
```

B.2 fft.c

B.2.1 calcPersistence

Listing B.6: *calculate persistence properties for peaks /*

```
int calcPersistence( /*in/out*/ fft_persist_data_s *dat,
                                 double *dstBins,
                                 fft_peaks_s *peaks,
                                 fft_persist_params_s *params ) {
/** @brief calculate persistence properties for peaks */
```

```

    if ( !dat ) {
        return(-1);
    }

    if ( !dstBins || !peaks || !params ) return(-2);

    unsigned int p, b;
    double srcF_left, srcF_right, dstF_left, dstF_right, perc;



    /* decay all bins in amplitude and time */
    /* NB: ignore DC bin at zero index */
    for ( b=1; b<dat->n; ++b){
        /* amplitude */
        if ( dat->amp[b] > 0 ) {
            dat->amp[b] *= params->degradeFactor;
            if ( dat->amp[b] < 0.0 ) dat->amp[b] = 0.0;
        }
        /* time */
        if ( dat->time[b] > 0.0 ) {
            dat->time[b] -= params->src_dt;
            if ( dat->time[b] < 0.0 ) dat->time[b] = 0.0;
        }
    }

    if ( (int)peaks->nPeaks[0] == 0 ) return(0);


    for (p=0; p<(unsigned int)peaks->nPeaks[0]; ++p ) {
        srcF_left  = peaks->freqs[p] - params->src_dF/2.0;
        srcF_right = peaks->freqs[p] + params->src_dF/2.0;

        for ( b=1; b<dat->n; ++b){
            dstF_left  = dstBins[b] - params->dst_dF/2.0;
            dstF_right = dstBins[b] + params->dst_dF/2.0;


            if ( srcF_left > dstF_right ) continue;
        if ( params->doSmear == 0 ) {
            /* all peak data will fit within one destination bin */
            if ( dstF_left <= peaks->freqs[p] && peaks->freqs[p] <= dstF_righ
                dat->time[b]       = dat->time[b] + params->src_dt;
                dat->amp[b]        = peaks->amps[p] * dat->time[b];
                dat->rank[b]       = p+1;
                if ( dat->time[b] <= params->src_dt ) {
                    dat->deltaPhase[b] = 0.0;
                } else {
                    dat->deltaPhase[b] = peaks->phases[p] - dat->prevPhase[b];
                }
                dat->prevPhase[b] = peaks->phases[p];
                break; /* stops searching bins further */
            }
        } else {
            /* peak energy might be spread across multiple destination bins *
            if ( srcF_left < dstF_left && dstF_right < srcF_right ) {
                /* source is much larger than destination */
                perc               = params->dst_dF / params->src_dF;
                dat->time[b]        = dat->time[b] + params->src_dt;
                dat->amp[b]         = (perc * peaks->amps[p])*dat->time[b];
                dat->rank[b]        = p+1;
                if ( dat->time[b] <= params->src_dt ) {
                    dat->deltaPhase[b] = 0.0;
                } else {
                    dat->deltaPhase[b] = peaks->phases[p] - dat->prevPhase[b];
                }
                dat->prevPhase[b] = peaks->phases[p];
            }
            if ( srcF_left < dstF_left && srcF_right <= dstF_right ) {
                /* source is partially within bin from the left edge */
                perc                = (srcF_right - dstF_left)/params->src_dF;
                dat->time[b]        = dat->time[b] + params->src_dt;
                dat->amp[b]         = (perc * peaks->amps[p])*dat->time[b];
                dat->rank[b]        = p+1;
                if ( dat->time[b] <= params->src_dt ) {
                    dat->deltaPhase[b] = 0.0;
                } else {
                    dat->deltaPhase[b] = peaks->phases[p] - dat->prevPhase[b];
                }
                dat->prevPhase[b] = peaks->phases[p];
            }
            if ( srcF_left >= dstF_left && srcF_right <= dstF_right ) {
                /* source is fully within destination bin */
                dat->time[b] = dat->time[b] + params->src_dt;
                dat->amp[b]  = peaks->amps[p] * dat->time[b];
                dat->rank[b] = p+1;
                if ( dat->time[b] <= params->src_dt ) {
                    dat->deltaPhase[b] = 0.0;
                } else {
                    dat->deltaPhase[b] = peaks->phases[p] - dat->prevPhase[b];
                }
                dat->prevPhase[b] = peaks->phases[p];
            }
            if ( srcF_left >= dstF_left && srcF_right > dstF_right ) {
                /* source is partially within destination bin from the right *
                perc             = (dstF_right - srcF_left)/params->src_dF;
                dat->time[b]     = dat->time[b] + params->src_dt;
                dat->amp[b]      = (perc * peaks->amps[p])*dat->time[b];
                dat->rank[b]        = p+1;
                if ( dat->time[b] <= params->src_dt ) {
                    dat->deltaPhase[b] = 0.0;
                } else {
                    dat->deltaPhase[b] = peaks->phases[p] - dat->prevPhase[b];
                }
                dat->prevPhase[b] = peaks->phases[p];
            }

        } /* end smear-else */
    } /* end dest bin for-loop */

} /* end nPeaks for-loop */

/* zero out prevPhase for bins that didn't increase this time */
for ( b=1; b<dat->n; ++b){
    if ( dat->time[b] < params->src_dt ) {
        dat->prevPhase[b] = 0.0;
    }
}
```

```


   return(0);
}
```

B.2.2 checkSmear

Listing B.7: *Compare soure and destination bins set doSmear flag accordingly /*

```
int checkSmear( fft_persist_params_s *params) {
/** @brief Compare soure and destination bins, set doSmear flag accordingly *
   if ( !params ) return(-1);

   if ( fabs((params->src_dF - params->dst_dF)/params->dst_dF) > params->binE
      params->doSmear = 1;
   } else {
      params->doSmear = 0;
   }

   return(0);
}
```

B.2.3 doFFT

Listing B.8: *perform FFT on complex data return magnitude and phase inrads /*

```
int doFFT( /*out*/double *mag, double *phase,
                        /*in*/ double *complexData, unsigned int elemStride, unsign
ned int nBins ) {
/* @brief perform FFT on complex data, return magnitude and phase (in rads) *
/* mag - out - per-bin magnitude abs val complex fft output (must be alloc'd
    phase - out - per-bin phase in radians (must be alloc'd nBins length)
    complexData - in - WILL BE OVERWRITTEN by GSL, must be winLen in length
    elemStride - in - # elements to stride in window, i.e. 1 uses all data in
*/
    int rtn = GSL_SUCCESS;
    double invSqrtN = 1.0/sqrt(winLen);
    double re, im;
    unsigned int b;

    rtn = gsl_fft_complex_radix2_forward( complexData, elemStride, winLen);
    if ( rtn != GSL_SUCCESS) {
            fprintf(stderr,"gsl_fft_complex_radix2_forward() failed with retu
    }

    for ( b=0; b<nBins; ++b) {
        re = REAL(complexData, b) * invSqrtN;
        im = IMAG(complexData, b) * invSqrtN;
        /* absolute magnitude of the complex number */
        mag[b]    = sqrt(re*re + im*im);
        /* ignore magnitudes that are too low and would cause noisy phase calcu
        if ( mag[b] < 1e-4 ) {
            phase[b] = 0.0;
        } else {
            phase[b] = atan2(im, re);
        }
    }

    return(rtn);
}
```

B.2.4 findMax

Listing B.9: *find the max value and its index in a vector of doubles /*

```
int findMax( double *maxVal, unsigned int *maxId, double *sig, unsigned int l
/* @brief find the max value and its index in a vector of doubles */
/* @param[in] sig - the signal */
/* @param[in] len - length of the signal */
/* @param[out] maxVal - maximum value within 'sig' */
/* @param[out] maxId  - index within 'sig' that 'maxVal' can be found */
/* @return zero on succes, less otherwise */

    /* can't work without input */
    if ( !sig ) {
        return(-1);
    }
    /* won't waste time if neither output is provided */
    if ( !maxVal && !maxId ) {
        return(-2);
    }

    if ( len == 1 ) {
        *maxVal = *sig;
        *maxId  = 0;
        return(0);
    }


    double *k, *m = sig;
    unsigned int n;
    for ( k = &sig[1], n=0; k <= &sig[len-1]; ++k, ++n) {
        if ( *k > *m ) {
            m = k;
        }
    }
    if (maxVal) *maxVal = *m;
    if (maxId)  *maxId  = (m - sig);


```

```
    return(0);
}
```

B.2.5 findNmaxSubset

Listing B.10: *find the N max values in a subset of a signal /*

```
int findNmaxSubset( /*out*/unsigned int *foundIds,
                           unsigned int *nFound,
                           /*in*/ double *sig, unsigned int *ids,
                           unsigned int nIds,  unsigned int maxN ) {
/** @brief find the N max values in a subset of a signal */
/*  @param[out] foundIds - pre-allocated vector of top maxN indices of subset
    (ids match index in 'sig', not position within 'ids' vector) */
/* @param[out] nFound - # of maxes actually found */
/* @param[in] sig - signal of amplitudes (remains unchanged) */
/* @param[in] ids - indices of subset of interest */
/* @param[in] nIds - length of 'ids' (length of subset) */
/* @param[in] maxN - maximum number of maxes to find */
/* @return zero on success, less on error */
```

```

    if ( !foundIds || !nFound || !sig || !ids ) {
        fprintf(stderr,"%s: null pointer received!\n", __FUNCTION__);
        return(-1);
    }
    if ( nIds == 0 || maxN == 0 ) {
        fprintf(stderr,"%s: illegal zero parameter received (nIds or maxN)!\n",
                    __FUNCTION__);
        return(-2);
    }

    int myIds[nIds];
    unsigned int k, mK;

    for (k=0;k<maxN;++k) {
        foundIds[k] = 0;
    }

    /* trivial case */
    if ( nIds == 1 ) {
        *nFound = 1;
        foundIds[0] = ids[0];
        return(0);
    }

    /* copy the ids vector so we can modify it without harming calling data */
    for (k=0;k<nIds;++k){ myIds[k] = (int)ids[k]; }

    *nFound = 0;
    while ( *nFound < maxN ) {
        mK = 0;
        while ( myIds[mK] < 0 && mK < nIds) {
            ++mK;
        }
        if ( mK == nIds ) {
            /* all entries in subset are marked found */
            break;
        }
        for (k=0;k<nIds;++k) {
            if ( myIds[k] >= 0 && sig[myIds[k]] > sig[myIds[mK]] ) mK = k;
        }
        foundIds[*nFound] = myIds[mK];
        ++(*nFound);
        myIds[mK] = −1; /* mark entry found */
    }


    return(0);
}
```

B.2.6 findPeaks

Listing B.11: *Find Peaks in an FFT signal /*

```
int findPeaks(/*out*/ unsigned int *nFound,
                      double *outFreq,
                      double *outAmp,
                      double *outPhase,
                      /*in*/ double *inAmp,
                      double *inFreq,
                      double *inPhase,
                      unsigned int len,
                      unsigned int smoothSpan,
                      unsigned int inflectN,
                      unsigned int locMaxN,
                      unsigned int numPeaksToFind) {
/* @brief Find Peaks in an FFT signal */
if ( !nFound || !outFreq || !outAmp || !outPhase ||
      !inAmp  || !inFreq  || !inPhase ) {
  return (-1);
}

if ( len == 0 || smoothSpan == 0 || inflectN == 0 ||
      locMaxN == 0 || numPeaksToFind == 0 ) {
  return(-2);
}

*nFound = 0;

double maxVal;
```

```
    unsigned int k, j, i, dupe;
    int rtn = 0;
    const unsigned int tail = (smoothSpan-1)/2;
    const unsigned int sLen = len - tail - tail;
    unsigned int smoothLen = 0;

    double sAmp[sLen], sFreq[sLen]; /* post-smooth data */
    unsigned int inflectId[sLen], nInflectFound;
    unsigned int nLocMaxFound, isMax;
    unsigned int maxId[sLen], mId;
    unsigned int outIds[sLen];
    unsigned int nSearch;
    double thisAmp[len];

    for (k=0; k<numPeaksToFind; ++k){
        outFreq[k]  = 0;
        outAmp[k]   = 0;
        outPhase[k] = 0;
    }
    /* normalize */
    memcpy(thisAmp, inAmp, len*sizeof(double));
    rtn = findMax( &maxVal, maxId, thisAmp, len );
    if ( rtn != 0 ) {
        fprintf(stderr,"%s: findMax() failed! (rtn = %d, inAmp @ %p, len = %u)\
                __FUNCTION__, rtn, inAmp, len);
        fflush(stderr);
        return(-3);
    }

    for ( k=0; k<len; ++k) {
        thisAmp[k] /= maxVal;
    }


    /* smooth with sliding window */
    /* smooth() doesn't handle the phases,
        so be careful further on indexing the found phases */
    rtn = smooth( sAmp, sFreq, &smoothLen, thisAmp, inFreq, len, smoothSpan );
    if ( rtn != 0 ) {
        fprintf(stderr,"%s: smooth() failed! (rtn=%d)\n",
                __FUNCTION__, rtn );
        return(-4);
    }
    if ( smoothLen != sLen ) {
        fprintf(stderr,"%s: WARNING sLen (%u) != smoothLen (%u)\n",
                __FUNCTION__, sLen, smoothLen );
    }


    /* inflection point search */
    /* inflectId will be subset of smooth indices */
    rtn = inflectNeighborSearch( inflectId, &nInflectFound,
                                 sAmp, smoothLen, inflectN );
    if ( rtn != 0 ) {
        fprintf(stderr,"%s: inflectNeighborSearch() failed! (rtn=%d)\n",
                __FUNCTION__, rtn );
        return(-5);
    }
    if ( nInflectFound == 0 ) {
        nFound = 0;
        return(0);
    }


    /* local maximum search */
    nLocMaxFound = 0;
    for (k=0; k<nInflectFound; ++k ) {
        isMax = 0;
        rtn = localMaxSearch( &isMax, sAmp, smoothLen, inflectId[k], locMaxN );
        if ( rtn != 0 ) {
            fprintf(stderr,"%s: localMaxSearch() failed! (rtn=%d)\n",
                        __FUNCTION__, rtn);
            fprintf(stderr,"\tinflectId[%u] = %u; sAmp = %f; locMaxN = %u\n",
                        k, inflectId[k], sAmp[inflectId[k]], locMaxN );
            return(-6);
        }
        if ( isMax ) {
            maxId[nLocMaxFound] = inflectId[k];
            ++nLocMaxFound;
        }
    }


    if ( nLocMaxFound == 0 ) {
        nFound = 0;
        return(0);
    }

    /* find top n peaks */
    *nFound = 0;
    nSearch = ( numPeaksToFind > nLocMaxFound ) ? nLocMaxFound : numPeaksToFin
    for (k=0; k<nSearch;++k){

        j = 0;
        for(i=0;i<*nFound;++i) {
            if ( outIds[i] == maxId[j] ) {
                ++j;
                i = 0; /* reset search */
            }
        }
        if ( j >= nLocMaxFound ) {
        }
        mId = maxId[j];
        for (j=1; j<nLocMaxFound;++j){
            if ( sAmp[maxId[j]] > sAmp[mId] ) {
                dupe = 0;
                for (i=0;i<*nFound;++i) {
                    if ( maxId[j] == outIds[i] ) {
                        dupe = 1;
                        break;
                    }
                }
                if ( !dupe ) {
                    mId = maxId[j];
                }
            }
        } /* end nLocMaxFound for */
        outIds[*nFound] = mId;
        ++(*nFound);
    } /* end numPeaksToFind for */

    /* fill output arrays */
    for ( k = 0; k < *nFound; ++k ) {
        outFreq[k]  = sFreq[outIds[k]];
        outAmp[k]   = sAmp[outIds[k]];
        outPhase[k] = inPhase[outIds[k]+tail];
    }

    return (0);
}
```

B.2.7 inflectNeighborSearch

Listing B.12: *Inflection Search within a neighborhood /*

```
int inflectNeighborSearch( /*outs*/ unsigned int *inflectId, unsigned int *nF
                           /*ins*/  double *sig, unsigned int len, unsigned i
    /** @brief Inflection Search within a neighborhood */
    /** @brief search within a neighborhood of back-differenced values for inf
                first, calculates backdifferences, then find which points of th
                signal (sig) are peaks: rising from left, falling to right.
                I.e. all n slopes to left of point must be positive and all n s
int
                must be negative
        @param inflectId: (out) indices of 'sig' which are inflection points;
                            must be allocated to size 'len'-1 before call
        @param nFound:    (out) number of inflection points found (scalar)
        @param sig:       (in)  signal to be back-differenced (sig remains una
        @param len:       (in)  length of 'sig' and 'lbl' in samples (not byte
        @param n:         (in)  single-sided neighborhood for search (i.e. ful
2*n + 1)
    */

    if ( !inflectId || !nFound || !sig ) {
        fprintf(stderr,"%s: inflectId, nFound and sig are required; at least on
                __FUNCTION__);
        return (-1);
    }
    if ( len == 0 ) {
        fprintf(stderr,"%s: len is zero!\n",__FUNCTION__);
        return(-2);
    }
    if ( n == 0 ) {
        fprintf(stderr,"%s: neighborhood size 'n' is zero!\n", __FUNCTION__);
        return(-3);
    }
    /* so named because rightSlope[i] is slope between sig[i] and sig[i+1] */
    double rightSlope[len-1];
    unsigned int k,j, itr;
    char inflectFound;

    *nFound = 0;


    /* backdifference */
    for (k=0; k<(len-1);++k) {
        rightSlope[k] = sig[k+1]-sig[k];
    }

    /* perform neighborhood search */
```

```
  k = 0;
  itr=0;
  while ( k < (len-1) && itr < 1e4) {

     ++itr;
     inflectFound = 1;
     /* search left side */
     for ( j=1; j<=n; ++j ) {
        if ( ((int)k-(int)j) < 0 ) {
           if ( n == 1 ) inflectFound = 0;
           break;
        } else {
           if ( rightSlope[k-j] < 0 ) {
              /* need slopes to be positive on the left, but we're not */
              inflectFound = 0;
              break;
           }
        }
     }
     if ( !inflectFound ) {
            ++k;
            continue;
        }
        /* search right side */
        for ( j=0; j<=(n-1); ++j ){
            if ( (k+j) > (len-2) ) {
                /* don't penalize for running off the end of signal to the right
                break;
            }
            if ( rightSlope[k+j] > 0 ) {
                /* need slopes to be negative on the right, but we're not */
                inflectFound = 0;
                break;
            }
        }
        if ( inflectFound ) {
            inflectId[*nFound] = k;
            ++(*nFound);
            k = k + n;
        } else {
            ++k;
        }
    }

    return(0);
}
```

B.2.8 localMaxSearch

Listing B.13: *check if 'idx' is local maximum with 'n' neighboring samples /*

```
int localMaxSearch( /*outs*/unsigned int *isMax,
                    /*ins*/double *sig, unsigned int len,
                           unsigned int idx, unsigned int n ) {
/** @brief check if 'idx' is local maximum with 'n' neighboring samples */
/*  @param[out] isMax - nonzero if 'idx' is index of local maximum, zero othe
/*  @param[in]  sig   - vector of signal samples (remains unchanged) */
/*  @param[in]  len   - length of 'sig' and 'lbl' */
/*  @param[in]  idx   - index of sample to check */
/*  @param[in]  n     - number of samples in neighborhood to each side */
/*  @return zero on success, less on error */

    if ( !isMax || !sig  ) {
        fprintf(stderr,"%s: null pointer received.\n", __FUNCTION__);
        return(-1);
    }
    if ( len < 2 ) {
        fprintf(stderr,"%s: signal length is too short for search (%d)\n",
                  __FUNCTION__, len);
        return(-2);
    }
    if ( idx >= len ) {
        fprintf(stderr,"%s: target index (%d) is beyond signal length (%d).\n",
                  __FUNCTION__, idx, len );
        return(-3);
    }

    unsigned int k;

    *isMax = 0; /* start assuming it's not */

    /* NaN and Infs are ignored */
    if ( !isfinite(sig[idx]) ) return(0);

    /* search left */
    for (k=1; k<=n && ((int)idx-(int)k)>=0; ++k) {
        if ( isfinite(sig[idx-k]) && sig[idx-k] >= sig[idx] ) return(0);
    }
    for (k=1; k<=n && (idx+k)<len; ++k) {
        if ( isfinite(sig[idx+k]) && sig[idx+k] > sig[idx] ) return(0);
    }

    *isMax = 1;
    return(0);
}
```

B.2.9 smooth

Listing B.14: *sliding window smoothing*

```
int smooth( /*out*/double *outSig, double *outLbl,
                   unsigned int *outLen,
            /*in*/ double *inSig,  double *inLbl,
                   unsigned int inLen, unsigned int win) {
/** @brief sliding window smoothing
    @param outSig[out] output signal after smoothing, must be pre-allocated
    @param outLbl[out] output label for each signal bin, must be pre-allocate
    @param outLen[out] length of outputs
    @param inSig[in] input signal prior to smoothing
    @param inLbl[in] input signal labels
    @param inLen[in] length of input signal
    @param win[in] length of smoothing window
    @return zero on success, less on error
*/
   if ( !outSig || !outLbl || !outLen || !inSig || !inLbl ) {
      fprintf(stderr,"%s: null pointer received. \n", __FUNCTION__);
      return(-1);
   }
   if ( inLen == 0 ) {
      fprintf(stderr,"%s: inLen is too short\n", __FUNCTION__);
```

```
        return(-2);
    }
    if ( win > inLen ) {
        fprintf(stderr,"%s: sliding window size is less than length of signal.\
                    __FUNCTION__ );
        return(-3);
    }
    if ( win % 2 == 0 ) {
        fprintf(stderr, "%s: window size must be odd.\n", __FUNCTION__);
        return(-4);
    }

    int tail = (win-1)/2;
    int k;
    int j;

    *outLen = inLen - tail*2;

    for ( k=tail; k<((int)inLen-tail); ++k ) {
        outLbl[k-tail] = inLbl[k];
        outSig[k-tail] = 0.0;
        for ( j=-tail; j<=tail; ++j) {
            outSig[k-tail] += inSig[k+j];
        }
        outSig[k-tail] /= (double)win;
    }

    return(0);
}
```

B.3 fts.c

B.3.1 fts

```
System *fts(domain_t *domain ) {
  System *sys = fspNewSystem( domain, "FTS", "Force Torque Sensor", sizeof(ft

  /* function pointers */
  sys->init = fts_init;
  sys->drt  = fts_drt;

  return(sys);
}
```

B.3.2 fts_drt

Listing B.15: *fts_drtSystemsys* {

```
static int fts_drt(  System* sys) {
    fts_s *FTS = (fts_s*)sys->data;
    fft_s *FFT;
    unsigned int d,  /*n, b,*/ k;
    /*long unsigned int subWinSize;*/
    double *sample;
    struct timespec start_ts, end_ts;
#if 0
    gsl_complex_packed_array winOutCoeffs;
#endif

```

```
  /* compute current FTS pose in servicer base from engine truth */
  Qmult_rt( FTS->q_fts2base, FTS->q_i2base, 1, FTS->q_i2fts, -1 );

  /*winOutCoeffs  = calloc(2*(FFT->winLen), sizeof(double));*/

  for (d=0; d<NUMDOF; ++d ) {
     FFT = &(FTS->FFT[d]);

     clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start_ts);

     /* ************************************* */
     /* ************************************* */
     /* add latest data to buffer */
     /* 1. update circbuf pointers and count */
     ++(FFT->nSamples);
     ++(FFT->winCnt);
     if ( FFT->nSamples > 1 ) {
        if ( FFT->bufHead >= FFT->bufEnd ) {
           FFT->bufHead = &(FFT->buf[0]);
        } else {
            ++(FFT->bufHead);
        }
    }
    if ( FFT->nSamples > FFT->winLen) {
        if ( FFT->bufTail >= FFT->bufEnd ) {
            FFT->bufTail = &(FFT->buf[0]);
        } else {
            ++(FFT->bufTail);
        }
    }


    /* 2. insert new data */
    if ( d < 3 ) {
        /* force DOF */
        FFT->bufHead[0] = FTS->forc_j[d];
    } else {
        FFT->bufHead[0] = FTS->torq_j[d-3];
    }

    /* ************************************ */
    /* ************************************ */
    if ( FFT->winCnt >= FFT->winStride ) {
       FFT->winCnt = 0;
       FFT->newMeas[0] = 1;

       memset(FFT->fftWin, 0, FFT->winLen*2*sizeof(double));

       if ( FFT->nSamples <= FFT->winLen ) {     /* circular buffer hasn't w

          for (k=0; k< FFT->nSamples; ++k ) {
             REAL(FFT->fftWin, k) = FFT->bufTail[k];
             IMAG(FFT->fftWin, k) = 0.0;
          }
       } else {
          /* circular buffer has wrapped, copy in two parts */
          k = 0;
          for ( sample = FFT->bufTail; sample <=  FFT->bufEnd; ++sample, ++
             REAL(FFT->fftWin, k) = *sample;
             IMAG(FFT->fftWin, k) = 0.0;
```

```
            }
            for ( sample = FFT->buf; sample < FFT->bufTail; ++sample, ++k) {
                REAL(FFT->fftWin, k) = *sample;
                IMAG(FFT->fftWin, k) = 0.0;
            }
        }

        /* remove mean from signal prior to FFT if desired */
        if ( FFT->remMean )  {
            double thisMean, tot =0.0;
            for (k=0;k<FFT->nBins;++k) {
                tot += REAL(FFT->fftWin, k);
            }
            thisMean = tot / (double)FFT->winLen;
            for (k=0;k<FFT->nBins;++k) {
                REAL(FFT->fftWin,k) -= thisMean;
            }
        }

#if 0
```

```
            if ( (FFT->fftRtn = gsl_fft_complex_radix2_forward( FFT->fftWin , FFT
inLen)) != GSL_SUCCESS) {
                fprintf(stderr,"gsl_fft_complex_radix2_forward() failed with retu
n);
            }
            double invSqrtN = 1.0/sqrt(FFT->winLen);
            double re, im;
            for ( b=0; b<FFT->nBins; ++b) {
                re = REAL(FFT->fftWin, b) * invSqrtN;
                im = IMAG(FFT->fftWin, b) * invSqrtN;
                /* absolute magnitude of the complex number */
                FFT->fftMag[b]    = sqrt(re*re + im*im);
                /* ignore magnitudes that are too low and would cause noisy phase
                if ( FFT->fftMag[b] < 1e-5 ) {
                    FFT->fftPhase[b] = 0.0;
                } else {
                    FFT->fftPhase[b] = atan2(im, re);
                }
            }
#else
```

```
            FFT->fftRtn = doFFT( FFT->fftMag, FFT->fftPhase,
                                 FFT->fftWin, FFT->elemStride,
                                 FFT->winLen, FFT->nBins );
#endif

        } else {
            FFT->newMeas[0] = 0;
        }
#if 0
        /* ************************************** */
        /* Determine max freq & amplitude         */
        /* ************************************** */
        unsigned int maxId = 1; /* bin zero is DC - ignore it */
        for ( b=maxId+1; b<FFT->nBins; ++b) {
            /* strictly less-than, favoring lower frequencies */
            if ( FFT->fftMag[b] > abs(FFT->fftMag[maxId]) ) {
                maxId = b;
            }
        }
        FTS->maxFreqAmp[d] = abs(FFT->fftMag[maxId]);
        FTS->maxFreq[d]      = FFT->freqBins[maxId];
#else
        /* ************************************** */
        /* Find Peaks                             */
        /* ************************************** */
        unsigned nFound;
        FFT->peaksRtn = findPeaks( &nFound, FFT->peaks.freqs, FFT->peaks.amps,
                                   FFT->fftMag, FFT->freqBins, FFT->fftPhase, F
                                   FFT->smoothSpan, FFT->inflectN, FFT->locMaxN

        FFT->peaks.nPeaks[0] = (double)nFound;
#endif
        clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end_ts);

        FFT->compTime_ns  = (long)(end_ts.tv_sec - start_ts.tv_sec)*1000000000;
        FFT->compTime_ns += (long)end_ts.tv_nsec - (long)start_ts.tv_nsec;

        /* ************************************** */
        /* Calculate Persistence Stats            */
        /* ************************************** */
        int rtn;
        rtn = calcPersistence(&(FFT->persistData), FFT->freqBins, &(FFT->peaks
ams) );
        if ( rtn != 0 ) {
            fprintf(stderr,"%s: DOF %d persistence calculation failed! (rc = %d)
, rtn );
        }

    } /* end DOF for-loop */


    return(FSP_NO_ERROR);
}
```

B.3.3 fts_init

Listing B.16: *fts_initSystemsys* {

```
static int fts_init( System* sys) {
    fts_s *FTS = (fts_s*)sys->data;
    double Fs, delta;
    unsigned winLen, winStride, elemStride, d, f, nBins, maxPeaks;
    unsigned int smoothSpan, inflectN, locMaxN, remMean;
    fft_s *FFT;
    char *DOF_LAB[] = {"FX", "FY", "FZ", "MX", "MY", "MZ"};
    fft_persist_params_s *PP;
    fft_persist_data_s   *PD;

    /* FFT parameters */
    FSPgetParamInt( FTS->en_fft ,"MANIP.FTS.FFT.en",          "nonzero to enable
    FSPcopyParam1d( Fs,          "MANIP.FTS.FFT.Fs",          "sampling frequenc
    FSPgetParamInt( winLen,      "MANIP.FTS.FFT.winLen",      "# samples in each
    FSPgetParamInt( winStride,   "MANIP.FTS.FFT.winStride",   "# samples to skip
FT window [-]");
    FSPgetParamInt( elemStride, "MANIP.FTS.FFT.elemStride", "# samples to skip
");
    FSPgetParamInt( maxPeaks,    "MANIP.FTS.FFT.maxNpeaks",   "max # of fft peak
    FSPgetParamInt( smoothSpan, "MANIP.FTS.FFT.smoothSpan", "# samplies in spa
w");
    FSPgetParamInt( inflectN,    "MANIP.FTS.FFT.inflectN",    "single-side neigh
on point search");
    FSPgetParamInt( locMaxN,     "MANIP.FTS.FFT.locMaxN",     "single-side neigh
ximum search");
    FSPgetParamInt( remMean,     "MANIP.FTS.FFT.remMean",     "nonzero to remove
k search");

    /* FTS parameters */
    FSPgetParamStr( FTS->jointName, "MANIP.FTS.joint_name", "name of FTS joint
    FSPgetParamStr( FTS->outerBodyName, "MANIP.FTS.outerBodyName", "name of th
he FTS joint in VSD");

    /* inputs */
    fspSetVarExpandTags( sys, 1, FTS->jointName );
    FSPgetInputVec( FTS->forc_j, 3, "JOINTS.%.force_j", "force in joint frame
    FSPgetInputVec( FTS->torq_j, 3, "JOINTS.%.torq_j", "torque in joint frame
    FSPgetInputVec( FTS->q_i2base, 4, "MPROP.base.q_i2b", "inertial orientatio
se body");

    fspJointInfo_s jtInfo;
    int n, *matched_ids;
    n = fspJointFindId( FTS->jointName, &matched_ids );
    if ( n == 0 ) {
        fprintf(stderr,"\n\n\nFailed to find joint named '%s'!!!!\n", FTS->joint
        fprintf(stderr,"Exiting!\n");
        return(FSP_PARAM_MISSING);
    }
    if ( n > 1 ) {
        fprintf(stderr,"\n\n\nERROR found multiple joints matching pattern '%s'
);
        fprintf(stderr,"Found joint IDs: ");
        for (int i = 0; i < n; i++ ) {
            fprintf(stderr,"%d ",matched_ids[i]);
        }
        fprintf(stderr,"\n\n");
        return(FSP_PARAM_MISSING);
    }
    FTS->jointId = matched_ids[0]; /* assumes there's only one */
    fspJointGetInfo( FTS->jointId, &jtInfo );
    FTS->outerBodyId = jtInfo.outerBodyID;
    /* outerBodyName = fspBodyGetName( FTS->outerBodyId ); */
    fprintf(stderr,"\n\n%s: FTS outer body ID is %d and name is '%s'.\n\n", __F
erBodyId, FTS->outerBodyName );
    fspSetVarExpandTags( sys, 1, FTS->outerBodyName );
    FSPgetInputVec( FTS->q_i2fts, 4, "MPROP.%.q_i2b", "FTS outer body inertial

    /* outputs */
    FSPnewOutput( FTS->q_fts2base, 4,       1, "q_fts2base", FSP_LOG1x, "(engin
om FTS measurement frame to servicer base frame");
#if 0
    FSPnewOutput( FTS->maxFreq,     NUMDOF, 1, "maxFreq",    FSP_LOG1x, "per-DO
FT output (if enabled) [Hz]");
    FSPnewOutput( FTS->maxFreqAmp, NUMDOF, 1, "maxFreqAmp", FSP_LOG1x, "per-DO
 freq [-]");
#endif

    fspSetDRTsampleTime( sys, 1.0 / Fs );

    /* allocate per-DOF memory for FFTs */
    for ( d = 0; d < NUMDOF; ++d ) {
        FFT = &(FTS->FFT[d]);
        PP  = &(FTS->FFT[d].persistParams);
        PD  = &(FTS->FFT[d].persistData);

        nBins = (unsigned)(( winLen / 2 ) + 1);
        FFT->nBins = nBins;

        /* allocate */
        FFT->fftWin   = calloc(winLen*2, sizeof(double));      /* (input) indiv
lex data to be sent to FFT} */
        /* FFT->buf      = calloc(winLen, sizeof(double)); */

        FFT->fftWinCopy = calloc(winLen*2, sizeof(double));

        /* persistence parameters */
        FSPgetParamInt( PP->dst_nBins,     "MANIP.FTS.PERSIST.dst_nBins",
"number of bins in de
stination FFT");
        FSPcopyParam1d( PP->dst_dF,         "MANIP.FTS.PERSIST.dst_dF",
"destination FFT delt
a-frequency per bin [Hz]");
```

```
        FSPgetParamInt( PP->src_nBins,      "MANIP.FTS.PERSIST.src_nBins",
    "number of bins in so
urce FFT");
        FSPcopyParam1d( PP->src_dF,         "MANIP.FTS.PERSIST.src_dF",
    "source FFT delta-fre
quency per bin [Hz]");
        FSPcopyParam1d( PP->src_dt,         "MANIP.FTS.PERSIST.src_dt",
    "time between source
samples, [sec]");
        FSPcopyParam1d( PP->degradeFactor, "MANIP.FTS.PERSIST.degradeFactor", "
tude degradation factor [0:1]");
        FSPcopyParam1d( PP->binEpsilon,     "MANIP.FTS.PERSIST.binEpsilon",
    "bin size percentage
epsilon for smearing check");
        FSPgetParamInt( PP->doSmear,        "MANIP.FTS.PERSIST.doSmear",
    "nonzero if the sourc
e peaks will be smeared across mult dest bins");
        PD->n = PP->dst_nBins;

        /* copy locally */
        FFT->winLen      = winLen;
        FFT->winStride   = winStride;
        FFT->elemStride = elemStride;
        FFT->Fs          = Fs / (double)elemStride;
        FFT->maxNpeaks   = maxPeaks;
        FFT->smoothSpan = smoothSpan;
        FFT->inflectN    = inflectN;
        FFT->locMaxN     = locMaxN;
        FFT->remMean     = remMean;

        /* declare as telemetry / output*/
        fspSetVarExpandTags(sys, 1, DOF_LAB[d]);
        FSPnewOutput( FFT->freqBins,      nBins,      1, "%.freqBins",
    FSP_LOG1x, "% freque
ncy labels for FFT output bins [Hz]");
        FSPnewOutput( FFT->fftMag,        nBins,      1, "%.fftMag",
    FSP_LOG1x, "% magnit
ude of FFT output [?]");
        FSPnewOutput( FFT->fftPhase,      nBins,      1, "%.fftPhase",
    FSP_LOG1x, "% phase
```

```
of FFT output [rads]");
        fspTelemetry( &FFT->compTime_ns, FSP_LONG,   1, "%.compTime_ns", FSP_LOG0,  "time required for FFT computation, including copies, [nsec]");
        fspTelemetry( &FFT->fftRtn,       FSP_INT,    1, "%.fftRtn", FSP_LOG0,  "FFT alg. return status");
        fspTelemetry( &FFT->peaksRtn,     FSP_INT,    1, "%.peaksRtn", FSP_LOG0,  "findPeaks() return status");
        FSPnewOutput( FFT->buf,             winLen,     1, "%.buf", FSP_LOG0,  "Input buffer to FFT");
        FSPnewOutput( FFT->newMeas,                   1,1, "%.newMeas", FSP_LOG1x, "nonzero when new FFT output is available [bool]");
        FSPnewOutput( FFT->peaks.freqs, maxPeaks, 1, "%.peakFreqs", FSP_LOG1x, "Frequencies of peaks [Hz]");
        FSPnewOutput( FFT->peaks.amps,  maxPeaks, 1, "%.peakAmps",
    FSP_LOG1x, "Normalized
 Amplitudes of peaks [-]");
        FSPnewOutput( FFT->peaks.phases, maxPeaks,1, "%.peakPhases",
    FSP_LOG1x, "Phases of
peaks [Rad]");
        FSPnewOutput( FFT->peaks.nPeaks,            1, 1, "%.nPeaks",
    FSP_LOG1x, "number
 of peaks found");
        FSPnewOutput( PD->amp,                 PD->n, 1, "%.PERSIST.amp",
    FSP_LOG1x, "persiste
nt amplitude (n  length)");
        FSPnewOutput( PD->time,                PD->n, 1, "%.PERSIST.time",
    FSP_LOG1x, "persiste
nt time (n length)");
        FSPnewOutput( PD->deltaPhase,          PD->n, 1, "%.PERSIST.deltaPhase",
n phase of this bin's persistent phase (n length)");
        FSPnewOutput( PD->prevPhase,           PD->n, 1, "%.PERSIST.prevPhase",
    FSP_LOG0,  "phase in
 this bin from previous sample (n length)");
        FSPnewOutput( PD->rank,                PD->n, 1, "%.PERSIST.rank",
FSP_LOG1x, "rank of
peak (1 being largest peak) of bins (n length)");

        /* debug */
        /* fspTelemetry( FFT->fftWinCopy,  FSP_DOUBLE, winLen, "%.fftInput", FS
 of samples sent into FFT"); */

        /* compute frequency bin labels */
        delta = 1.0/(winLen/2.0) * (Fs/2.0);
        FFT->freqBins[0] = 0.0;
        for ( f = 1; f < FFT->nBins; ++f ) {
            FFT->freqBins[f] = (double)f * delta - 0.5*delta; /* center freq. of
        }


        /* per-DOF initialization */
        FFT->nSamples = 0;
        FFT->bufTail  = &(FFT->buf[0]);
        FFT->bufHead  = &(FFT->buf[0]);
        FFT->bufEnd   = &(FFT->buf[FFT->winLen-1]);
```

```

    } /* end of DOF for-loop */

    /* ---------------- */ FSP_assemble_IO_finished();/* ---------------- */



    return (FSP_NO_ERROR);
}
```

B.4 manip.c

B.4.1 manip

```
System *manip( domain_t *domain )
{
    System *sys = fspNewSystem(domain, "MANIP", "Manipulator Controller", size

    sys->add  = NULL;
    sys->init = &manip_init;
    sys->rt   = &manip_rt;
    sys->drt  = &manip_drt;
    sys->exit = NULL;
    sys->log  = NULL;

    return( sys );
}
```

B.4.2 manip_init

Listing B.17: *manip_initSystemsys*

```
static int manip_init( System *sys )
{
    manip_s *manip = (manip_s*)sys->data;
    int         i;
    char        str[256];
    int         logRate; /* computed from engine's print interval and desired log
    double      printInt, logFreq;

    /* workspace paremeters */
    FSPgetParamVec( manip->period, 1,   "period",        "drt period, seconds")
    fspSetDRTsampleTime(sys, manip->period[0]);


    FSPgetParamInt( manip->enHardStop,  "CONFIG.en_hard_stop", "flag, nonzero
 stops");
    FSPgetParamInt( manip->enTorqLimit, "CONFIG.en_torq_lim",  "flag, nonzero
ue limits");
    FSPgetParamInt( manip->enRateLimit, "CONFIG.en_rate_lim",  "flag, nonzero
 limits");
    FSPgetParamInt( manip->enOutFilt,   "CONFIG.en_output_filter", "flag, nonz
int command output filter should be used.");
    /* FSPgetParamInt( manip->enFts,       "CONFIG.en_fts",         "flag, nonz
torque sensor"); */
    FSPgetParamInt( manip->isLocked,    "locked",                "nonzero if
e locked");

    FSPcopyParam1f( printInt, "ENG.INTEGRATOR.printint", "Integrator logging p
    FSPcopyParam1f( logFreq,  "logFreq",                    "Desired logging freq
lator module");
    logRate = (int)round(logFreq*printInt);
    if ( logRate < 1 ) logRate = 1;
    fprintf(stderr,"%s: ====================================================
__);
    fprintf(stderr,"%s: printInt = %f\n", __FUNCTION__, printInt );
    fprintf(stderr,"%s: logFreq  = %f\n", __FUNCTION__, logFreq );
    fprintf(stderr,"%s: logRate = %d (as int) and %f (as double)\n", __FUNCTIO
eq*printInt) );
    fprintf(stderr,"%s: ====================================================
__);

    FSPgetParamInt( manip->nLinks,       "n_links", "number of actuated links i

    i = fspJointFindId( "manip_joint*", &(manip->jid));
    if ( manip->nLinks > i ) {
        fprintf(stderr,"%s: ERROR: manip n_links = %d but only found %d joint I
                __FUNCTION__, manip->nLinks, i );
        return(FSP_PARAM_MISSING);
    }

    if ( manip->nLinks > NUM_LINKS_MAX ) {
        fprintf(stderr,"%s: ERROR: manipulator MANIP has %d links, but module c
imum.\n",
                    __FUNCTION__, manip->nLinks, NUM_LINKS_MAX );
        return( FSP_PARAM_MISSING );
    }

    FSPgetParamVec( manip->Kp,             manip->nLinks, "kp",
    "proportional gain, per
 joint");
    FSPgetParamVec( manip->Ki,             manip->nLinks, "ki",
    "integral gain, per join
 t");
    FSPgetParamVec( manip->Kd,             manip->nLinks, "kd",
    "derivative gain, per jo
 int");
    FSPgetParamVec( manip->jntTorqMax,     manip->nLinks, "jnt_torq_max", "per
m");
```

```
    FSPgetParamVec( manip->jntRateMax,      manip->nLinks, "jnt_rate_max", "per
/sec");
    FSPgetParamMat( manip->jntHardStop, 2, manip->nLinks, "hard_stop",
"per-joint hard stop ang
les RADIANS [ min; max ]");
    FSPgetParamVec( manip->jntAntiWindup,  manip->nLinks, "antiWindup",
"per-joint anti-windup l
imit for error integral");
    FSPgetParamVec( manip->jntLockCmd,      manip->nLinks, "jnt_lock_cmd", "nor
t to LOCKED state");
    FSPgetParamVec( manip->newLockCmd,      1,             "new_lock_cmd", "fla
 jntLockCmd values have changed");



    /* inputs */
    /* FSPgetInputVec( manip->Fcmd,              6,                    "ARMCTRL.Fcmd_",
"reactVel mode s
tacked force-torque command"); */
    FSPgetInputVec( manip->ang_cmd,         manip->nLinks, "ARMCTRL.ang_cmd",
"commanded manipulat
```

```
or joint angles, radians");
    FSPgetInputVec( manip->rate_cmd,       manip->nLinks, "ARMCTRL.rate_cmd",
or joint rates, rad/sec");
    FSPgetInputVec( manip->torq_cmd,       manip->nLinks, "ARMCTRL.torq_cmd",
or joint torques (TORQ_CMD mode only), Nm");
    FSPgetInputVec( manip->mode,           1,             "ARMCTRL.manip_mode"
ller (0=DISABLE,1=ANGLE)");

    manip->driveAxis = calloc( manip->nLinks, sizeof(int));

    for ( i = 0; i < manip->nLinks; i++ ) {
        fspJointInfo_s jInfo;

        fspJointGetInfo( manip->jid[i], &jInfo);
        fspSetVarExpandTags( sys, 1, jInfo.name );

#if MANIP_VSD_VERBOSE
        fprintf(stderr,"%s: manip->nLinks = %d, i = %d\n", __FUNCTION__, manip-
        fprintf(stderr,"%s: joint %d: info.name = '%s' info.axis = ( %f %f %f )
,
                 jInfo.name, jInfo.axis_j[0], jInfo.axis_j[1], jInfo.axis_j[2] )
        fprintf(stderr,"%s: joint %d: manip->jid = %d; jInfo.id = %d\n", __FUNC
d[i], jInfo.id );
#endif

        if ( jInfo.axis_j[0] > 0.0 ) {
            manip->driveAxis[i] = 0;
        } else {
            if ( jInfo.axis_j[1] > 0.0 ) {
                manip->driveAxis[i] = 1;
            } else {
                if ( jInfo.axis_j[2] > 0.0 ) {
                    manip->driveAxis[i] = 2;
                } else {
                    fprintf(stderr,"%s: WARNING joint %d axis_j = [ %1.1f %1.1f %1
!\n",
                            __FUNCTION__, i, jInfo.axis_j[0], jInfo.axis_j[1], jIn
                }
            }
        }

        if ( manip->isLocked == 0 ) {
          snprintf(str, 256, "current joint angle (inner to outer) for manip j
, i, jInfo.name );
          FSPgetInputVec( manip->ang[i], 1, "JOINTS.%.angle", str );

          snprintf(str, 256, "current joint rates for manip joint %d '%s', [ra
);
          FSPgetInputVec( manip->rate[i], 3, "JOINTS.%.dw", str );
        }

        /*debug*/
#if MANIP_VSD_VERBOSE
        fprintf(stderr,"%s: manip->ang[%d]  = %p\n", __FUNCTION__, i, manip->an
        fprintf(stderr,"%s: manip->rate[%d] = %p\n", __FUNCTION__, i, manip->ra
        fflush(stderr);
#endif
    }

    /* outputs */
    FSPnewOutput( manip->ang_out,        manip->nLinks, 1, "ang", logRate, "current joint angle, rads");
    FSPnewOutput( manip->ang_cmd_out,    manip->nLinks, 1, "ang_cmd", logRate, "commanded joint angle, rads");
    FSPnewOutput( manip->rate_out,       manip->nLinks, 1, "rate", logRate, "current joint rate, rad/sec");
    FSPnewOutput( manip->rate_cmd_out,   manip->nLinks, 1, "rate_cmd", logRate, "commanded joint rate, rad/sec");
    FSPnewOutput( manip->torq,           manip->nLinks, 1, "torq", logRate, "applied joint torques, Nm");
    FSPnewOutput( manip->ang_err,        manip->nLinks, 1, "ang_err", logRate, "error in joint angle, rads");
    FSPnewOutput( manip->rate_err,       manip->nLinks, 1, "rate_err", logRate, "error in join
t rate, rad/sec");
    FSPnewOutput( manip->hardStopFlag,  manip->nLinks, 1, "hardStopFlag",
logRate, " nonzero indi
cates a joint hit a hard stop limit");
    FSPnewOutput( manip->torqLimitFlag, manip->nLinks, 1, "torqLimitFlag", log
ates a joint was torque limited");
    FSPnewOutput( manip->rateLimitFlag, manip->nLinks, 1, "rateLimitFlag", log
ates a joint was rate limited");
    FSPnewOutput( manip->jntErrInt,     manip->nLinks, 1, "jntErrInt",
logRate, "per-joint err
or integral term");
    FSPnewOutput( manip->mode_out,      1,             1, "mode",
logRate, "current manip
 module mode (0=DISABLE,1=JOINT)");

#if 0
    /* set initial joint lock state */
    for ( i=0; i < manip->nLinks; ++i ) {
        fprintf(stderr,"%s: setting joint %d, id = %d, to state %s.\n", __FUNCT
                manip->jid[i], ((manip->jntLockCmd[i] > 0) ? "LOCKED":"UNLOCKED
        if ( manip->jntLockCmd[i] > 0 ) {
            if ( 0 != (rc = fspJointModifyType( manip->jid[i], "LOCKED")) ){
                fprintf(stderr,"%s: fspJointModifyType(LOCKED) failed for joint %
TION__, i, rc);
            }
        } else {
            if ( 0 != (rc = fspJointModifyType( manip->jid[i], "REV1"))) {
                fprintf(stderr,"%s: fspJointModifyType(REV1) failed for joint %d
ON__, i, rc);
            }
        }
    }
#endif


    return (FSP_NO_ERROR);
}
```

B.5 armCtrl.c

B.5.1 Quat2Rot

Listing B.18: *Convert quaternion to rotation matrix /*

```
int Quat2Rot( double DCM[3][3], double *q_in, int inv_flag ) {
  /*** @brief Convert quaternion to rotation matrix */
  /* q[0]-q[2] is the vector, q[3] is the scalar */
   double   in, out[3][3], skew[3][3], TwoQ4;

   if ( !DCM || !q_in ) {
      return(-1);
   }

   long double q[4], qnorm;
   extern long double sqrtl(long double x);
   qnorm = sqrtl( q_in[0]*q_in[0] +
                  q_in[1]*q_in[1] +
                  q_in[2]*q_in[2] +
                        q_in[3]*q_in[3]  );

    q[0] = q_in[0]/qnorm;
    q[1] = q_in[1]/qnorm;
    q[2] = q_in[2]/qnorm;
    q[3] = q_in[3]/qnorm;

    TwoQ4 = 2.0*q[3];
    if ( inv_flag < 0 )  TwoQ4 = -TwoQ4;
    in = q[3]*q[3]-Dot_rt(q,q);
    Outer_rt( out, q, q );
    Skewsym_rt( skew, q );
    DCM[0][0] = in + 2.0*out[0][0];
    DCM[0][1] = 2.0*out[0][1] - TwoQ4*skew[0][1];
    DCM[0][2] = 2.0*out[0][2] - TwoQ4*skew[0][2];
    DCM[1][0] = 2.0*out[1][0] - TwoQ4*skew[1][0];
    DCM[1][1] = in + 2.0*out[1][1];
    DCM[1][2] = 2.0*out[1][2] - TwoQ4*skew[1][2];
    DCM[2][0] = 2.0*out[2][0] - TwoQ4*skew[2][0];
    DCM[2][1] = 2.0*out[2][1] - TwoQ4*skew[2][1];
    DCM[2][2] = in + 2.0*out[2][2];
    return(0);
}
```

B.5.2 armAngRateMode_exec

Listing B.19: *armAngRateMode_execarmCtrl_sARM*{

```
int armAngRateMode_exec(armCtrl_s *ARM){
    if ( !ARM ) return (-1);

    angRate_s *DATA = &ARM->angRateData;
    unsigned int k;

    /* on new command, recalculate rate command and errors */
    if ( ARM->new_theta_des[0] > 0 ) {

        if ( ARM->t_slew[0] <= 0 ) {
            fprintf(stderr,"\e[1m\e[31m%s: Error - received t_slew event less th
f)\e[0m\e[39m\n",
                  __FUNCTION__, ARM->t_slew[0] );
            fflush(stderr);
            for ( k=0; k < (unsigned int)ARM->nLinks; k++ ) {
                ARM->ang_des[k]   = ARM->ang[k][0];
                ARM->ang_cmd[k]   = ARM->ang[k][0];
            }
            memset( ARM->rate_cmd,       0, ARM->nLinks * sizeof(double) );
            memset( ARM->ang_cmd_delta, 0, ARM->nLinks * sizeof(double) );
            memcpy( DATA->ang_cmd_last, ARM->ang_cmd, ARM->nLinks *sizeof(double
        } else {

            fprintf(stderr, "\e[1m\e[32m%s: New Joint Command Received:\e[0m\e[3
;
            fprintf(stderr,             "%s:\t t_slew = %f\n", __FUNCTION__, ARM-
            fflush(stderr);

            for ( k=0; k < (unsigned int)ARM->nLinks; k++ ) {
                ARM->ang_des[k]       = ARM->theta_des[k];
                ARM->ang_cmd_delta[k] = ARM->theta_des[k] - ARM->ang[k][0];
                if ( fabs(ARM->ang_cmd_delta[k]) < 1e-5 ) { /* step size too smal
```

```
                    ARM->rate_cmd[k]    = 0;
                } else {
                    ARM->rate_cmd[k]    = ARM->ang_cmd_delta[k] / ARM->t_slew[0];
                }
                ARM->ang_cmd[k]         = ARM->ang[k][0] + ARM->rate_cmd[k] * ARM->
                DATA->ang_cmd_last[k]  = ARM->ang_cmd[k];

                fprintf(stderr, "%s:\t Joint %d : ( curr, des, rate ) : ( %2.8f,
                        __FUNCTION__, k+1, ARM->ang[k][0], ARM->ang_des[k], ARM->
                fflush(stderr);
            }

            ARM->new_theta_des[0] = 0; /* lower the flag to prevent reuse within

        }

    } else { /* new_theta_des[0] if's else statement */

        for ( k=0; k < (unsigned int)ARM->nLinks; k++ ) {

            /* increment the desired position */
            /*ARM->ang_cmd[k] = ARM->ang[k][0] + ARM->rate_cmd[k] * ARM->period[
            ARM->ang_cmd[k] = DATA->ang_cmd_last[k] + ARM->rate_cmd[k] * ARM->pe

            /* don't allow overshoot in the command */
            if  ( ARM->rate_cmd[k] > 0.0 ) {
                if (   ARM->ang_cmd[k] >= ARM->ang_des[k]   ) {
                    ARM->ang_cmd[k]   = ARM->ang_des[k];
                    /* fprintf(stderr, "\e[1m\e[33m%s: joint %u position reached.\
CTION__, k);*/
                }
            } else {
                if (   ARM->rate_cmd[k] < 0.0 ) {
                    if ( ARM->ang_cmd[k] <= ARM->ang_des[k] )  {
                        ARM->ang_cmd[k]   = ARM->ang_des[k];
                        /*fprintf(stderr, "\e[1m\e[33m%s: joint %u position reached
UNCTION__, k);*/
                    }
                } else {
                    /* rate must be zero */
                    ARM->ang_cmd[k] = ARM->ang_des[k];
                    ARM->rate_cmd[k]     = 0.0;
                }
            }
            DATA->ang_cmd_last[k] = ARM->ang_cmd[k];

        } /* end for-k */

    }/* end new_theta_des if-else-statement */

    return(0);
}
```

B.5.3 armAngRateMode_exit

```
int armAngRateMode_exit(armCtrl_s *ARM) {
    if ( !ARM ) return (-1);
    return(0);
}
```

B.5.4 armAngRateMode_init

```
int armAngRateMode_init(armCtrl_s *ARM){
    if ( !ARM ) return (-1);
    fprintf(stderr,"%s(): sim time = %f\n\n", __FUNCTION__, fspGetSimTime() );

   ARM->manip_mode[0] = (double)ARM->angRateData.manip_mode;
    return(0);
}
```

B.5.5 armCartMode_exec

Listing B.20: *armCartMode_execarmCtrl_sARM* {

```
int armCartMode_exec(armCtrl_s *ARM) {
    if ( !ARM ) return (-1);

    cart_s *DATA = &ARM->cartData;
    double R0T[3][3];
    unsigned int k;
    double   ang[DOF];
    int      doLatch = 0;
    /*************************************************************************
    
    /* check for new commands */
    if ( DATA->cmd_flag[0] > 0 ) {
        fprintf(stderr, "\e[1m\e[32m%s: New Cartesian Command Received:\e[0m\e[
);
        fprintf(stderr,              "%s:\t t_slew = %f\n", __FUNCTION__, ARM->t_
        fflush(stderr);

        DATA->finalp0T[0]  = DATA->cmd_p0T[0];
        DATA->finalp0T[1]  = DATA->cmd_p0T[1];
        DATA->finalp0T[2]  = DATA->cmd_p0T[2];
        DATA->finalRPY[0]  = DATA->cmd_RPY[0];
        DATA->finalRPY[1]  = DATA->cmd_RPY[1];
        DATA->finalRPY[2]  = DATA->cmd_RPY[2];
        DATA->finalSEW     = DATA->cmd_SEW[0];
        DATA->t_slew       = ARM->t_slew[0];
        /* scale the position if it's reaching beyond limits */
        if ( Norm_rt( DATA->finalp0T ) > DATA->maxReach[0] ) {
            double scale = DATA->maxReach[0] / Norm_rt( DATA->finalp0T );
            DATA->finalp0T[0] *= scale;
            DATA->finalp0T[1] *= scale;
            DATA->finalp0T[2] *= scale;
        }

        RPYToRot( DATA->finalRPY, R0T );
        RotToQuat( DATA->q_final, R0T );

        /* lower the flag until the next event raises it */
        DATA->cmd_flag[0] = 0.0;
        if ( DATA->t_slew > 0 ) {
            DATA->configured  = 1;
            DATA->t_slew_start = fspGetSimTime();
            DATA->startp0T[0] = ARM->p0T[0];
            DATA->startp0T[1] = ARM->p0T[1];
            DATA->startp0T[2] = ARM->p0T[2];
            DATA->q_start[0]  = ARM->q_mb2tt[0];
            DATA->q_start[1]  = ARM->q_mb2tt[1];
            DATA->q_start[2]   = ARM->q_mb2tt[2];
            DATA->q_start[3]   = ARM->q_mb2tt[3];
            DATA->startSEW     = ARM->sew_deg[0] * deg2rad;
            DATA->latched      = 0;
        } else {
            DATA->configured = 0;
        }
    } /* end new-command-if */

    /**************************************************************************

    for ( k=0; k<ARM->nLinks; ++k) {
        ang[k] = ARM->ang[k][0];
    }

    /* maintain current position if not configured */
    if ( DATA->configured == 0 ) {
        memcpy( ARM->ang_cmd, ang, ARM->nLinks*sizeof(double));
        memset( ARM->rate_cmd, 0.0, ARM->nLinks*sizeof(double));
        return(0);
    }

    /**********************************************************************

    /* maintain LATCHED commanded joint position if not configured */
    if ( DATA->latched ) {
        memcpy( ARM->ang_cmd, DATA->latchJoints, ARM->nLinks*sizeof(double));
        memset( ARM->rate_cmd, 0.0, ARM->nLinks*sizeof(double));
        return(0);
    }

    /**********************************************************************

    /* Compute new desired location along the trajectory */
    DATA->frac = (fspGetSimTime() - DATA->t_slew_start) / DATA->t_slew;
    if ( DATA->frac >= 1.0 ) {
        DATA->frac = 1.0;
        DATA->desp0T[0] = DATA->finalp0T[0];
        DATA->desp0T[1] = DATA->finalp0T[1];
        DATA->desp0T[2] = DATA->finalp0T[2];
```

```
      DATA->q_des[0]   = DATA->q_final[0];
      DATA->q_des[1]   = DATA->q_final[1];
      DATA->q_des[2]   = DATA->q_final[2];
      DATA->q_des[3]   = DATA->q_final[3];
      DATA->desSEW     = DATA->finalSEW;
      /* set doLatch here so we don't under or over shoot the cartesian pose
of invkin */
      doLatch          = 1;
  } else {

      /* difference in position desired from current */
      DATA->desp0T[0] = ( DATA->finalp0T[0] - DATA->startp0T[0])*DATA->frac +
      DATA->desp0T[1] = ( DATA->finalp0T[1] - DATA->startp0T[1])*DATA->frac +
      DATA->desp0T[2] = ( DATA->finalp0T[2] - DATA->startp0T[2])*DATA->frac +
      /* quaternion interpolate attitude, determine delta from current */
      Qslerp_rt( DATA->q_des, DATA->q_start, DATA->q_final, DATA->frac );
      /* don't forget the SEW... */
      DATA->desSEW = ( DATA->finalSEW - DATA->startSEW)*DATA->frac + DATA->sta

  }
```

```

    DATA->p0T_delta[0] = DATA->desp0T[0] - ARM->p0T[0];
    DATA->p0T_delta[1] = DATA->desp0T[1] - ARM->p0T[1];
    DATA->p0T_delta[2] = DATA->desp0T[2] - ARM->p0T[2];

    Qmult_rt( DATA->q_delta, DATA->q_des, 1, ARM->q_mb2tt, -1 );
    Quat2Rot( R0T, DATA->q_delta, 0 /*don't invert*/ );

    DATA->deltaSEW =  DATA->desSEW - (ARM->sew_deg[0]*deg2rad);

    /* InvKin to desired joint angles for desired point along slew */
    invKin( /* in */      DATA->p0T_delta, R0T, DATA->deltaSEW, &(ARM->armParam
            /* in/out */ ang,
            /* out */    DATA->deltaAng, &DATA->manip_idx, &DATA->nullspace_id
lar, DATA->sigma );

    memcpy( ARM->ang_cmd, ang, ARM->nLinks * sizeof(double));

    for (k=0; k<ARM->nLinks; ++k) {
        ARM->rate_cmd[k] = DATA->deltaAng[k] / ARM->period[0];
    }

    if ( doLatch ) {
        memcpy( DATA->latchJoints, ARM->ang_cmd, ARM->nLinks*sizeof(double));
        DATA->latched = 1;
    }

    return(0);
}
```

B.5.6 armCartMode_exit

```
int armCartMode_exit(armCtrl_s *ARM) {
    if ( !ARM ) return (-1);
    return(0);
}
```

B.5.7 armCartMode_init

Listing B.21: *armCartMode_initarmCtrl_sARM* {

```
int armCartMode_init(armCtrl_s *ARM) {
    if ( !ARM ) return (-1);
    fprintf(stderr,"%s(): sim time = %f\n\n", __FUNCTION__, fspGetSimTime() );
   ARM->manip_mode[0] = (double)ARM->cartData.manip_mode;

   memcpy( ARM->cartData.finalp0T, ARM->p0T,        3*sizeof(double));
   memcpy( ARM->cartData.finalRPY, ARM->rpy0T_deg, 3*sizeof(double));
   ARM->cartData.finalRPY[0] *= deg2rad;
   ARM->cartData.finalRPY[1] *= deg2rad;
   ARM->cartData.finalRPY[2] *= deg2rad;
   memcpy( ARM->cartData.q_final, ARM->q_mb2tt, 4*sizeof(double));
   ARM->cartData.finalSEW = ARM->sew_deg[0] * deg2rad;

   memcpy( ARM->cartData.desp0T, ARM->p0T, 3*sizeof(double));
   memcpy( ARM->cartData.desRPY, ARM->rpy0T_deg, 3*sizeof(double));
   ARM->cartData.desRPY[0] *= deg2rad;
   ARM->cartData.desRPY[1] *= deg2rad;
   ARM->cartData.desRPY[2] *= deg2rad;
   memcpy( ARM->cartData.q_des, ARM->q_mb2tt, 4*sizeof(double));
```

```
    ARM->cartData.desSEW = ARM->sew_deg[0] *deg2rad;

    ARM->cartData.t_slew = 0;

    memset( ARM->cartData.delta, 0.0, DOF*sizeof(double));

    memset( ARM->cartData.deltaAng, 0.0, DOF*sizeof(double));

    ARM->cartData.configured = 0; /* low until good command received */

    return(0);
}
```

B.5.8 armCtrl

```
System *armCtrl( domain_t *domain )
{
    System *sys = fspNewSystem(domain, "ARMCTRL", "Robot Arm Trajectory Contro
l_s) );
```

```

    sys->add  = NULL;
    sys->exit = NULL;
    sys->rt   = NULL;
    sys->log  = NULL;
    sys->drt  = &armCtrl_drt;
    sys->init = &armCtrl_init;

    return( sys );
}
```

B.5.9 armCtrl_drt

Listing B.22: *armCtrl_drtSystemsys*

```
static int armCtrl_drt( System *sys )
{
    armCtrl_s      *armCtrl = (armCtrl_s*)sys->data;
    unsigned int   k;
    static double ang_cmd_last[DOF];
    armParam_s      *PARAMS=&(armCtrl->armParams);
    double J0t[3][DOF], J0r[3][DOF], R0T[3][3], R0Tnext[3][3];
    double p0s[3], p0e[3], p0w[3], pvec[3];
    double ang[DOF]; /* copy of radian joint angles to be in form suitable for

    if ( armCtrl->armLocked > 0 ) { return(FSP_NO_ERROR); }

    for ( k=0; k<DOF; ++k ) {
        ang[k] = armCtrl->ang[k][0];

        ang_cmd_last[k] = armCtrl->ang_cmd[k];
    }

    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    /* compute current forward kinematics */
    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    fwdKin( /* --- inputs  --- */ DOF, DOF, ang, PARAMS->a, PARAMS->d, PARAMS-
pha, PARAMS->pNT, PARAMS->RNT,
            /* --- outputs --- */ armCtrl->p0T, R0T, &(J0t[0][0]), &(J0r[0][0]
    RotToRPY( /* input */  R0T,
```

```
                /* output */ &(armCtrl->singular), armCtrl->rpy0T_deg );
    armCtrl->rpy0T_deg[0] *= rad2deg;
    armCtrl->rpy0T_deg[1] *= rad2deg;
    armCtrl->rpy0T_deg[2] *= rad2deg;
    RotToQuat( armCtrl->q_mb2tt, R0T );
    Qmult_rt( armCtrl->q_sb2tt, armCtrl->q_mb2tt, 1, armCtrl->q_sb2mb, 1 );

    /* Calculate SEW angle */
    fwdKin(JOINT_SHOULDER, DOF, ang, PARAMS->a, PARAMS->d, PARAMS->calpha, PAR
>pNT, PARAMS->RNT,
              p0s, R0Tnext, &J0t[0][0], &J0r[0][0]);

    fwdKin(JOINT_ELBOW, DOF, ang, PARAMS->a, PARAMS->d, PARAMS->calpha, PARAMS
T, PARAMS->RNT,
              p0e, R0Tnext, &J0t[0][0], &J0r[0][0]);

    fwdKin(JOINT_WRIST, DOF, ang, PARAMS->a, PARAMS->d, PARAMS->calpha, PARAMS
T, PARAMS->RNT,
              p0w, R0Tnext, &J0t[0][0], &J0r[0][0]);

    sewfwdKin(p0s, p0e, p0w, PARAMS->vhat, pvec, armCtrl->sew_deg );
    armCtrl->sew_deg[0] *= rad2deg;

    /* compute the end-effector to servicer base rotation from Engine truth */
    Qmult_rt( armCtrl->q_beng2ee, armCtrl->q_i2ee, 1, armCtrl->q_i2base, -1 );

    double r_base2ee_i[3];
    Subt_rt( r_base2ee_i, armCtrl->r_i2ee_i, armCtrl->r_i2base_i );
    Qtrans_rt( armCtrl->r_sb2ee_sb, armCtrl->q_i2base, 1, r_base2ee_i );


    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    /* -------------------- **
    **  mode transitions  **
    ** -------------------- */
    if ( ((int)armCtrl->newMode[0] != armCtrl->currMode )
          && (armCtrl->newMode[0] > ARMCTRL_UNDEFINED && armCtrl->newMode[0] <

       if ( armCtrl->currMode > ARMCTRL_UNDEFINED ) armCtrl->modes[armCtrl->cu
```

```
);
        armCtrl->modes[(int)armCtrl->newMode[0]].init(armCtrl);
        armCtrl->currMode = (int)armCtrl->newMode[0];
        armCtrl->t_mode_start = fspGetSimTime();
    }
    /* ------------------------------------------ **
    ** increment the time                        **
    ** ------------------------------------------ */
    armCtrl->t_in_mode = fspGetSimTime() - armCtrl->t_mode_start;

    /* ------------------------------------------ **
    ** execute the current control mode          **
    ** ------------------------------------------ */
    armCtrl->modes[armCtrl->currMode].exec(armCtrl);

    for ( k=0; k<armCtrl->nLinks; k++ ) {
        armCtrl->ang_err[k] = armCtrl->ang_des[k] - armCtrl->ang[k][0];
    }

#if ARMCTRL_VSD_VERBOSE
```

```
    fprintf(stderr,"%s: mode = %d\n", __FUNCTION__, (int)armCtrl->currMode );
    fflush(stderr);
#endif


    for (k=0; k<DOF; ++k) {
    armCtrl->ang_cmd_delta[k] = armCtrl->ang_cmd[k] - ang_cmd_last[k];
}
```

B.5.10 armCtrl_init

Listing B.23: *armCtrl_initSystemsys*

```
static int armCtrl_init( System *sys )
{
    armCtrl_s      *armCtrl = (armCtrl_s*)sys->data;
    int            tmpInt;
    unsigned int i;
    char           str[256];
    int            enFts, n;
    int r, c, j;
    double *initMode;
    char     *DOF_LAB[] = {”FX”, ”FY”, ”FZ”, ”MX”, ”MY”, ”MZ”};
#if 0
    fft_out_s *PROPFFT = armCtrl->prop_fft;
    fft_out_s *FTSFFT  = armCtrl->fts_fft;
    int ftsMaxPeaks, propMaxPeaks;
#endif
    fft_peaks_s *PROP_PEAK = armCtrl->prop_peaks;
    fft_peaks_s *FTS_PEAK  = armCtrl->fts_peaks;

    /* parameters from workspace */
    FSPgetParamVec( armCtrl->period,  1, ”period”, ”drt period, seconds”);
    fspSetDRTsampleTime( sys, armCtrl->period[0]);

    FSPgetParamInt( enFts,          ”MANIP.CONFIG.en_fts”,          ”flag, nonzero
ue sensor”);
    FSPgetParamInt( armCtrl->armLocked, ”MANIP.locked”, ”nonzero if whole arm

    FSPgetParamVec( initMode,     1, ”initMode”,    ”initial control mode”);
    FSPgetParamVec( armCtrl->newMode,       1, "newMode",     "event-driven new c
    FSPgetParamInt( armCtrl->enCartFilt,      "enable_cart_filter", "flag, nonz
sian space filter");
    FSPgetParamInt( armCtrl->enJointFilt,     "enable_joint_filter", "flag, non
t-space torque filter");
    FSPgetParamVec( armCtrl->r_scm2mb_sb, 3, "MANIP.r_scm2mb_sb",
   "position of the manipulator b
ase (mb) wrt servicer center of mass (scm) in servicer body frame");
    FSPgetParamVec( armCtrl->q_sb2mb,      4, "MANIP.q_sb2mb",
   " quaternion, scalar last, ser
vicer body frame to manipulator base frame");
    FSPgetParamVec( armCtrl->r_sb2ttd_sb, 3, "r_sb2ttd_sb",
   "position of desired tool tip
wrt servicer body in servicer body frame");

    FSPgetParamInt( tmpInt, "MANIP.n_links", "number of actuated joints in the
    armCtrl->nLinks = (unsigned int)tmpInt;
    i = fspJointFindId( "manip_joint*", &(armCtrl->jid));
    if ( armCtrl->nLinks > i ) {
        fprintf(stderr,"%s: ERROR: manip n_links = %d but only found %d joint I
                    __FUNCTION__, armCtrl->nLinks, i );
        return(FSP_PARAM_MISSING);
    }

    n = ( enFts == 0 ? armCtrl->nLinks : (armCtrl->nLinks + 1));

    FSPgetParamVec( armCtrl->ang_init,        (int)armCtrl->nLinks, "MANIP.jnt_a
 angles, rads");
    FSPgetParamVec( armCtrl->cartData.maxReach, 1, "maxReach", "max reach of t
;
    FSPgetParamVec( armCtrl->velData.maxReach,  1, "maxReach", "max reach of t
;


    /* event parameters */
    FSPgetParamVec( armCtrl->new_theta_des, 1,                        "new_theta_d
to indicate new desired joint angles present");
    FSPgetParamVec( armCtrl->theta_des,      (int)armCtrl->nLinks, "theta_des",
 "EVENT new desi
red joint angles, rads");
```

```
   FSPgetParamVec( armCtrl->t_slew,        1,                    "t_slew",
"EVENT time to
reach new desired joint angles, in seconds");
   FSPgetParamVec( armCtrl->cartData.cmd_flag, 1, "new_cmd_cart", "non-zero i
and event");
   FSPgetParamVec( armCtrl->cartData.cmd_p0T, 3, "cmd_p0T_cart", "event-drive
ion wrt shoulder [m]");
   FSPgetParamVec( armCtrl->cartData.cmd_RPY, 3, "cmd_RPY_cart", "event-drive
ude wrt shoulder [rads]");
   FSPgetParamVec( armCtrl->cartData.cmd_SEW, 1, "cmd_SEW_cart", "event-drive
[rads]");
   /* reactive velocity command data */
   FSPgetParamVec( armCtrl->velData.cmd_flag, 1, "new_cmd_vel", "non-zero if
d event");
   FSPgetParamVec( armCtrl->velData.cmd_p0T, 3, "cmd_p0T_vel", "event-driven
n wrt shoulder [m]");
   FSPgetParamVec( armCtrl->velData.cmd_RPY, 3, "cmd_RPY_vel", "event-driven
e wrt shoulder [rads]");
   FSPgetParamVec( armCtrl->velData.cmd_SEW, 1, "cmd_SEW_vel", "event-driven
ads]");
```

```


    /* manipulator parameters */
    FSPcopyParamMf( armCtrl->link_b ,   n, 3, "MANIP.link_b", "forward vector f
xt link, m");
    FSPcopyParamMf( armCtrl->link_q ,   n, 4, "MANIP.link_q", "forward orientat
 quaternion");
    FSPcopyParamVd( armCtrl->mass,      n,    "MANIP.mass",   "link mass, kg");
    FSPcopyParamMf( armCtrl->r_l2cm_l , n, 3, "MANIP.r_l2cm_l", "position of li
 link frame, m");
#if 0
    FSPcopyParamMf( armCtrl->Icm,       (int)armCtrl->nLinks, 6, "MANIP.I_lcm_l
 "link moment of
inertia at CoM, Ixx, Iyy, Izz, Ixy, Ixz, Iyz");
#endif
    for ( i = 0; i < armCtrl->nLinks; i++ ) {
        snprintf( str, 255, "MANIP.linkName%u", i+1 );
        FSPgetParamStr( armCtrl->linkName[i], str, "Link i description");
    }
    FSPgetParamStr( armCtrl->eeBodyName, "MANIP.eeBodyName", "end effector bod

    if ( enFts ) {
        /* combine links 7 & 8 parameters because the FTS "joint" is integral t
        /* mass */
        armCtrl->mass[n-2] = armCtrl->mass[n-2] + armCtrl->mass[n-1];
        /* CM location - assumes identity transform from pre-FTS to post-FTS bo
        /* r_l2cmcombined = ( r_l2cm1*m1 + r_l2cm2*m2 ) / (m1 + m2)
 */
        fprintf(stderr,"%s: PRE-COMBINE:\n",__FUNCTION__);
        r = n-2;
        fprintf(stderr,"                    r_l2cm_l[%d] = [ %e %e %e ] mass[%d] = %e
                    r, armCtrl->r_l2cm_l[r][0], armCtrl->r_l2cm_l[r][1], armCtrl->r
mCtrl->mass[r] );
        r = n-1;
        fprintf(stderr,"                    r_l2cm_l[%d] = [ %e %e %e ] mass[%d] = %e
                    r, armCtrl->r_l2cm_l[r][0], armCtrl->r_l2cm_l[r][1], armCtrl->r
mCtrl->mass[r] );
        r = n-1;
        armCtrl->r_l2cm_l[r-1][0] = armCtrl->r_l2cm_l[r-1][0] * armCtrl->mass[r
m_l[r][0] * armCtrl->mass[n-1];
```

```
        armCtrl->r_l2cm_l[r-1][1] = armCtrl->r_l2cm_l[r-1][1] * armCtrl->mass[r
m_l[r][1] * armCtrl->mass[n-1];
        armCtrl->r_l2cm_l[r-1][2] = armCtrl->r_l2cm_l[r-1][2] * armCtrl->mass[r
m_l[r][2] * armCtrl->mass[n-1];
        armCtrl->r_l2cm_l[r-1][0] = armCtrl->r_l2cm_l[r-1][0] / ( armCtrl->mass
s[n-1] );
        armCtrl->r_l2cm_l[r-1][1] = armCtrl->r_l2cm_l[r-1][1] / ( armCtrl->mass
s[n-1] );
        armCtrl->r_l2cm_l[r-1][2] = armCtrl->r_l2cm_l[r-1][2] / ( armCtrl->mass
s[n-1] );
        fprintf(stderr,"%s: POST-COMBINE:\n",__FUNCTION__);
        r = n-2;;
        fprintf(stderr,"                   r_l2cm_l[%d] = [ %e %e %e ] mass[%d] = %e
                r, armCtrl->r_l2cm_l[r][0], armCtrl->r_l2cm_l[r][1], armCtrl->r
mCtrl->mass[r] );
    }
    /* parameters for FFTs from other modules */
    FSPgetParamInt( armCtrl->prop_nPeaks, "PROP.FFT.maxNpeaks", "max # peaks t

    FSPgetParamInt( armCtrl->fts_nPeaks,  "MANIP.FTS.FFT.maxNpeaks", "max # pe
FT");


    for (i=0; i<6; ++i ) {
        fspSetVarExpandTags(sys, 1, DOF_LAB[i]);
#if 0
        /* ACS/PROP actuation */
        FSPgetParamInt( PROPFFT[i].nBins, "PROP.FFT.nBins", "PROP module FFT nu
        FSPgetParamVec( PROPFFT[i].Fs,    1, "PROP.FFT.Fs",   "PROP module FFT sa
        FSPgetParamVec( PROPFFT[i].df,    1, "PROP.FFT.df",   "PROP module FFT de
]");
        /* FTS subsystem */
        FSPgetParamInt( FTSFFT[i].nBins,   "MANIP.FTS.FFT.nBins", "MANIP module
        FSPgetParamVec( FTSFFT[i].Fs,    1, "MANIP.FTS.FFT.Fs",   "MANIP module F
]");
        FSPgetParamVec( FTSFFT[i].df,    1, "MANIP.FTS.FFT.df",   "MANIP module F
n [Hz]");
#endif
    }


    armCtrl->driveAxis = calloc( armCtrl->nLinks, sizeof(int));

    /* inputs from other modules */
    for ( i = 0; i < armCtrl->nLinks; i++ ) {
        fspJointInfo_s jInfo;

        fspJointGetInfo( armCtrl->jid[i], &jInfo );
        fspSetVarExpandTags( sys, 1, jInfo.name );

        fprintf(stderr,"%s: armCtrl->nLinks = %d, i = %d\n", __FUNCTION__, armC
        fprintf(stderr,"%s: joint %d: info.name = '%s' info.axis = ( %f %f %f )
,
                    jInfo.name, jInfo.axis_j[0], jInfo.axis_j[1], jInfo.axis_j[2] )
        fprintf(stderr,"%s: joint %d: armCtrl->jid = %d; jInfo.id = %d\n", __FU
->jid[i], jInfo.id );

        if ( jInfo.axis_j[0] > 0.0 ) {
            armCtrl->driveAxis[i] = 0;
        } else {
            if ( jInfo.axis_j[1] > 0.0 ) {
                armCtrl->driveAxis[i] = 1;
            } else {
                if ( jInfo.axis_j[2] > 0.0 ) {
                    armCtrl->driveAxis[i] = 2;
                } else {
                    fprintf(stderr,"%s: WARNING joint %d axis_j = [ %1.1f %1.1f %1
!\n",
                              __FUNCTION__, i, jInfo.axis_j[0], jInfo.axis_j[1], jIn
                }
            }
        }
        if ( armCtrl->armLocked == 0 ) {
            snprintf(str, 256, "current joint angle (inner to outer) for manip j
, i, jInfo.name );
            FSPgetInputVec( armCtrl->ang[i], 1, "JOINTS.%.angle", str );

            snprintf(str, 256, "current joint rates for manip joint %d '%s', [ra
);
            FSPgetInputVec( armCtrl->rate[i], 3, "JOINTS.%.dw", str );
        }

        /*debug*/
        fprintf(stderr,"%s: armCtrl->ang[%d]  = %p\n", __FUNCTION__, i, armCtrl
        fprintf(stderr,"%s: armCtrl->rate[%d] = %p\n", __FUNCTION__, i, armCtrl
        fflush(stderr);
    }

    FSPgetInputVec( armCtrl->acsM, 3, "PROP.torq_b", "Applied ACS torque to se
s");
    FSPgetInputVec( armCtrl->acsF, 3, "PROP.force_b", "applied ACS force [N]")
    FSPgetInputVec( armCtrl->q_i2base,     4, "MPROP.base.q_i2b",
"inertial orientation of the
servicer base body");
    FSPgetInputVec( armCtrl->r_i2base_i, 3, "MPROP.base.Ro_i", "inertial posit
");

    fspSetVarExpandTags( sys, 1, armCtrl->eeBodyName );
    FSPgetInputVec( armCtrl->q_i2ee,       4, "MPROP.%.q_i2b",
"inertial orientation of the
```

```
end effector body");
    FSPgetInputVec( armCtrl->r_i2ee_i , 3, "MPROP.%.Ro_i", "inertial position o
n");

    fspSetVarExpandTags( sys, 1, armCtrl->linkName[2] );
    FSPgetInputVec( armCtrl->q_i2link3 ,    4, "MPROP.%.q_i2b",
"inertial orientation of link
 3 (for jacobian calcs)");

    /* inputs from other module FFTs */
    for (i=0; i<6; ++i ) {
        fspSetVarExpandTags(sys, 1, DOF_LAB[i]);
#if 0
        /* n = PROPFFT[i].nBins; */
        /* FSPgetInputVec( PROPFFT[i].freqBins, n, "PROP.%.freqBins", "PROP moc
[Hz]"); */
        /* FSPgetInputVec( PROPFFT[i].fftMag,    n, "PROP.%.fftMag",
"PROP module FFT magnitudes [
-]"); */
        /* FSPgetInputVec( PROPFFT[i].fftPhase, n, "PROP.%.fftPhase", "PROP moc
");  */
        FSPgetInputVec( PROP_FFT[i].newMeas,   1, "PROP.%.newMeas",   "PROP modul
t flag [-]");
        FSPgetInputVec( PROP_FFT[i].peakFreqs,propMaxPeaks,"PROP.%.peakFreqs","
 [Hz]");
        FSPgetInputVec( PROP_FFT[i].peakAmps,  propMaxPeaks,"PROP.%.peakAmps", "
s of peaks [-]");
        FSPgetInputVec( PROP_FFT[i].peakPhases,propMaxPeaks,"PROP.%.peakPhases"
ad]");
        /* n = FTSFFT[i].nBins; */
        /* FSPgetInputVec( FTSFFT[i].freqBins, n, "FTS.%.freqBins", "FTS module
]"); */
        /* FSPgetInputVec( FTSFFT[i].fftMag,    n, "FTS.%.fftMag",    "FTS module
); */
        /* FSPgetInputVec( FTSFFT[i].fftPhase, n, "FTS.%.fftPhase", "FTS module
  */
        FSPgetInputVec( FTSFFT[i].newMeas,   1, "FTS.%.newMeas",   "FTS module FF
ag [-]");
        FSPgetInputVec( FTSFFT[i].peakFreqs,ftsMaxPeaks,"FTS.%.peakFreqs","Freq
]");
```

```
        FSPgetInputVec( FTSFFT[i].peakAmps, ftsMaxPeaks,"FTS.%.peakAmps", "Norr
 peaks [-]");
        FSPgetInputVec( FTSFFT[i].peakPhases,ftsMaxPeaks,"FTS.%.peakPhases","Ph
);
#endif
        /* PROP actuation */
        /* PROP_PEAK[i].maxPeaks = armCtrl->prop_nPeaks; */
        n = armCtrl->prop_nPeaks; /* PROP_PEAK[i].maxPeaks; */
        FSPgetInputVec( PROP_PEAK[i].nPeaks, 1, "PROP.%.nPeaks", "# peaks found
T");
        /* FSPgetInputVec( PROP_PEAK[i].new,     1, "PROP.%.newMeas", "nonzero i
usrement"); */
        FSPgetInputVec( PROP_PEAK[i].amps,    n, "PROP.%.peakAmps", "amplitude
        FSPgetInputVec( PROP_PEAK[i].freqs,   n, "PROP.%.peakFreqs", "frequency
        FSPgetInputVec( PROP_PEAK[i].phases,  n, "PROP.%.peakPhases", "Phase [r
        /* FTS subsystem */
        /* FTS_PEAK[i].maxPeaks = armCtrl->fts_nPeaks; */
        /* n = FTS_PEAK[i].maxPeaks; */
        n = armCtrl->fts_nPeaks;
        FSPgetInputVec( FTS_PEAK[i].nPeaks, 1, "FTS.%.nPeaks", "# peaks found i
);
      /* FSPgetInputVec( FTS_PEAK[i].new,    1, "FTS.%.newMeas", "nonzero ind
rement"); */
      FSPgetInputVec( FTS_PEAK[i].amps,    n, "FTS.%.peakAmps", "amplitude of
      FSPgetInputVec( FTS_PEAK[i].freqs,   n, "FTS.%.peakFreqs", "frequency [
      FSPgetInputVec( FTS_PEAK[i].phases,  n, "FTS.%.peakPhases", "Phase [rad
   }

   /* outputs */
   FSPnewOutput( armCtrl->ang_des,       armCtrl->nLinks, 1, "ang_des",
   FSP_LOG1x, "desired
 final joint angles, rads");
   FSPnewOutput( armCtrl->ang_err,       armCtrl->nLinks, 1, "ang_err",
   FSP_LOG1x, "error b
etween desired and actual joint angles, rads");
   FSPnewOutput( armCtrl->ang_cmd_delta, armCtrl->nLinks, 1, "ang_cmd_delta",
etween cmd angle and curent when received, rads");
   FSPnewOutput( armCtrl->ang_cmd,       armCtrl->nLinks, 1, "ang_cmd",
   FSP_LOG1x, "command
ed joint angles, rads");
```

```
    FSPnewOutput( armCtrl->rate_cmd,      armCtrl->nLinks, 1, "rate_cmd",
FSP_LOG1x, "command
ed joint rates, rad/sec");
    FSPnewOutput( armCtrl->torq_cmd,      armCtrl->nLinks, 1, "torq_cmd",
FSP_LOG1x, "command
ed joint torques, Nm");
    FSPnewOutput( armCtrl->manip_mode,    1,               1, "manip_mode",
FSP_LOG1x, "comman
ded manipulator joint controller mode");
    fspTelemetry( &armCtrl->currMode,     FSP_INT,         1, "mode",
FSP_LOG1x, "current
 armCtrl internal mode");
    FSPnewOutput( armCtrl->acsF_tt_sb,    6,               1, "acsF_tt_sb",
FSP_LOG1x, "ACS tor
que induced forces and torques at tool tip measured in the servicer body fram
    FSPnewOutput( armCtrl->acsF_tt_tt,    6,               1, "acsF_tt_tt",
FSP_LOG1x, "ACS tor
que induced forces and torques at tool tip measured in the tool tip frame");
#if 0
    FSPnewOutput( armCtrl->r_mb2tt_sb,    3,               1, "r_mb2tt_sb",
FSP_LOG1x, " position of tool tip wrt manip base in servicer body frame");
    FSPnewOutput( armCtrl->r_mb2tt_mb,    3,                1, "r_mb2tt_mb", FSP_LOG1x, "position of tool tip wrt manip base in manip base frame");
    FSPnewOutput( armCtrl->r_scm2tt_sb,   3,                1, "r_scm2tt_sb", FSP_LOG1x, "position of tool tip wrt servicer center of mass in servicer body frame");
#endif
    FSPnewOutput( armCtrl->r_sb2ee_sb,    3,                1, "r_sb2ee_sb", FSP_LOG1x, "position of end effector wrt servicer base in servicer body frame");
    FSPnewOutput( armCtrl->q_mb2tt,       4,                1, "q_mb2tt", FSP_LOG1x, "quaternion, scalar last, rotation from manipulator base to tool tip frame");
    FSPnewOutput( armCtrl->q_sb2tt,       4,                1, "q_sb2tt", FSP_LOG1x, "quaternion, scalar last, rotation from servicer body to tool tip frame");
    FSPnewOutput( armCtrl->q_beng2ee,     4,                1, "q_beng2ee", FSP_LOG1x, "quatern
ion from servicer body to end effector using engine−truth, not kinematic libr
    if ( armCtrl−>armLocked == 0 ) {
        /* LOCKED joints don't have joint angles, so can't pull for kinematics
        FSPnewOutput( armCtrl−>p0T,           3, 1,               "p0T",
    FSP_LOG1x, "curr
ent tool position in the arm base frame [m]");
        FSPnewOutput( armCtrl−>rpy0T_deg,     3, 1,               "rpy0T_deg",
    FSP_LOG1x, "curr
ent roll−pitch−yaw of tool in arm base [deg]");
        FSPnewOutput( armCtrl−>sew_deg,       1, 1,               "SEW_deg",
    FSP_LOG1x, "curr
ent Shoulder−Elbow−Wrist (SEW) angle [deg]");
        fspTelemetry( &(armCtrl−>singular),  FSP_INT,  1,        "singular",
    FSP_LOG1x, "nonz
ero if arm kinematics are singular [−]");
    }
    fspTelemetry( &armCtrl−>t_in_mode,       FSP_DOUBLE, 1, "t_in_mode",
    FSP_LOG1x, "elap
sed time in current mode [sec]");
    fspTelemetry( &armCtrl−>cartData.desp0T, FSP_DOUBLE, 3, "CART.desp0T",
FSP_LOG1x, "inst
antaneous position of tool throughout slew");
    fspTelemetry( &armCtrl->cartData.desRPY, FSP_DOUBLE, 3, "CART.desRPY",
FSP_LOG1x, "[rad
]");
    fspTelemetry( &armCtrl->cartData.desSEW, FSP_DOUBLE, 1, "CART.desSEW",
FSP_LOG1x, "[rad
]");
    fspTelemetry( &armCtrl->cartData.q_des,  FSP_DOUBLE, 4, "CART.des_q",
FSP_LOG1x, "[-]"
);
    fspTelemetry( &armCtrl->cartData.frac,   FSP_DOUBLE, 1, "CART.frac",
FSP_LOG1x, "frac
tion through the slew");
    fspTelemetry( &armCtrl->cartData.finalp0T, FSP_DOUBLE, 3, "CART.finalp0T",
FSP_LOG1x, "
instantaneous position of tool throughout slew");
    fspTelemetry( &armCtrl->cartData.finalRPY, FSP_DOUBLE, 3, "CART.finalRPY",
FSP_LOG1x, "
[rad]");
```

```
    fspTelemetry( &armCtrl->cartData.finalSEW, FSP_DOUBLE, 1, "CART.finalSEW",
FSP_LOG1x, "
[rad]");
    fspTelemetry( &armCtrl->cartData.q_final,  FSP_DOUBLE, 4, "CART.final_q",
FSP_LOG1x, "
[-]");
    fspTelemetry( &armCtrl->cartData.manip_idx,FSP_DOUBLE, 1, "CART.manip_idx"
FSP_LOG1x,""
);
    fspTelemetry( &armCtrl->cartData.nullspace_idx,FSP_DOUBLE,1,"CART.nullspac
;
    fspTelemetry( &armCtrl->cartData.Jpsissingular,FSP_INT,1,"CART.Jpsissingul
    fspTelemetry( &armCtrl->cartData.sigma,     FSP_DOUBLE, DOF, "CART.sigma",
FSP_LOG1x,"")
;
    fspTelemetry( &armCtrl->cartData.p0T_delta, FSP_DOUBLE, 3, "CART.deltap0T"
FSP_LOG1x,
"instantaneous position of tool throughout slew");
    fspTelemetry( &armCtrl->cartData.deltaSEW, FSP_DOUBLE, 1, "CART.deltaSEW",
FSP_LOG1x, "
[rad]");
    fspTelemetry( &armCtrl->cartData.q_delta,  FSP_DOUBLE, 4, "CART.delta_q",
FSP_LOG1x, "
[-]");
    fspTelemetry( &armCtrl->cartData.deltaAng, FSP_DOUBLE,DOF,"CART.deltaTheta
FSP_LOG1x, "[r
ad]");
    fspTelemetry( &armCtrl->cartData.latched,  FSP_INT,    1, "CART.latched",
FSP_LOG1x, "no
nzero if latched to final cmd angles of invkin slew");
    FSPnewOutput( armCtrl->Fcmd_b, 6, 1, "Fcmd_b", FSP_LOG1x, "(base body) sta
e command for manipulator end effector [N,Nm]");
    FSPnewOutput( armCtrl->Fcmd_3, 6, 1, "Fcmd_3", FSP_LOG1x, "(link 3) stacke
ommand for manipulator end effector [N,Nm]");
    FSPnewOutput( armCtrl->q_base2link3, 4, 1, "q_base2link3", FSP_LOG1x, "rel
nip link 3");

    /* -------------- */ FSP_assemble_IO_finished();/* -------------- */

    /* configure mode structures */
    armCtrl->modes[ARMCTRL_DISABLED].init = &armDisabledMode_init;
    armCtrl->modes[ARMCTRL_DISABLED].exec = &armDisabledMode_exec;
    armCtrl->modes[ARMCTRL_DISABLED].exit = &armDisabledMode_exit;
    armCtrl->modes[ARMCTRL_DISABLED].data = (void*)&(armCtrl->disabledData);
    armCtrl->modes[ARMCTRL_DISABLED].id   = ARMCTRL_DISABLED;
    armCtrl->disabledData.manip_mode = MANIP_MODE_DISABLED;

    armCtrl->modes[ARMCTRL_ANGRATE].init = &armAngRateMode_init;
    armCtrl->modes[ARMCTRL_ANGRATE].exec = &armAngRateMode_exec;
    armCtrl->modes[ARMCTRL_ANGRATE].exit = &armAngRateMode_exit;
    armCtrl->modes[ARMCTRL_ANGRATE].data = (void*)&(armCtrl->angRateData);
    armCtrl->modes[ARMCTRL_ANGRATE].id   = ARMCTRL_ANGRATE;
    armCtrl->angRateData.manip_mode = MANIP_MODE_ANGLE;

    armCtrl->modes[ARMCTRL_TORQ].init = &armTorqMode_init;
    armCtrl->modes[ARMCTRL_TORQ].exec = &armTorqMode_exec;
    armCtrl->modes[ARMCTRL_TORQ].exit = &armTorqMode_exit;
    armCtrl->modes[ARMCTRL_TORQ].data = (void*)&(armCtrl->torqData);
    armCtrl->modes[ARMCTRL_TORQ].id   = ARMCTRL_TORQ;
    armCtrl->torqData.manip_mode = MANIP_MODE_TORQUE;

    armCtrl->modes[ARMCTRL_CART].init = &armCartMode_init;
    armCtrl->modes[ARMCTRL_CART].exec = &armCartMode_exec;
    armCtrl->modes[ARMCTRL_CART].exit = &armCartMode_exit;
    armCtrl->modes[ARMCTRL_CART].data = (void*)&(armCtrl->cartData);
    armCtrl->modes[ARMCTRL_CART].id   = ARMCTRL_CART;
    armCtrl->cartData.manip_mode = MANIP_MODE_ANGLE;

    armCtrl->modes[ARMCTRL_VEL].init = &armVelMode_init;
    armCtrl->modes[ARMCTRL_VEL].exec = &armVelMode_exec;
    armCtrl->modes[ARMCTRL_VEL].exit = &armVelMode_exit;
    armCtrl->modes[ARMCTRL_VEL].data = (void*)&(armCtrl->velData);
    armCtrl->modes[ARMCTRL_VEL].id   = ARMCTRL_VEL;
    armCtrl->velData.manip_mode      = MANIP_MODE_TORQUE;

    armCtrl->modes[ARMCTRL_FDFWD].init = &armFdFwdMode_init;
    armCtrl->modes[ARMCTRL_FDFWD].exec = &armFdFwdMode_exec;
    armCtrl->modes[ARMCTRL_FDFWD].exit = &armFdFwdMode_exit;
    armCtrl->modes[ARMCTRL_FDFWD].data = (void*)&(armCtrl->cartData);
    armCtrl->modes[ARMCTRL_FDFWD].id   = ARMCTRL_FDFWD;
    armCtrl->fdFwdData.manip_mode = MANIP_MODE_TORQUE;

    /* armCtrl->manip_mode[0] = armCtrl->modes[armCtrl->currMode].data->manip_
  */
    if ( initMode[0] > ARMCTRL_UNDEFINED && initMode[0] < ROBOT_NUM_MODES ) {
        /* initialize the mode on the first drt cycle */
        armCtrl->currMode    = ARMCTRL_UNDEFINED;
        armCtrl->newMode[0] = initMode[0];
    } else {
        armCtrl->currMode = ARMCTRL_UNDEFINED;
        armCtrl->newMode[0] = ARMCTRL_DISABLED;
    }

    /* initialize outputs */
    for (i = 0; i < armCtrl->nLinks; i++ ) {
        armCtrl->ang_cmd[i]   = armCtrl->ang_init[i];
        armCtrl->ang_des[i]   = armCtrl->ang_init[i];
        armCtrl->rate_cmd[i] = 0;
    }

```

```c


    /* robot_ik initialization */
    memcpy( armCtrl->armParams.a, robot_a,            (DOF+1)*sizeof(double));
    memcpy( armCtrl->armParams.d, robot_d,            (DOF+1)*sizeof(double));
    memcpy( armCtrl->armParams.alpha, robot_alpha, (DOF+1)*sizeof(double));
    for ( r=0; r<3; ++r){
        for ( c=0;c<3;++c) {
            armCtrl->armParams.RNT[r][c] = RNT[r][c];
        }
        armCtrl->armParams.pNT[r]   = pNT[r];
        armCtrl->armParams.vhat[r] = vhat[r];
    }
    for (j=0;j<DOF; ++j){
        armCtrl->armParams.calpha[j] = cos(armCtrl->armParams.alpha[j]);
        armCtrl->armParams.salpha[j] = sin(armCtrl->armParams.alpha[j]);
    }


    return(FSP_NO_ERROR);
```

```
}
```

B.5.11 armDisabledMode_exec

```
int armDisabledMode_exec(armCtrl_s *ARM) {
    if ( !ARM ) return (-1);
    return(0);
}
```

B.5.12 armDisabledMode_exit

```
int armDisabledMode_exit(armCtrl_s *ARM) {
    if ( !ARM ) return (-1);
    return(0);
}
```

B.5.13 armDisabledMode_init

```
int armDisabledMode_init(armCtrl_s *ARM) {
    if ( !ARM ) return (-1);

    fprintf(stderr,"%s(): sim time = %f\n\n", __FUNCTION__, fspGetSimTime() );

   ARM->manip_mode[0] = (double)ARM->disabledData.manip_mode;
    return(0);
}
```

B.5.14 armVelMode_exec

Listing B.24: *Reactive Velocity Control Execution /*

```
int armVelMode_exec(armCtrl_s *ARM) {
    /* @brief Reactive Velocity Control Execution */
    if ( !ARM ) return (-1);

    int d, k;
    vel_s *DATA = &ARM->velData;
    armParam_s      *PARAMS=&(ARM->armParams);
    double R0T[3][3];

    /* check for new commands */
    if ( DATA->cmd_flag[0] > 0 ) {
        fprintf(stderr, "\e[1m\e[32m%s: New Reaction Velocity Command Received:
NCTION__);
        fprintf(stderr,             "%s:\t  t_slew = %f\n", __FUNCTION__, ARM->t_
        fflush(stderr);

        DATA->finalp0T[0] = DATA->cmd_p0T[0];
        DATA->finalp0T[1] = DATA->cmd_p0T[1];
        DATA->finalp0T[2] = DATA->cmd_p0T[2];
        DATA->finalRPY[0] = DATA->cmd_RPY[0];
        DATA->finalRPY[1] = DATA->cmd_RPY[1];
        DATA->finalRPY[2] = DATA->cmd_RPY[2];
        DATA->finalSEW    = DATA->cmd_SEW[0];
        DATA->t_slew      = ARM->t_slew[0];
        /* scale the position if it's reaching beyond limits */
        if ( Norm_rt( DATA->finalp0T ) > DATA->maxReach[0] ) {
            double scale = DATA->maxReach[0] / Norm_rt( DATA->finalp0T );
            DATA->finalp0T[0] *= scale;
            DATA->finalp0T[1] *= scale;
            DATA->finalp0T[2] *= scale;
        }

        RPYToRot( DATA->finalRPY, R0T );
        RotToQuat( DATA->q_final, R0T );

        /* lower the flag until the next event raises it */
        DATA->cmd_flag[0] = 0.0;
        if ( DATA->t_slew > 0 ) {
            DATA->configured  = 1;
            DATA->t_slew_start = fspGetSimTime();
            DATA->startp0T[0] = ARM->p0T[0];
            DATA->startp0T[1] = ARM->p0T[1];
            DATA->startp0T[2] = ARM->p0T[2];
            DATA->q_start[0]  = ARM->q_mb2tt[0];
            DATA->q_start[1]  = ARM->q_mb2tt[1];
            DATA->q_start[2]  = ARM->q_mb2tt[2];
            DATA->q_start[3]  = ARM->q_mb2tt[3];
            DATA->startSEW     = ARM->sew_deg[0] * deg2rad;
            DATA->latched      = 0;
        } else {
            DATA->configured = 0;
        }
    } /* end new-command-if */

    double Freact[6]; /* force stacked w/ torque */
    double A = 1.0; /* amplitude scaling, function of react frequency TODO */
    double f = 0.0; /* reaction frequency, Hz TODO */
    double phi = 0.0; /* reaction phase, rads */
    double freq_eps = 0.01; /* lowest frequency to react to, Hz */
    double t = fspGetSimTime();
    double ang[DOF];
    double J[6][7], JT[7][6];

    for (d = 0; d<6; ++d){
        /* frequency reaction portion */
        if ( d == DATA->actDOF ) {
            if ( f >= freq_eps ) {
                Freact[d] = A*sin( 2*M_PI* f * t + phi );
            }
        } else {
            Freact[d] = 0.0;
        }
        /* ACS feed-forward portion */
        if ( d <= 3 ) {
            ARM->Fcmd_b[d] = ARM->acsF[d] + Freact[d];
        } else {
            ARM->Fcmd_b[d] = ARM->acsM[d-3] + Freact[d];
        }
    }

    for ( k=0; k<DOF; ++k ) {
        ang[k] = ARM->ang[k][0];
    }
    /* calculate jacobian */
    jac3_frend( PARAMS->d, ang, J );

    /* determine frame transforms */
    Qmult_rt( ARM->q_base2link3,  ARM->q_i2link3, -1, ARM->q_i2base, -1 );

    /* calc tool force in necessary frame */
    double *F_3 = &ARM->Fcmd_3[0];
    double *M_3 = &ARM->Fcmd_3[3];
    Qtrans_rt( F_3, ARM->q_base2link3, 1, &(ARM->Fcmd_b[0]));
    Qtrans_rt( M_3, ARM->q_base2link3, 1, &(ARM->Fcmd_b[3]));

    /* calc joint torques */
    MatrixTranspose( JT, J, 6, 7 );
    matmul( *JT, ARM->Fcmd_3, 7, 6, 1,  ARM->torq_cmd );

  return(0);
}
```

B.5.15 armVelMode_exit

Listing B.25: *Reactive Velocity Control Shutdown /*

```
int armVelMode_exit(armCtrl_s *ARM) {
  /* @brief Reactive Velocity Control Shutdown */
  if ( !ARM ) return (-1);
  return(0);
}
```

B.5.16 armVelMode_init

Listing B.26: *Reactive Velocity Control initialization /*

```
int armVelMode_init(armCtrl_s *ARM) {
  /* @brief Reactive Velocity Control initialization */
  if (!ARM) return (-1);

   fprintf(stderr,"%s(): sim time = %f\n\n", __FUNCTION__, fspGetSimTime() );
   ARM->manip_mode[0] = (double)ARM->velData.manip_mode;

   memcpy( ARM->velData.finalp0T , ARM->p0T,         3*sizeof(double));
   memcpy( ARM->velData.finalRPY , ARM->rpy0T_deg , 3*sizeof(double));
   ARM->velData.finalRPY[0] *= deg2rad;
   ARM->velData.finalRPY[1] *= deg2rad;
```

```
    ARM->velData.finalRPY[2] *= deg2rad;
    memcpy( ARM->velData.q_final, ARM->q_mb2tt, 4*sizeof(double));
    ARM->velData.finalSEW = ARM->sew_deg[0] * deg2rad;

    memcpy( ARM->velData.desp0T, ARM->p0T, 3*sizeof(double));
    memcpy( ARM->velData.desRPY, ARM->rpy0T_deg, 3*sizeof(double));
    ARM->velData.desRPY[0] *= deg2rad;
    ARM->velData.desRPY[1] *= deg2rad;
    ARM->velData.desRPY[2] *= deg2rad;
    memcpy( ARM->velData.q_des, ARM->q_mb2tt, 4*sizeof(double));
    ARM->velData.desSEW = ARM->sew_deg[0] *deg2rad;

    ARM->velData.t_slew = 0;

    memset( ARM->velData.delta, 0.0, DOF*sizeof(double));

    memset( ARM->velData.deltaAng, 0.0, DOF*sizeof(double));

    ARM->velData.configured = 0; /* low until good command received */

```

```
  return(0);
}
```

B.6 acs.c

B.6.1 acs

```
System *acs(domain_t *domain ) {
  System *sys = fspNewSystem( domain,
                              "ACS",
                              "Attitude Control System",
                              sizeof(acs_s));

  /* function pointers */
  sys->init = acs_init;
  sys->rt   = NULL;
  sys->drt  = acs_drt;

  return(sys);
}
```

B.6.2 acs_drt

Listing B.27: *Discrete time Attitude Control System function*

```
static int acs_drt(  System* sys) {
/**
   @brief Discrete time Attitude Control System function
 */
   acs_s *acs = (acs_s*)sys->data;
   int    rtn = FSP_NO_ERROR;
   int    rc, i;

   /* copy for logging */
   acs->q_i2b_out[0] = acs->q_i2b[0];
   acs->q_i2b_out[1] = acs->q_i2b[1];
   acs->q_i2b_out[2] = acs->q_i2b[2];
   acs->q_i2b_out[3] = acs->q_i2b[3];
   acs->enabled_out[0] = acs->enabled[0];
```

```

    /* calculate an error quaternion */
    Qmult_rt( acs->q_d2b, acs->q_i2b, 1, acs->q_i2d, -1 );
    /* Q3Positive_rt( acs->q_b2d ); */

    /* get the euler angles */
    Quat2euler_rt( acs->err_rpy_deg, acs->q_d2b, "123");
    Scale_rt( acs->err_rpy_deg, RAD2DEG, acs->err_rpy_deg );

    acs->rate_err[0] = acs->w_i2b_b[0] - acs->wd_b[0];
    acs->rate_err[1] = acs->w_i2b_b[1] - acs->wd_b[1];
    acs->rate_err[2] = acs->w_i2b_b[2] - acs->wd_b[2];

    acs->angle_err[0] = 2.0 * acs->q_d2b[0] * RAD2DEG;
    acs->angle_err[1] = 2.0 * acs->q_d2b[1] * RAD2DEG;
    acs->angle_err[2] = 2.0 * acs->q_d2b[2] * RAD2DEG;

    if ( acs->enabled[0] ) {
        /* compute a control torque using Qfeedback */
        qFeedback( acs->torque_b_des, acs->q_d2b, acs->rate_err, acs->Kp, acs->
        rtn = FSP_DISCRETE_EVENT;
    } else {
        acs->torque_b_des[0] = acs->torque_b_des[1] = acs->torque_b_des[2] = 0.
        rtn = FSP_NO_ERROR;
    }

    /* limit desired torque, if enabled */
    if ( acs->en_torq_lim ) {
        if ( acs->torque_b_des[0] > 0.0 && acs->torque_b_des[0] > acs->torq_b_m
            acs->torque_b_des[0] = acs->torq_b_max[0];
        if ( acs->torque_b_des[0] < 0.0 && acs->torque_b_des[0] < -1.0*acs->tor
            acs->torque_b_des[0] = -1.0*acs->torq_b_max[0];
        if ( acs->torque_b_des[1] > 0.0 && acs->torque_b_des[1] > acs->torq_b_m
            acs->torque_b_des[1] = acs->torq_b_max[1];
        if ( acs->torque_b_des[1] < 0.0 && acs->torque_b_des[1] < -1.0*acs->tor
            acs->torque_b_des[1] = -1.0*acs->torq_b_max[1];
        if ( acs->torque_b_des[2] > 0.0 && acs->torque_b_des[2] > acs->torq_b_m
            acs->torque_b_des[2] = acs->torq_b_max[2];
        if ( acs->torque_b_des[2] < 0.0 && acs->torque_b_des[2] < -1.0*acs->tor
            acs->torque_b_des[2] = acs->torq_b_max[2];
    }

    memset( acs->torque_b_cmd, 0, 3*sizeof(double));
    if ( acs->en_out_filt ) {
        /* execute filter */
        for (i = 0; i < 3; ++i ) {
            rc = cascFilt( &(acs->torque_b_des[i]),
                           &(acs->torque_b_cmd[i]),
                           &(acs->outFilt[i]), 1 );
            if ( rc < 0 ) {
                fprintf(stderr,"%s: @ %f sec, DOF %d, filter step failed. In = %f
                        __FUNCTION__, fspGetSimTime(),  i+1,
                        acs->torque_b_des[i], acs->torque_b_cmd[i] );
                fflush(stderr);
            }
        }
    } else {
        /* copy des to cmd */
        for (i = 0; i < 3; ++i ) {
            acs->torque_b_cmd[i] = acs->torque_b_des[i];
        }
    }

    /* log the root-sum-square (RSS) of the body angular rate */
    acs->w_b_rss[0] = Norm_rt(acs->w_i2b_b) * RAD2DEG;

  return(rtn);
}
```

B.6.3 acs_init

Listing B.28: *ACS module parameter/input/output initialization function*

```
static int acs_init( System* sys ) {
/***
      @brief ACS module parameter/input/output initialization function
*/
    acs_s *acs = (acs_s*)sys->data;
    char  *name;
    char  str[1024];
    int     i;

    /* === Parameters === */
    FSPgetParamVec( acs->Kp,          3,
                    "ACS.Kp",
                    "per-axis proportional gains");
    FSPgetParamVec( acs->Kd,          3,
                    "ACS.Kd",
                    "per-axis derivative gains");
    FSPgetParamVec( acs->enabled,     1,
                    "ACS.enabled",
                    "positive indicates ACS can apply torques to body");
    FSPgetParamVec( acs->Earth_mu,    1,
                    "K.Earth.mu",
                    "gravitational constant of Earth ");
    FSPgetParamInt( acs->logRate,
                    "ACS.logRate",
                    "freespace logging rate");
    FSPgetParamVec( acs->period,      1,
                    "period",
                    "module execution period [sec]");
    FSPgetParamVec( acs->q_i2d,       4,
                    "q_i2d",
                    "desired inertial attitude [-]");
    FSPgetParamVec( acs->wd_b,        3,
                    "wd_b",
                    "desired body rate [rad/sec]");
    FSPgetParamStr( name,
                    "name",
                    "Vehicle name");
    fprintf(stderr,"%s: found name = '%s'\n",
            __FUNCTION__, name);
    FSPgetParamInt( acs->en_torq_lim,
                    "ACS.en_torq_lim",
                    "nonzero to enable torque limiting");
    FSPgetParamVec( acs->torq_b_max, 3,
                    "ACS.torq_max",
                    "maximum torque in body per axis to attempt commanding");
    FSPgetParamInt( acs->en_out_filt,
                    "ACS.en_out_filt",
                    "nonzero to enable output structural filters");

    fspSetDRTsampleTime( sys, acs->period[0] );

    /* === Inputs === */
    snprintf(str, 1023, "MPROP.base.Rcm_i");
    fprintf(stderr,"%s: searching for input '%s'\n",__FUNCTION__, str);
    FSPgetInputVec( acs->r_i2b_i,     3,     str,
                    "meters, vector from ECI origin to body CoM in ECI");
    FSPgetInputVec( acs->r_i2sun_i,  3,    "ENV.SOLSYS.sun.r",
                    "meters, vector from ECI origin to sun center in ECI");
    snprintf(str, 1023, "MPROP.base.q_i2b");
    fprintf(stderr,"%s: searching for input '%s'\n",__FUNCTION__, str);
    FSPgetInputVec( acs->q_i2b,       4,  str,
                    "quaternion from inertial to body frame" );
    snprintf(str, 1023, "MPROP.base.w_b");
    fprintf(stderr,"%s: searching for input '%s'\n",__FUNCTION__, str);
    FSPgetInputVec( acs->w_i2b_b,     3,     str,
                    "rad/sec, angular rate of body in body coords");
    snprintf(str, 1023, "MPROP.base.Vcm_i");
    fprintf(stderr,"%s: searching for input '%s'\n", __FUNCTION__, str);
    FSPgetInputVec( acs->v_i2b_i ,     3,      str ,
                        "m/sec, velocity of satellite wrt ECI, in ECI coords");
    snprintf(str, 1023, "MPROP.base.cm_b");
    fprintf(stderr,"%s: searching for input '%s'\n", __FUNCTION__, str);
    FSPgetInputVec( acs->r_b2cm_b ,    3,      str ,
                        "vector from body origin to center of mass, in body");
    /* === Outputs === */
    fprintf(stderr,"%s: creating outputs...\n", __FUNCTION__ );
    fflush(stderr);
    FSPnewOutput( acs->torque_b_des ,       3, 1, "torque_b_des",
                      acs->logRate, "desired torques about body X,Y,Z in Nm");
    FSPnewOutput( acs->torque_b_applied , 3, 1, "torque_b_applied",
                      acs->logRate, "applied torques about body XYZ in Nm");
    FSPnewOutput( acs->torque_b_cmd ,       3, 1, "torque_b_cmd",
                      acs->logRate, "commanded torques in body [Nm]");
    FSPnewOutput( acs->u_b2sun_b ,          3, 1, "u_b2sun_b",
                      acs->logRate, "unit vector to sun, in body coords");
    FSPnewOutput( acs->err_rpy_deg ,       3, 1, "err_rpy_deg",
                      acs->logRate, "degrees, Roll, Pitch, Yaw in LVLH");
    FSPnewOutput( acs->angle_err,        3, 1, "angle_error",
                  acs->logRate,
                  "roll pitch yaw error from desired, in deg");
    FSPnewOutput( acs->rate_err,         3, 1, "rate_error",
                  acs->logRate, "rate error from desired, in rad/sec");
    FSPnewOutput( acs->q_d2b,            4, 1, "q_d2b",
                  acs->logRate, "quaternion from desired to current body");
    FSPnewOutput( acs->q_i2b_out,        4, 1, "q_i2b",
                  acs->logRate, "quaternion from inertial to body (copy)");
    FSPnewOutput( acs->enabled_out,      1, 1, "enabled",
                  acs->logRate, "nonzero means ACS can apply torques to body."
    FSPnewOutput( acs->w_b_rss,          1, 1, "w_b_rss",
                  acs->logRate, "scalar, RSS of body angular rate, deg/sec");

    fprintf(stderr,"%s: completed successfully.\n", __FUNCTION__);
    fflush(stderr);

    FSP_assemble_IO_finished();

    if ( acs->en_out_filt ) {
        int rc;
        /* configured output filters */
        for (i = 0; i < 3; ++i ) {
            rc = cascFilt_initFromMatlab( &(acs->outFilt[i]),
                                          HIGHPASS_FS10_IIR_FPASS1_5_ORDER3_NSEC
                                          highpass_Fs10_IIR_Fpass1_5_order3_c,
                                          highpass_Fs10_IIR_Fpass1_5_order3_d);
            if ( rc < 0 ) {
                fprintf(stderr,"%s: failed to initialize output filter on DOF %d\
                          __FUNCTION__, i);
            }
        }
    }

    return(FSP_NO_ERROR);
}
```

B.6.4 qFeedback

Listing B.29: *Diagonal-matrix gain quaternion feedback with shortest path*

```
static int qFeedback( double *torq_b,
                      const double *q_d2b,
                      const double *werr,
                      const double *kq,
                      const double *kw ) {
/**
   @brief Diagonal-matrix gain quaternion feedback with shortest path
*/
   double dw[3], dq[3], qsign=0;
   double dotq = Dot_rt(q_d2b,q_d2b);

   qsign = (q_d2b[3] >= 0.0 ? 1.0 : -1.0 );

   Scale_rt( dq, qsign,      q_d2b );
   Scale_rt( dw, (1.0-dotq), werr  );

   /* PID control - diagonal gain matrices */
   torq_b[0] = -kq[0]*dq[0] - kw[0]*dw[0];
   torq_b[1] = -kq[1]*dq[1] - kw[1]*dw[1];
```

```
    torq_b[2] = -kq[2]*dq[2] - kw[2]*dw[2];

    return(FSP_NO_ERROR);
}
```

B.7 prop.c

B.7.1 prop_calcM

Listing B.30: *Calculate thruster mapping matrix /*

```
static int prop_calcM(System *sys ) {
    /* @brief Calculate thruster mapping matrix */
    prop_s    *PROP = (prop_s*)sys->data;
    int n = PROP->nThrusters;
    int c, r, nCols;
    /* working variables for pseudo-inverse law calculations */
    double *one;
    double   *Mtrans, *U, *S, *V, *pinvVn, *VnpVn, *Vn; /* SVD products, inter
    double   torqMmt[3]; /* temporary torque moment vector */

    /* thrust direction & torque moments pseudo-inverse of their direction ma
    PROP->M         = calloc( 6*n, sizeof(double));
    PROP->arm       = calloc(   n, sizeof(double*));
    for ( c=0;c<n;++c) {
       PROP->arm[c] = calloc(   3, sizeof(double));
    }

    /* construct M matrix */
    /* nRows = 6; */
    nCols = n;
    for ( c=0; c<nCols; ++c) {
        double thisArm[3];
        for (r=0;r<3;r++) {
           PROP->M[ c + r*nCols ] = PROP->dir_b[c][r] * PROP->maxThrust[0];
        }
        thisArm[0] = (PROP->pos_b[c][0] - PROP->cm_b[0]) * PROP->maxThrust[0];
        thisArm[1] = (PROP->pos_b[c][1] - PROP->cm_b[1]) * PROP->maxThrust[0];
        thisArm[2] = (PROP->pos_b[c][2] - PROP->cm_b[2]) * PROP->maxThrust[0];

        Cross_rt( torqMmt, thisArm, PROP->dir_b[c] );
        COPY3( PROP->arm[c], thisArm ); /* copy for debug logging */
        for (r=0;r<3;r++) { /* rows 3-5 */
            PROP->M[ c + (r+3)*nCols ] = torqMmt[r];
        }
    }

    PROP->pinvM = calloc( n * 6, sizeof(double));
    Mtrans      = calloc( n * 6, sizeof(double));
    /* nullspace scaling factor vector */
    PROP->fPlus = calloc( n, sizeof(double));
    /* SVD terms for fplus calculations, singly-indexed required for SVD routi
    S      = calloc(6,   sizeof(double));
    V      = calloc(n*n, sizeof(double));
    U      = calloc(6*6, sizeof(double));
    Vn     = calloc(n*(n-6), sizeof(double));
    VnpVn  = calloc(n*n, sizeof(double));
    pinvVn = calloc((n-6)*n,sizeof(double));

    Transpose_rt( Mtrans, PROP->M, 6, n );
    Svd_rt( Mtrans, n, 6, V, S, U, FALSE /* economy flag */ );

    /* pseudo-inverse of M is used in the distribution law */
    Pinv_rt( PROP->pinvM, PROP->M, (unsigned)6, (unsigned)n );

    /* calculation of the f-plus vector requires an SVD of M,    */
    /* V = [ V_0, V_{null} ], pull out V_null                      */
    /* i.e. Vnull = V(1:n,(n-5):n);                                 */
    for (r=0; r<n; ++r ) {
        for (c=0; c<(n-6); ++c ) {
            Vn[(r*(n-6))+c] = V[(r*n)+(6+c)];
        }
    } /* end V_{null} for loops */

    Pinv_rt( pinvVn, Vn, (unsigned)n, (unsigned)(n-6));
    MatMult_rt( VnpVn, Vn, n, (n-6), pinvVn, (n-6), n);
    one = calloc( n, sizeof(double));
    for (r=0; r<n; r++ ) {
        one[r] = 1.0;
    }
    MatMult_rt( PROP->fPlus, VnpVn, n, n, one, n, 1 );

    return(FSP_NO_ERROR);
}
```

B.7.2 prop_drt

Listing B.31: *prop_drtSystemsys* {

```
static int prop_drt( System *sys ) {
    /* PROP module discrete runtime function */
    prop_s    *PROP = (prop_s*)sys->data;
    int       i, actuation_changed;
    double    /*dv_b[3],*/ Fact[6], Fdes[6], dp, norm, normc;
    double    uc[3], ua[3]; /* unit vectors of commanded and applied force/torq
    fft_s *FFT;
    unsigned int d,  n, /* b, */ k;
    struct timespec start_ts, end_ts;
    double *sample = NULL;

    prop_calcM(sys); /* calculate M matrix, fplus vector, etc */


    PROP->forceCmd_b[0] = PROP->forceCmd_b[1] = PROP->forceCmd_b[2] = 0.0;

    Eq_rt( PROP->torqCmd_b, PROP->T_b_des );

    /* stack force and torque commands */
    Fdes[0] = PROP->forceCmd_b[0];
    Fdes[1] = PROP->forceCmd_b[1];
    Fdes[2] = PROP->forceCmd_b[2];
    Fdes[3] = PROP->torqCmd_b[0];
    Fdes[4] = PROP->torqCmd_b[1];
    Fdes[5] = PROP->torqCmd_b[2];

    double sum = 0.0;
    for (i=0;i<6;i++) sum += fabs(Fdes[i]);

    if ( sum == 0.0 ) {
        memset( PROP->tau0,  0, PROP->nThrusters*sizeof(double));
        PROP->gamma[0] = 0;
    }
    else {
        switch ( PROP->mapper ) {
        case PROP_MAPPER_PINV: {        /* psuedo-inverse */
            double x;

            MatMult_rt( PROP->tau0, PROP->pinvM,
                        PROP->nThrusters, 6, Fdes, 6, 1 );

            /* calculate scaling coefficient for null-space bias */
            PROP->gamma[0] = -1e9; /* something very negative in case all 'x' be
            for (n = 0; n<(unsigned)PROP->nThrusters; ++n ) {
                x = -1.0*( PROP->tau0[n] - PROP->tauMin ) / PROP->fPlus[n];
                if ( x > PROP->gamma[0] ) {
                    PROP->gamma[0] = x;
                }
            }
            break;
        }
        case PROP_PERFECT_ACTUATION: {
            /* perfect actuation - don't figure out what each thruster will do *
            break;
        }
        default:
            fprintf(stderr,"PROP: ERROR invalid mapper\n");
                return(FSP_ERROR);
        }
    }


    /* ---------------------------------------------------- **
    **   adjust thrust times (bias + clamping)   **
    ** ---------------------------------------------------- */
    if ( PROP->mapper != PROP_PERFECT_ACTUATION ) {
        for (n=0;n<(unsigned)PROP->nThrusters;++n) {

            /* compute thruster on-time percentages (+ null-space bias) */
            PROP->tau[n] = PROP->tau0[n] + PROP->gamma[0] * PROP->fPlus[n];

            /* truncate thrust below the minimum possible */
            if ( PROP->tau[n] < (PROP->tauMin - PROP->onTimeTol )) {
               PROP->tau[n]  = 0.0;
            }

            /* clip thrust above maximum */
            if ( PROP->tau[n] > 1.0 ) {
               PROP->tau[n] = 1.0;
            }

            PROP->thrustMag[n] = PROP->tau[n] * PROP->maxThrust[n];
         }

         /* ---------------------------------------------------------------- **
         **  Convert thruster magnitudes back to body force/torque to apply
  **
         ** ---------------------------------------------------------------- */
         MatMult_rt( Fact, PROP->M, 6, PROP->nThrusters, PROP->tau, PROP->nThrus

         actuation_changed = FALSE;
```

```
    for (i=0;i<3;i++) {
        /* determine if force/torque has changed since previous cycle */
        if ( fabs(Fact[i]   - PROP->force_b[i]) > 1e-9 ) actuation_changed =
        if ( fabs(Fact[i+3] - PROP->torq_b[i])  > 1e-9 ) actuation_changed =
        PROP->force_b[i]      = Fact[i];
        PROP->torq_b[i]       = Fact[i+3];
        PROP->forceErr_b[i]   = PROP->force_b[i] - PROP->forceCmd_b[i];
        PROP->torqErr_b[i]    = PROP->torq_b[i]  - PROP->torqCmd_b[i];
    }
    PROP->forceErrMag[0] = Norm_rt(PROP->forceErr_b);
    PROP->torqErrMag[0]  = Norm_rt(PROP->torqErr_b);

    if ( PROP->forceErrMag[0] < 1e-3 ) {
        PROP->forceErrDir_deg[0] = 0.0;
    }
    else {
        normc = Norm_rt( PROP->forceCmd_b );
        if ( normc > 1.0e-3 ) {
            norm = Norm_rt( PROP->force_b );
            if ( norm > 1.0e-3 ) {
                Scale_rt( uc, 1.0/normc, PROP->forceCmd_b );
                Scale_rt( ua, 1.0/norm,  PROP->force_b );
                dp = Dot_rt( uc, ua );
                if ( dp >= 1.0 )
                    PROP->forceErrDir_deg[0] = 0.0;
                else
                    PROP->forceErrDir_deg[0] = acos( dp ) * RAD2DEG;
            }
        }
        else {
            PROP->forceErrDir_deg[0] = 0.0;
        }

    }

    if ( PROP->torqErrMag[0] < 1e-3 ) {
        PROP->torqErrDir_deg[0] = 0.0;
    }
    else {
        normc = Norm_rt( PROP->torqCmd_b );
        if ( normc > 1.0e-3 ) {
            norm = Norm_rt( PROP->torq_b );
            if ( norm > 1.0e-3 ) {
                Scale_rt( uc, 1.0/normc, PROP->torqCmd_b );
                Scale_rt( ua, 1.0/norm,  PROP->torq_b );
                dp = Dot_rt( uc, ua );
                if ( dp >= 1.0 )
                    PROP->torqErrDir_deg[0] = 0.0;
                else
                    PROP->torqErrDir_deg[0] = acos( dp ) * RAD2DEG;
            }
        }
        else {
            PROP->torqErrDir_deg[0] = 0.0;
        }
    }

    /* ------------------------------------------------ **
    **  monitor duty-cycle and check for deadband  **
    ** ------------------------------------------------ */
    const float g0   = 9.809915;  /* m/s^2 at Earth's surface */
    double       mdot =  PROP->maxThrust[0] / (g0 * PROP->Isp[0] );

    PROP->inDeadband[0] = 1.0;
    for (n=0;n<(unsigned)PROP->nThrusters;++n) {
        /* duty cycle */
        PROP->accumTime[n] += PROP->sampleTime[0];
        PROP->accumDuty[n] += PROP->tau[n] * PROP->sampleTime[0];
        PROP->massflow[0]  += PROP->tau[n] * PROP->sampleTime[0] * mdot;
        PROP->dutyFrac[n]   = PROP->accumDuty[n] / PROP->monitorPeriod[0];
        if ( PROP->accumTime[n] >= PROP->monitorPeriod[0] ) {
            /* clear internal accumulation variables for duty cycle calculati
            PROP->accumTime[n] = 0.0;
            PROP->accumDuty[n] = 0.0;
        }

        /* deadband? */
        if ( PROP->tau[n] > PROP->tauMin ) {
            PROP->inDeadband[0] = 0.0;
        }
    }
} else {
    /* PROP_PERFECT_ACTUATION : copy command to actuated */
    COPY3( PROP->force_b , PROP->forceCmd_b );
    COPY3( PROP->torq_b , PROP->torqCmd_b  );
    actuation_changed = FALSE;
    for (i=0;i<3;i++) {
        /* determine if force/torque has changed since previous cycle */
        if ( fabs(PROP->force_b[i]) > 1e-9 ) actuation_changed = TRUE;
        if ( fabs(PROP->torq_b[i])  > 1e-9 ) actuation_changed = TRUE;
    }

}
/* @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ */
/* @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ */
/* @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ */
/* @@                                                   @@ */
/* @@          F F T                                    @@ */
/* @@                                                   @@ */
/* @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ */
```

```
/* @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ */
/* @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ */
if ( PROP->en_fft > 0 ) {
    for (d=0; d<PROP_DOF; ++d ) {
        FFT = &(PROP->FFT[d]);

        clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start_ts);

        /* ************************************** */
        /* circular buffer                        */
        /* ************************************** */
        /* add latest data to buffer */
        /* 1. update circbuf pointers and count */
        ++(FFT->nSamples);
        ++(FFT->winCnt);
        if ( FFT->nSamples > 1 ) {
            if ( FFT->bufHead >= FFT->bufEnd ) {
                FFT->bufHead = &(FFT->buf[0]);
            } else {
                ++(FFT->bufHead);
            }
        }
        if ( FFT->nSamples > FFT->winLen) {
            if ( FFT->bufTail >= FFT->bufEnd ) {
                FFT->bufTail = &(FFT->buf[0]);
            } else {
                ++(FFT->bufTail);
            }
        }

        /* 2. insert new data */
        if ( d < 3 ) {
            /* force DOF */
            FFT->bufHead[0] = PROP->force_b[d];
        } else {
            FFT->bufHead[0] = PROP->torq_b[d-3];
        }

        if ( FFT->winCnt >= FFT->winStride ) {
            FFT->winCnt = 0;
            FFT->newMeas[0] = 1;

            memset(FFT->fftWin, 0, FFT->winLen*sizeof(double));

            if ( FFT->nSamples <= FFT->winLen ) {    /* circular buffer hasn'
                for ( k=0; k<FFT->nSamples; ++k){
                    REAL(FFT->fftWin, k) = FFT->bufTail[k];
                    IMAG(FFT->fftWin, k) = 0.0;
                }
            } else {
                /* circular buffer has wrapped, copy in two parts */
                k = 0;
                for ( sample = FFT->bufTail; sample <=  FFT->bufEnd; ++sample,
                    REAL(FFT->fftWin, k) = *sample;
                    IMAG(FFT->fftWin, k) = 0.0;
                }
                for ( sample = FFT->buf; sample < FFT->bufTail; ++sample, ++k)
                    REAL(FFT->fftWin, k) = *sample;
                    IMAG(FFT->fftWin, k) = 0.0;
                }
            }

            /* ************************************ */
            /* ************************************ */
    #if 0
            FFT->fftRtn = gsl_fft_complex_radix2_forward( FFT->fftWin,
                                                          FFT->elemStride,
                                                          FFT->winLen);
            if ( FFT->fftRtn != GSL_SUCCESS) {
                fprintf(stderr,
                        "gsl_fft_complex_radix2_forward() failed with return %
                        FFT->fftRtn);
            }
            for ( b=0; b<FFT->nBins; ++b) {
                FFT->fftMag[b]   = fabs(REAL(FFT->fftWin, b));
                FFT->fftPhase[b] = IMAG(FFT->fftWin, b);
            }
    #else
            FFT->fftRtn = doFFT( FFT->fftMag, FFT->fftPhase,
                                 FFT->fftWin, FFT->elemStride,
                                 FFT->winLen, FFT->nBins );
    #endif

                } else {
            FFT->newMeas[0] = 0;
                }

                /* ************************************** */
                /* Find Peaks                             */
                /* ************************************** */
                if ( FFT->newMeas[0] > 0 ) {
            unsigned int nFound = 0;
            memset( FFT->peaks.amps, 0, FFT->maxNpeaks*sizeof(double));
            FFT->peaksRtn = findPeaks( &nFound,
                                       FFT->peaks.freqs,
                                       FFT->peaks.amps,
                                       FFT->peaks.phases,
                                       FFT->fftMag,
                                       FFT->freqBins,
                                       FFT->fftPhase,
                                                    FFT->nBins,
                                                    FFT->smoothSpan,
                                                    FFT->inflectN,
                                                    FFT->locMaxN,
                                                    FFT->maxNpeaks );
            FFT->peaks.nPeaks[0] = (double)nFound;
                } /* end newMeas check if */
                clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end_ts);

                FFT->compTime_ns  = (long)(end_ts.tv_sec - start_ts.tv_sec)*10000000
                FFT->compTime_ns += (long)end_ts.tv_nsec - (long)start_ts.tv_nsec;

            } /* end DOF for-loop */
        } /* end of en_fft if statement */

        if ( actuation_changed ) return(FSP_DISCRETE_EVENT);
        else                     return(FSP_NO_ERROR);
    }
```

B.7.3 prop_init

Listing B.32: *Initilize the PROP module /*

```
static int prop_init(System *sys ) {
   /* @brief Initilize the PROP module */
   prop_s    *PROP = (prop_s*)sys->data;
   int n;
   fft_s   *FFT = NULL;
   char *DOF_LAB[] = {"FX", "FY", "FZ", "MX", "MY", "MZ"};
   unsigned int winLen, winStride, elemStride, d, f, nBins, maxPeaks;
   unsigned int smoothSpan, inflectN, locMaxN;
   double Fs, delta;

   /* ---------------------------------------------------------------- */
   /* PARAMETERS FROM WORKSPACE
*/
   /* ---------------------------------------------------------------- */
   FSPgetParamVec( PROP->sampleTime,    1,
                         "cyclePeriod",
                          "sample time of AOCS's DRT function");
    FSPgetParamStr( PROP->mapperStr,
                          "mapper",
                          "name of thruster distribution method (e.g., 'pinv'" );
    FSPgetParamInt( PROP->errSeed,
                          "errSeed",
                          "seed for random number generator ");
    FSPgetParamVec( PROP->minOnTime,      1,
                          "minOnTime",
                          "minimum on time of a single thruster [sec]");
    FSPgetParamInt( PROP->thrusterDistro,
                          "thrusterDistro",
                          "nonzero indicates force/torque will be applied by THRUSTE
    FSPgetParamInt( PROP->nThrusters,
                          "nthrusters",
                          "number of thrusters");
    FSPgetParamVec( PROP->Isp,            1,
                          "Isp",
                          "specific impulse of fuel [sec]");
    n = PROP->nThrusters;
    FSPgetParamMat( PROP->pos_b,        n,3,
                          "pos_b",
                          "position of each thruster in body frame [m]");
    FSPgetParamMat( PROP->dir_b,        n,3,
                          "dir_b",
                          "direction of thrust per thruster, unit vectors");
    FSPgetParamVec( PROP->maxThrust,    n,
                          "maxThrust",
                          "maximum thrust per thruster [N]");
    FSPgetParamVec( PROP->quantum,      1,
                          "quantum",
                          "on-time quantization [sec]");
    FSPgetParamMat( PROP->Icm,          3,3,
                          "MPROP.Icm",
                          "moments of inertia (about CM) estimate for RSV [kg-m^2]")
    FSPgetParamVec( PROP->mass,         1,
                          "MPROP.m0",
                          "current mass estimate from RSV [kg]");
    FSPgetParamVec( PROP->monitorPeriod,1,
                          "monitorPeriod",
                          "period over which to monitor individual thruster (on-time
      );
    /* FFT parameters */
    FSPgetParamInt( PROP->en_fft,
                          "FFT.en",
                          "nonzero to enable FFT output");
    FSPcopyParam1d( Fs,
                          "FFT.Fs",
                          "sampling frequency [Hz]");
    FSPgetParamInt( winLen,
                          "FFT.winLen",
                          "# samples in each FFT window [-]");
    FSPgetParamInt( winStride,
                          "FFT.winStride",
                          "# samples to skip between executing FFT window [-]");
    FSPgetParamInt( elemStride,
                          "FFT.elemStride",
                          "# samples to skip within a window [-]");
    FSPgetParamInt( maxPeaks,
                          "FFT.maxNpeaks",
                              "max # of fft peaks to report");
          FSPgetParamInt( smoothSpan,
                              "FFT.smoothSpan",
                              "# samplies in span of smoothing window");
          FSPgetParamInt( inflectN,
                              "FFT.inflectN",
                              "single-side neighborhood for inflection point search");
          FSPgetParamInt( locMaxN,
                              "FFT.locMaxN",
                              "single-side neighborhood for local maximum search");



          /* >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */
          /* DYNAMIC INPUTS & STATES
       */
          /* >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */
          FSPgetInputVec( PROP->q_i2b,          4, "MPROP.base.q_i2b",
       "engine truth for base bod
      y inertial attitude [-]");
```

```
    FSPgetInputVec( PROP->T_b_des,        3, "ACS.torque_b_cmd",
 "desired torques about bod
y [Nm]");
    FSPgetInputVec( PROP->cm_b,           3, "MPROP.base.Rbary_b",
 "barycenter (COM) of all bo
dies in vehicle [m]");

    /* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< */
    /* DYNAMIC OUTPUTS
 */
    /* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< */
    FSPnewOutput( PROP->gamma,           1, 1,
                  "gamma",
                  FSP_LOG1x,
                  "thrust correction for SSL's pseudo-inverse method ");
    FSPnewOutput( PROP->thrustMag,       n, 1,
                  "thrustMag",
                  FSP_LOG1x,
                  "magnitude of thrust for each thruster [N]");
    FSPnewOutput( PROP->tau0,            n, 1,
                  "tau0",
                  FSP_LOG1x,
                  "thruster on-time fraction of control-cycle (prior to bias o

  FSPnewOutput( PROP->tau,            n, 1,
                  "tau",
                  FSP_LOG1x,
                  "actual thruster on-time fraction of control-cycle (after bi
;
    FSPnewOutput( PROP->dutyFrac,       n, 1,
                  "dutyFrac",
                  FSP_LOG1x,
                  "thruster duty-cycle fraction over monitoring-peroid [-]" );
    FSPnewOutput( PROP->forceCmd_b,     3, 1,
                  "forceCmd_b",
                  FSP_LOG1x,
                  "commanded delta-V as force in body frame [N]");
    FSPnewOutput( PROP->torqCmd_b,      3, 1,
                  "torqCmd_b",
                  FSP_LOG1x,
                  "commanded delta-omega as torque in body frame [N-m]");
    FSPnewOutput( PROP->force_b,          3, 1,
                  "force_b",
                  FSP_LOG1x,
                  "force in body frame, actually applied [N]");
    FSPnewOutput( PROP->torq_b,           3, 1,
                  "torq_b",
                  FSP_LOG1x,
                  "torque in body frame, actually applied [N-m]");
    FSPnewOutput( PROP->forceErr_b,       3, 1,
                  "forceErr_b",
                  FSP_LOG1x,
                  "error between actuated forces and TCS desired [N]");
    FSPnewOutput( PROP->torqErr_b,        3, 1,
                  "torqErr_b",
                  FSP_LOG1x,
                  "error between actuated torques and ACS desired [N-m]");
    FSPnewOutput( PROP->forceErrMag,      1, 1,
                  "forceErrMag",
                  FSP_LOG1x,
                  "magnitude of applied force error [N]");
    FSPnewOutput( PROP->forceErrDir_deg ,1, 1,
                  "forceErrDir_deg",
                  FSP_LOG1x,
                  "direction of applied force error [deg]");
    FSPnewOutput( PROP->torqErrMag,       1, 1,
                  "torqErrMag",
                  FSP_LOG1x,
                  "magnitude of applied torque error [N-m]");
    FSPnewOutput( PROP->torqErrDir_deg, 1, 1,
                  "torqErrDir_deg",
                  FSP_LOG1x,
                  "direction of applied torque error [deg]");
    FSPnewOutput( PROP->res_torq_b,       3, 1,
                  "res_torq_b",
                  FSP_LOG1x,
                  "residual torque on body from fspApplyForce() calls");
    FSPnewOutput( PROP->inDeadband,       1, 1,
                  "inDeadband",
                  FSP_LOG1x,
                  "nonzero indicates all commanded on-times were below the min
);
    FSPnewOutput( PROP->massflow,          1, 1,
                  "massflow",
                  FSP_LOG1x,
                  "accumulated amount of mass expended by all thrusters based


    if ( PROP->en_fft > 0 ) {
        /* allocate per-DOF memory for FFTs */
        for ( d = 0; d < PROP_DOF; ++d ) {
            FFT = &(PROP->FFT[d]);

            nBins = (unsigned)(( winLen / 2 ) + 1);
            FFT->nBins = nBins; /* a local variable makes later calls shorter :/

            /* allocate */
            FFT->fftWin    = calloc(winLen, sizeof(double));


            /* copy locally */
            FFT->winLen     = winLen;
            FFT->winStride  = winStride;
            FFT->elemStride = elemStride;
            FFT->Fs         = Fs / (double)elemStride;
            FFT->maxNpeaks  = maxPeaks;
            FFT->smoothSpan = smoothSpan;
            FFT->inflectN   = inflectN;
            FFT->locMaxN    = locMaxN;

            snprintf(FFT->name, FFT_NAME_MAX_LEN-1, "%s", DOF_LAB[d]);

            /* declare as telemetry */
            fspSetVarExpandTags(sys, 1, DOF_LAB[d]);
            FSPnewOutput( FFT->freqBins,     nBins, 1,
                          "%.freqBins",
                          FSP_LOG0,
                          "frequency labels for FFT output bins [Hz]");
            FSPnewOutput( FFT->fftMag,       nBins, 1,
                          "%.fftMag",
                          FSP_LOG1x,
                          "magnitude of FFT output [?]");
            FSPnewOutput( FFT->fftPhase,      nBins, 1,
                          "%.fftPhase",
                          FSP_LOG1x,
                          "phase of FFT output [rads]");
            fspTelemetry( &FFT->compTime_ns, FSP_LONG,   1,
                          "%.compTime_ns",
                          FSP_LOG0,
                          "time required for FFT computation, including copies,
            fspTelemetry( &FFT->fftRtn,       FSP_INT,    1,
                          "%.fftRtn",
                          FSP_LOG0,
                          "FFT alg. return status");
            fspTelemetry( &FFT->peaksRtn,     FSP_INT,    1,
                          "%.peaksRtn",
                          FSP_LOG0,
                          "findPeaks() return status");
            FSPnewOutput( FFT->buf,          winLen,1,
                          "%.buf",
                          FSP_LOG0,
                          "Input buffer to FFT");
            FSPnewOutput( FFT->newMeas,              1,1,
                          "%.newMeas",
                          FSP_LOG1x,
                          "nonzero when new FFT output is available [bool]");
            FSPnewOutput( FFT->peaks.freqs, maxPeaks, 1,
                          "%.peakFreqs",
                          FSP_LOG1x,
                          "Frequencies of peaks [Hz]");
            FSPnewOutput( FFT->peaks.amps,  maxPeaks, 1,
                          "%.peakAmps",
                          FSP_LOG1x,
                          "Normalized Amplitudes of peaks [-]");
            FSPnewOutput( FFT->peaks.phases, maxPeaks,1,
                          "%.peakPhases",
                          FSP_LOG1x,
                          "Phases of peaks [Rad]");
            FSPnewOutput( FFT->peaks.nPeaks,      1, 1,
                          "%.nPeaks",
                          FSP_LOG1x,
                          "number of peaks found");

            /* compute frequency bin labels */
            delta = 1.0/((double)winLen/2.0) * (Fs/2.0);
            FFT->freqBins[0] = 0.0;
            for ( f = 1; f < FFT->nBins; ++f ) {
               FFT->freqBins[f] = (double)f * delta -0.5*delta; /* center freq o
            }


            /* per-DOF initialization */
            FFT->nSamples = 0;
            FFT->bufTail  = &(FFT->buf[0]);
            FFT->bufHead  = &(FFT->buf[0]);
            FFT->bufEnd   = &(FFT->buf[FFT->winLen-1]);

         } /* end of DOF for-loop */
      } /* end of en_fft if statement */

```

```
    /* ---------------- */ FSP_assemble_IO_finished();/* ---------------- */

    if        ( strcasecmp( PROP->mapperStr, "pinv" )==0    )
        PROP->mapper = PROP_MAPPER_PINV;
    else
        if ( strcasecmp( PROP->mapperStr, "perfect" )==0 )
            PROP->mapper = PROP_PERFECT_ACTUATION;

    PROP->tauMin     = PROP->minOnTime[0] / PROP->sampleTime[0]; /* NB: this as
have same minimum on time - BD */
    PROP->onTimeTol = PROP->quantum[0] / PROP->sampleTime[0];
    PROP->errTol     = PROP->quantum[0] * PROP->maxThrust[0];

    /* for per-thruster duty cycle calculations */
    PROP->accumDuty = calloc(n, sizeof(double));
    PROP->accumTime = calloc(n, sizeof(double));

    int    *matches, i;
    char   bodyName[] = "$DOMAIN/base";
    int    found = fspBodyFindId( bodyName, &matches );
    if ( !found ) {
        fprintf(stderr,"prop_init(): Failed to find VSD body ID for '%s'\n", bo
        fprintf(stderr,"\tnumber of body-matches found = %d; [ ", found);
        for (i=0;i<found;i++) {
            fprintf(stderr,"%d", matches[i]);
            if ( i < (found-1)) { fprintf(stderr,", "); }
        }
        return(FSP_ERROR);
    }
    PROP->busBodyID = matches[0];

    fspSetDRTsampleTime( sys, PROP->sampleTime[0] );

    PROP->maxErrThrust = 0.0;
    PROP->maxErrTorque = 0.0;

    prop_calcM(sys); /* calculate M matrix, fplus vector, etc */

    return(FSP_NO_ERROR);
}
```

B.7.4 prop_rt

```
static int prop_rt(  System *sys )
{
    prop_s *PROP = (prop_s*)sys->data;

    if ( PROP->thrusterDistro == 0 ) {
        /* apply force/torque locally --- ignore thruster module */
        fspBodyApplyForce( PROP->busBodyID,
                           PROP->force_b,
                           PROP->cm_b,
                           PROP->res_torq_b  );
        fspSysApplyTorque( sys,
                           PROP->torq_b,
                           PROP->busBodyID,
                           0  );
    }

    return(FSP_NO_ERROR);
}
```

Bibliography

[1] Nasa space science data coordinated archive. https://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1957-001B.

[2] Union of concerned scientists satellite database. https://www.ucsusa.org/nuclear-weapons/space-weapons/satellite-database#.W8OjlJxRcQg.

[3] Timothy Reedman, Dan King, and Pat Malaviarchchi. The economic case for satellite servicing. Vancouver, Canada, 2004. 55th International Astronautical Congress.

[4] Russell L Werneth. Lessons learned from Hubble space telescope extravehicular activity servicing missions. Technical report, SAE Technical Paper, 2001.

[5] Andrew Long, Matthew Richards, and Daniel E. Hastings. On-orbit servicing: A new value proposition for satellite design and operation. *Journal of Spacecraft and Rockets*, 44(4):964–976, 2007.

[6] Carole Joppin and Daniel Hastings. Upgrade and repair of a scientific mission using on-orbit servicing based on the Hubble space telescope example. In *Space 2004 Conference and Exhibit*, page 6052, 2004.

[7] Bo J. Naasz, III Frank L. Culbertson, Joseph F. Pellegrino, and Benjamin B. Reed. *Business Case for Geosynchronous Satellite Servicing: Refueling, Repair, and Inspection.* American Institute of Aeronautics and Astronautics, 2014/08/10 2013.

[8] Brook R Sullivan, David L Akin, and Gordon Roesler. A parametric investigation of satellite servicing requirements, revenues, and options in geostationary orbit. In *AIAA SPACE 2015 Conference and Exposition*, page 4477, 2015.

[9] Alex Ellery, Joerg Kreisel, and Bernd Sommer. The case for robotic on-orbit servicing of spacecraft: Spacecraft reliability is a myth. *Acta Astronautica*, 63(5-6):632–648, 2008.

[10] Orbital ATK unveils new version of satellite servicing vehicle. `https://spacenews.com/orbital-atk-unveils-new-version-of-satellite-servicing-vehicle/`, March 2018.

[11] Effective Space signs first contract for satellite life extension services. `https://spacenews.com/effective-space-signs-first-contract-for-satellite-life-extension-ser` January 2018.

[12] Marshall H. Kaplan, Bradley Boone, Robert Brown, Thomas B. Criss, and Edward W. Tunstel. Engineering issues for all major modes of in situ space debris capture. In *AIAA SPACE 2010 Conference and Exposition*. AIAA, 30 Aug - 2 Sep 2010.

[13] Wolfgang Priester, Max Römer, and H Volland. The physical behavior of the upper atmosphere deduced from satellite drag data. *Space Science Reviews*, 6(6):707–780, 1967.

[14] Alexandra Witze. Software error doomed japanese hitomi spacecraft, 2016.

[15] Jin Choi, Jung Hyun Jo, Myung-Jin Kim, Dong-Goo Roh, Sun-Youp Park, Hee-Jae Lee, Maru Park, Young-Jun Choi, Hong-Suh Yim, Young-Ho Bae, et al. Determining the rotation periods of an inactive leo satellite and the first korean space debris on geo, koreasat 1. *Journal of Astronomy and Space Sciences*, 33(2):127–135, 2016.

[16] Fumihito Sugai, Satoko Abiko, Teppei Tsujita, Xin Jiang, and Masaru Uchiyama. Detumbling an uncontrolled satellite with contactless force by using an eddy current brake. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 783–788. IEEE, 2013.

[17] Fumihito Sugai, Satoko Abiko, Teppei Tsujita, Xin Jiang, and Masaru Uchiyama. Development of an eddy current brake system for detumbling malfunctioning satellites. In *System Integration (SII), 2012 IEEE/SICE International Symposium on*, pages 325–330. IEEE, 2012.

[18] James Shoemaker and Melissa Wright. Orbital express space operations architecture program. In *Defense and Security*, pages 57–65. International Society for Optics and Photonics, 2004.

[19] Ioannis Rekleitis, Eric Martin, Guy Rouleau, Régent L'Archevêque, Kourosh Parsa, and Eric Dupuis. Autonomous capture of a tumbling satellite. *Journal of Field Robotics*, 24(4):275–296, 2007.

[20] Angel Flores-Abad, Ou Ma, Khanh Pham, and Steve Ulrich. A review of space robotics technolgies for on-orbit servicing. *Progress in Aerospace Sciences*, 68:1–26, 2014.

[21] Bong Wie. *Space Vehicle Dynamics and Control*. AIAA Education Series, 2nd edition, 2008.

[22] Mark C. Jackson. On orbit flight control stability analysis and design process. Technical report, Draper Laboratory, December 1998.

[23] M Azadi, SA Fazelzadeh, M Eghtesad, and E Azadi. Vibration suppression and adaptive-robust control of a smart flexible satellite with three axes maneuvering. *Acta Astronautica*, 69(5):307–322, 2011.

[24] S Di Gennaro. Active vibration suppression in flexible spacecraft attitude tracking. *Journal of Guidance, Control, and Dynamics*, 21(3):400–408, 1998.

[25] Stefano Di Gennaro. Output stabilization of flexible spacecraft with active vibration suppression. *IEEE Transactions on Aerospace and Electronic systems*, 39(3):747–759, 2003.

[26] Qinglei Hu and Guangfu Ma. Variable structure control and active vibration suppression of flexible spacecraft during attitude maneuver. *Aerospace Science and Technology*, 9(4):307–317, 2005.

[27] QL Hu and GF Ma. Vibration suppression of flexible spacecraft during attitude maneuvers. *Journal of guidance, control, and dynamics*, 28(2):377–380, 2005.

[28] John L Meyer, William B Harrington, Brij N Agrawal, and Gangbing Song. Vibration suppression of a spacecraft flexible appendage using smart material. *Smart Materials and Structures*, 7(1):95, 1998.

[29] Choong-Seok Oh, Hyochoong Bang, and Chang-Su Park. Attitude control of a flexible launch vehicle using an adaptive notch filter: Ground experiment. *Control Engineering Practice*, 16(1):30 – 42, 2008.

[30] Marco Sabatini, Paolo Gasbarri, Riccardo Monti, and Giovanni Battista Palmerini. Vibration control of a flexible space manipulator during on orbit operations. *Acta astronautica*, 73:109–121, 2012.

[31] William Singhose, Steve Derezinski, Neil Singer, et al. Extra-insensitive input shapers for controlling flexible spacecraft. *Journal of Guidance, Control, and Dynamics*, 19(2):385–391, 1996.

[32] Gangbing Song and Brij N Agrawal. Vibration suppression of flexible spacecraft during attitude control. *Acta Astronautica*, 49(2):73–83, 2001.

[33] G Song, SP Schmidt, and BN Agrawal. Experimental robustness study of positive position feedback control for active vibration suppression. *Journal of guidance, control, and dynamics*, 25(1):179–182, 2002.

[34] P. Zarafshan and S.A.A. Moosavian. Fuzzy tuning manipulation control of a space robot with passive flexible solar panels. In *Mechatronics and Automation (ICMA), 2011 International Conference on*, pages 404–409, Aug 2011.

[35] Everett Findlay, James R Forbes, Hugh HT Liu, Anton de Ruiter, Christopher J Damaren, and James Lee. *Investigation of Active Vibration Suppression of a Flexible Satellite Using Magnetic Attitude Control*. PhD thesis, University of Toronto, 2011.

[36] Bong Wie and Peter M Barba. Quaternion feedback for spacecraft large angle maneuvers. *Journal of Guidance, Control, and Dynamics*, 8(3):360–365, 1985.

[37] Bong Wie, H Weiss, and A Arapostathis. Quarternion feedback regulator for spacecraft eigenaxis rotations. *Journal of Guidance, Control, and Dynamics*, 12(3):375–380, 1989.

[38] JT-Y Wen and Kenneth Kreutz-Delgado. The attitude control problem. *IEEE Transactions on Automatic control*, 36(10):1148–1162, 1991.

[39] SM Joshi, AG Kelkar, and JT-Y Wen. Robust attitude stabilization of spacecraft using nonlinear quaternion feedback. *IEEE Transactions on Automatic control*, 40(10):1800–1803, 1995.

[40] F Landis Markley and John L Crassidis. *Fundamentals of spacecraft attitude determination and control*, volume 33. Springer, 2014.

[41] J Thienel and Robert M Sanner. A coupled nonlinear spacecraft attitude controller and observer with an unknown constant gyro bias and gyro noise. *IEEE Transactions on Automatic Control*, 48(11):2011–2015, 2003.

[42] Peter C Hughes. *Spacecraft attitude dynamics*. Courier Corporation, 2012.

[43] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976.

[44] Peter J Wiktor. Minimum control authority plot-a tool for designing thruster systems. *Journal of Guidance, Control, and Dynamics*, 17(5):998–1006, 1994.

[45] Kazuya Yoshida, Kenichi Hashizume, Dragomir N Nenchev, Noriyasu Inaba, and Mitsushige Oda. Control of a space manipulator for autonomous target capture-ets-vii flight experiments and analysis. In *AIAA Guidance, Navigation, and Control Conference*, pages 14–17, 2000.

[46] Dragomir N. Nenchev, Kazuya Yoshida, Prasert Vichitkulsawat, and Masaru Uchiyama. Reaction null-space control of flexible structure mounted manipulator systems. In *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION,*, 1999.

[47] K. Yoshida, K. Hashizume, and S. Abiko. Zero reaction maneuver: flight validation with ets-vii space robot and extension to kinematically redundant arm. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 441–446, 2001.

[48] Yusuke Fukazu, Naoyuki Hara, Yoshikazu Kanamiya, and Daisuke Sato. Reactionless resolved acceleration control with vibration suppression capability for jemrms/sfa. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 1359–1364. IEEE, 2009.

[49] Soo Han Lee and Wayne John Book. Robot vibration control using inertial damping forces. In *VIII CISM-IFToMM Symposium on the Theory and Practice of Robots and Manipulators (Ro. Man. Sy.'90)*, 1990.

[50] Seiichiro Washino, Michael Shoemaker, and Masaru Uchiyama. Application of reactionless motion to teleoperation of a flexible base dual-arm manipulator. In *Proc. of Japan Society of Instrument and Control Engineers, Northeastern Chapter, Student Conference*, 2003.

[51] T. Wongratanaphisan and M. Cole. Robust impedance control of a flexible structure mounted manipulator performing contact tasks. *Robotics, IEEE Transactions on*, 25(2):445–451, April 2009.

[52] H Morimoto, Y Kasama, S Doi, and Y Wakabayashi. Motion control of the small fine arm for the japanese experiment module of the international space station. *ICAR*, 2001.

[53] Enrico Sabelli, David L Akin, and Craig R Carignan. Selecting impedance parameters for the ranger 8-dof dexterous space manipulator. In *Proceedings of AIAA Aerospace Conference and Exhibit.* AIAA Rohnert Park, California, USA, 2007.

[54] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing.* Prentice Hall, 1975.

[55] Richard G. Lyons. *Understanding Digital Signal Processing.* Prentice Hall, 2nd edition, 2004.

[56] Steven Queen, Ken London, and Marcelo Gonzalez. Momentum-based dynamics for spacecraft with chained revolute appendages. In *Flight Mechanics Symposium 2005*, 2005.

[57] Bo J Naasz, Richard D Burns, Steven Z Queen, John Van Eepoel, Joel Hannah, and Eugene Skelton. The hst sm4 relative navigation sensor system: Overview and preliminary testing results from the flight robotics lab. *The Journal of the Astronautical Sciences*, 57(1-2):457–483, 2009.

[58] Matthew J Strube, Andrew M Hyslop, Craig R Carignan, and Joseph W Easley. Ground simulation of an autonomous satellite rendezvous and tracking system using dual robotic systems. 2012.

[59] Steven Z. Queen, Dean J. Chai, and Sam Placanica. Orbital maneuvering system design and performance for the magnetospheric multiscale formation. In *AAS/AIAA Astrodynamics Specialist Conference*, number 815, Aug 2015.

[60] Steven Z. Queen, Neerav Shah, Suyog S. Benegalrao, and Kathie Blackman. Generalized momentum control of the spin-stabilized magnetospheric multiscale formation. In *AAS/AIAA Astrodynamics Specialist Conference*, number 816, Aug 2015.

[61] Steven Z. Queen. A kalman filter for mass property and thrust identification of the spin-stabilized magnetospheric multiscale formation, 19-23 Oct 2015.

[62] Jean H Meeus. *Astronomical algorithms.* Willmann-Bell, Incorporated, 1991.

[63] FG Lemoine, DE Smith, L Kunz, R Smith, EC Pavlis, NK Pavlis, SM Klosko, DS Chinn, MH Torrence, RG Williamson, et al. The development of the nasa gsfc and nima joint geopotential model. In *Gravity, geoid and marine geodesy*, pages 461–469. Springer, 1997.

[64] Eric Stoneking. *Newton-Euler Dynamic Equations of Motion for a Multi-Body Spacecraft.* American Institute of Aeronautics and Astronautics, 2014/11/12 2007.

[65] Trent Yang. *Optimal thruster selection with robust estimation for formation flying applications.* PhD thesis, Massachusetts Institute of Technology, 2003.

[66] J Cole Smith and Z Caner Taskin. A tutorial guide to mixed-integer programming models and solution techniques. *Optimization in Medicine and Biology*, pages 521–548, 2008.

[67] Asher Smith and Dongeun Seo. Spacecraft thruster distribution matrix for precision 6dof control. In *AIAA SPACE and Astronautics Forum and Exposition*, page 5203, 2017.

[68] John Craig. *Introduction to Robotics*. Pearson Prentice Hall, 3rd edition, 2005.