NONLINEAR TRACKING OF NATURAL MECHANICAL

SYSTEMS FOR HWIL SIMULATION

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

_____

Justin N. Martin

Certificate of Approval:

_____	_____
John E. Cochran, Jr. Professor Aerospace Engineering	Andrew J. Sinclair, Chair Assistant Professor Aerospace Engineering
_____	_____
David A. Cicci Professor Aerospace Engineering	Joe F. Pittman Interim Dean Graduate School

NONLINEAR TRACKING OF NATURAL MECHANICAL SYSTEMS FOR HWIL SIMULATION

Justin N. Martin

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
August 4, 2007

NONLINEAR TRACKING OF NATURAL MECHANICAL

SYSTEMS FOR HWIL SIMULATION

Justin N. Martin

_____

Signature of Author

_____

Date of Graduation

THESIS ABSTRACT

NONLINEAR TRACKING OF NATURAL MECHANICAL

SYSTEMS FOR HWIL SIMULATION

Justin N. Martin

Master of Science, August 4, 2007
(B.S. Aero. Eng., West Virginia University, 2003)
(B.S., Mech. Eng., West Virginia University, 2003)

123 Typed Pages

Directed by Andrew Sinclair

Auburn University has entered into collaboration with the US Department of Defense for academic study and development of hardware-in-the-loop simulation laboratory. One aspect of this collaboration has been research into new concepts for the control of flight motion tables, a critical component in HWIL simulations.

Commonly used Proportional-Integral-Derivative (PID) controllers can suffer limitations in applications with nonlinear and multi-input/multi-output systems. To overcome these limitations, a nonlinear dynamic-inversion controller was developed. Applying Lagrange's equations to determine equations of motion, a Lyapunov function was used to develop a globally asymptotically stable controller.

After comparing PID and dynamic-inversion controllers through multiple commanded motions and adjustments to gain, the dynamic-inversion was more stable and produces less error. Both controllers are capable of performing real-time applications.

ACKNOWLEDGMENTS

The author would like to thank Scottie Mobley of the Aviation and Missile Research Development and Engineering Center and Ryan Brindley and Jeffrey Gareri of Simulation Technologies, Inc. for their continued assistance with HWIL systems.

Style manual or journal used:	Modern Language Association Style Manual
Computer software used:	Microsoft Office Word 2003 Microsoft Office Excel 2003 Matlab 7.3.0 (R2006b) UGS Solid Edge V19

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Prior to Hardware-in-the-Loop (HWIL) simulations, analyses of missile seeker head performance were conducted by live-fire tested at firing ranges. Missiles, with the seeker heads already installed, were sold in batches. To demonstrate acceptable performance, a certain percentage from each batch was field tested. Not only was this risky with the entire sale depending on a small, random potion of the batch, but it was also expensive with guaranteed losses of the tested missiles.

HWIL systems simplify this process of testing seeker heads. Rather than needing an entire range to fire a missile, HWIL systems use a flight motion table and scene generation to simulate the flight of a seeker head on a missile. The seeker head is placed in the gimbaled flight motion table while the simulated target is located at a stationary point in front of the flight table. Using the scene generator and synthetic lines of sight, the seeker simulates the tracking of a target.

Figure 1a (reproduced from Reference [High Performance]) and Figure 1b (reproduced from Reference [Hardware]) demonstrate the basic layout of a HWIL laboratory. Figure 1a displays the components and connections for an HWIL simulation, while Figure 1b is

a photograph of the Electronic Combat Simulation and Evaluation Laboratory (ECSEL) located in Point Mugu, CA.

Figure 1a: Layout of HWIL Laboratory

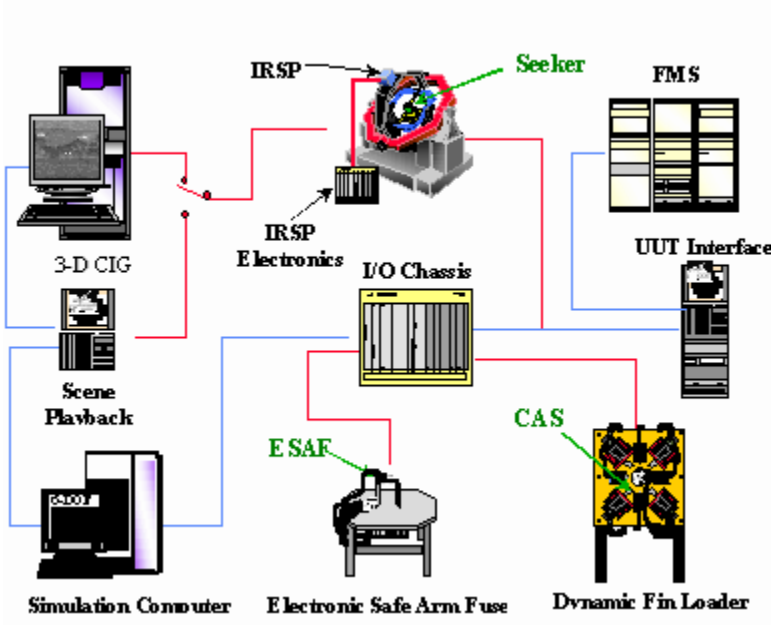


Figure 1b: ECSEL Laboratory in Point Mugu, CA

In the summer of 2006, Auburn University, in collaboration with Simulation Technologies, Inc. and the US Department of Defense, began a program to receive, install and test a flight table for a HWIL system. One of the goals of Auburn University is to study the design and implementation of controllers to guide the gimbaled system. The following paper presents the design and testing of a nonlinear dynamic-inversion controller with comparison to a Proportional-Integral-Derivative (PID) controller.

II. FLIGHT MOTION TABLES

Flight motion tables consist of multiple gimbaled joints whose motion is generated by hydraulic or electric actuators. The goal of using a flight table in a HWIL simulation is to reproduce some aspect of the rotational motion of the flight hardware. They are designed to accurately and precisely direct the motion of a missile seeker head towards a simulated target in order to replicate an engagement of that target in an actual flight. However, these tables are not limited in use to missile simulations. Three main functions this table can serve are:

1) simulating missile seeker head motions for HWIL systems
2) development and testing of guidance and navigational apparatuses
3) testing the stability and motion of satellite systems (Carter 425)

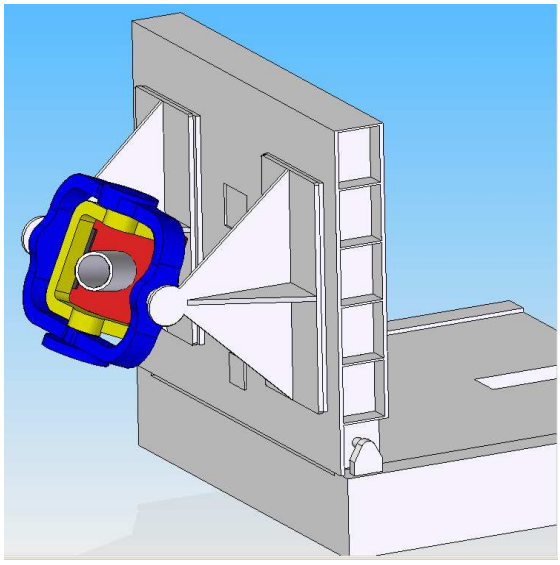
The development of more highly maneuverable missiles, tables and seeker heads adjust to meet the demands of HWIL simulations. This has led to tables capable of higher angular accelerations and angular velocities for dictating faster responding (Carter 426). The actuators being incorporated into these flight tables are required to generate enough torque to accelerate 50-100 lb. gimbals at rates of 50,000 deg/s^2. These requirements can dictate choices in the materials used to reduce vibration and deformation and also the types of actuators used to generate the needed torques (Carter 427).

Unfortunately, no matter how well the flight table and gimbaled arrangements are constructed, there will still remain some space between all bearings and connections. Seeing as how the control in each mode creates an oscillatory damping function until the error is nearly eliminated, the control will constantly be demanding reversal of torques. This reversing may lead to problems with pieces impacting each other, vibrations, and noise (Collins 579). Granted these are not short term catastrophic problems, but it may lead to wear and tear on the flight table.

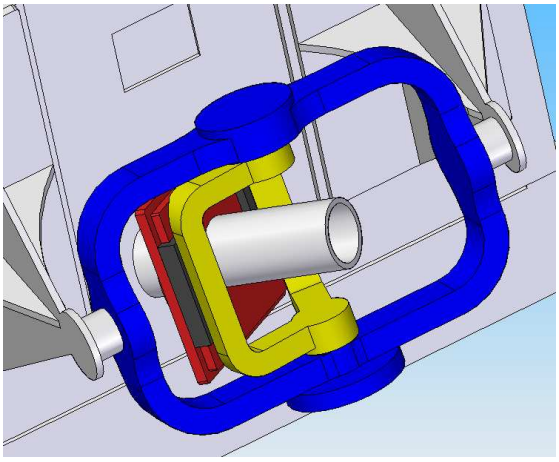
Along with the physical design of gimbaled joints and actuators, it is up to the engineer to determine a controller that will allow motion tracking with as little error as possible. Errors in the table motion refer to orientation errors: when the system's gimbaled angles do not exactly match the commanded values.

Major manufacturers of flight motion tables for HWIL laboratories include Acutronic USA, Inc. and Ideal Aerosmith. They produce tables that can be used in a pitch-yaw-roll simulation which greatly reduces the cost of developing seeker heads. The basic model of the table studied in this work is shown in Figures 2a and 2b below and is similar to the Carco Series S-450R-3 Simulator produced by Acutronic (Three). Some representative specifications of the flight table are shown in Table 1.

Figures 2a-b: Three Axis Flight Motion Table



(a)



(b)

Table 1: Performance Specifications

Performance Spec.	Type of Motion	Component and Axis Rotation		
		Component #1: Pitch	Component #2: Yaw	Component #3: Roll
Angular Freedom	-	+/- 50 deg	+/- 50 deg	+/- 50 deg
Positioning Accuracy	-	0.002 deg	0.002 deg	0.002 deg
Rate Range	Continuous	+/- 200 deg/sec	+/- 200 deg/sec	+/- 1800 deg/sec
	Non-Continuous	+/- 200 deg/sec	+/- 200 deg/sec	+/- 200 deg/sec
Acceleration, w/ load	Continuous	20,000 deg/sec^2	20,000 deg/sec^2	18,000 deg/sec^2
	Non-Continuous	20,000 deg/sec^2	20,000 deg/sec^2	20,000 deg/sec^2

III. PID CONTROLLERS

In nearly all mechanically operated systems in the industrial world, controllers are used. These controller algorithms need to be designed in order meet performance requirements while also maintaining a reasonable amount of simplicity (Gutirrez). Proportional-integral-derivative (PID) controllers are widely used to satisfy these conditions. PID or some forms of the algorithm are currently being used in approximately 95% of control loops found in modern industries (Astrom 216). The versatility of the PID-control approach is one reason PID controllers are so prevalent in modern industries.

The three terms of a PID controller (proportional, integral and derivative) all serve a specific purpose in the control algorithm. As shown in the paragraphs to come, each part determines how the system will behave. A quick overview will show that the proportional term allows the controller motion to converge to the desired response but does not eliminate the steady state error. The integral term will eliminate the steady error but can degrade the transient response. The derivative term will increase the stability of the system (PID-Tutorial). Figure 3 below is a block diagram demonstrating the basic outline of a PID algorithm. The controller signal, u, for a single-input, single-output (SISO) is:

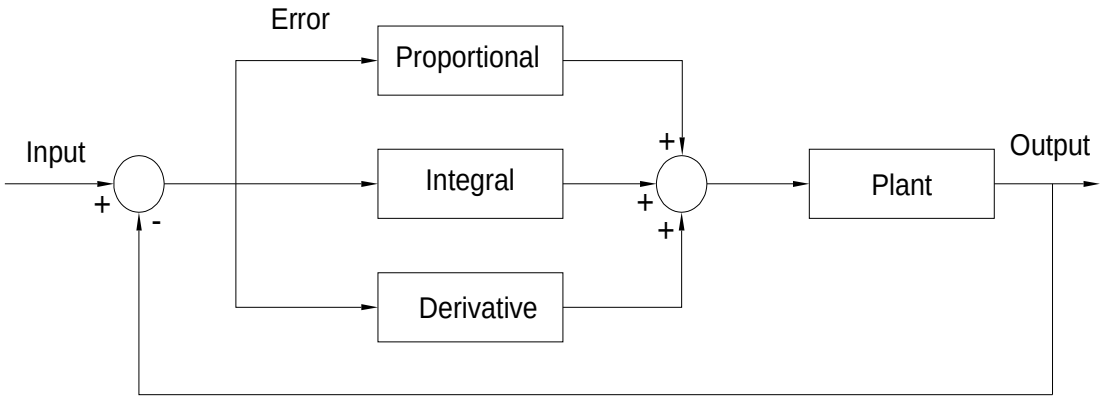
$$\boldsymbol{u} = K_P \boldsymbol{e} + K_I \int \boldsymbol{e} \mathrm{d}t + K_D \frac{\mathrm{d}\boldsymbol{e}}{\mathrm{d}t} \qquad (1)$$

where K_P, K_I and K_D are the corresponding gains for each term and e is the system error (the difference between the input and the output) at each time step (PID-Tutorial). Table 2 (reproduced from Reference [PID-Tutorial]) shows the effects of each of the three terms on a closed loop system. Each cell in the table represents a general system where there is a small increase or decrease in the gains K_P, K_I or K_D.

Table 2: Sensitivity due to Changes in Gains

CL RESPONSE	**RISE TIME**	**OVERSHOOT**	**SETTLING TIME**	**S-S ERROR**
$\boldsymbol{K_P}$	Decrease	Increase	Small Change	Decrease
$\boldsymbol{K_I}$	Decrease	Increase	Increase	Eliminate
$\boldsymbol{K_D}$	Small Change	Decrease	Decrease	Small Change

Figure 3: Block Diagram of a PID Program



Proportional Term

As mentioned before, the purpose of the proportional term is to force the system to respond in the direction of the input. The response is based purely on the error in the system. If the system is far from convergence, the error is large. Because of the direct relationship between the proportional control and the error, if the error is large then the control due to the proportional term is also large and vice versa.

The convergence rate of the system can be greatly affected by the magnitude of K_P. By increasing the value of the proportional gain, the rise time (time it takes to approach the commanded value) decreased. However, one detail to note is the overshoot (the amount the gimbaled motion exceeds the commanded motion). Although it will take the system

longer to reach the ideal state, the overshoot is smaller, which can help lead to faster convergence.

The proportional term allows for a brisk adjustment in the controlled variable. However, it does not provide "zero offset" even though it significantly reduces the error in the system. The term's primary purpose is to quicken the response (Marlin 270).

Integral Term

The primary purpose of the integral term is to eliminate or reduce the steady state error, the error between the input and output of the system as the time approaches infinity. As shown in Equation 1, the integral term is the area under the curve in an error vs. time graph. An increase in the transient error is one flaw to the integral term. With an increase in the integral gain, K_I, the rise time decreases. However, increases in overshoot and settling time (time taken for the system to converge to the commanded value) will occur. The integral term can help achieve zero offset in the system response to a step input; unfortunately, it may sometimes cause instability due to its poor dynamic performance (Marlin 271).

Derivative Term

The derivative term is the final piece in the PID controller for creating a stable system. The derivative term is proportional to the time rate of change in the error. The derivative term requires "lead", that is, information about future values of the error, allowing the controller to react faster to any changes (Gutirrez). This prediction allows the controller to converge quickly, while increasing stability without the transient error.

By increasing the derivative gain, K_D, the damping in the system can be increased. Greater damping will result in a more rigid system that is slower in convergence. A negative aspect of this term is that it may require numeric differentiation that can amplify high frequency noise in the system (Marlin 274).

IV. DYNAMIC-INVERSION CONTROLLERS

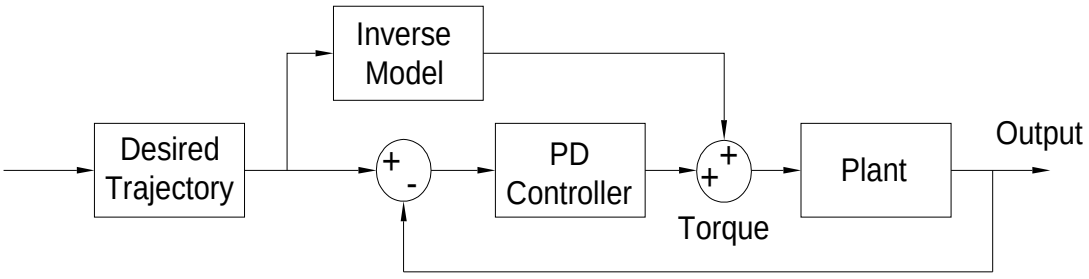
As discussed in the previous section, PID controllers are not only simple in structure, but solve a wide range of control cases with suitable results. In order to achieve good performance with the PID, the tuning of the parameters and the employment of a functional such as anti-windup and derivative filtering are vital (Visioli). There are also a few limitations to a PID controller. With a PID, it can be difficult to prove stability for nonlinear systems (such as a multi-gimbaled system on a flight table). Also, the implementation of a PID is less clear for multi-input, multi-output (MIMO) systems. An alternative control structure that may overcome these limitations is dynamic-inversion control.

According to Looye and Joos, "dynamic-inversion is a straight forward methodology for designing multi-variable control laws for nonlinear systems (1)." Dynamic-inversion methods are commonly used in aerospace applications. One such example is the development of a controller to operate in nonlinear flight schemes such as post-stall applications (Looye 1). In terms of a flight table, a dynamic-inversion controller is motivated by the multiple gimbals whose motions affect neighboring gimbals (this is shown explicitly in the derivation of the equations of motion found in Appendix A).

The defining trait of a dynamic-inversion controller is the use of a dynamic model (i.e. equations of motion) to compute the inputs necessary to generate the desired output. Hence, the name refers to the inversion of the dynamic model from the form typically used in solving for system motion. It is noteworthy that a model of the system is built into the controller, which is not the case in PID control.

A dynamic-inversion controller consists of both feedback and feed-forward sections, as shown in Figure 4. An inner-loop, structured as a closed loop, applies an inverse in dynamics in order to negate the nonlinearities in the system (Plett 360). This closed-loop system is simplified into a set of integrators to be used in the feed-forward section (Looye 1). The feedback section is the outer portion in this arrangement. The feedback loop contains a standard linear controller, such as a PID controller, in order to minimize "mismatches" and disturbances in the model created by the nonlinearities (Plett 360).

Figure 4: Block Diagram of a Dynamic-inversion Program



One key to the dynamic-inversion controller is the dynamic model located in the nonlinear feedback block. The dynamic model is a model of the input-output relationship for the system to be controlled. Plett suggests allowing the dynamic reference model is a delayed version of the actual model of the system (364). This would allow the controller to adjust *a priori* to a delayed inverse of the system dynamics (Plett 360).

There are some problems to consider when introducing a dynamic-inversion controller into a system. Looye and Joos state that dynamic-inversion tends to lead to poor robustness (1). Because the system model is built into the controllers, it is sensitive to errors in this model. The authors believe the uncertainties can be counteracted by attempting to increase the robustness of the system within the linear loop (Looye 1). Other problems created by the dynamic-inversion include the requirement that an inverse exists (non-singular) and the system typically needs *a priori* information that may need to be more precise than that available. Dynamic-inversion techniques, such as adaptive inverse control, can be applied to the arrangement (Plett 360).

V. MODEL AND CONTROLLER DEVELOPMENT

Development of Equations of Motion

To develop controller designs through numerical simulation, a model of the flight-table motion is required. A specific model is developed in this section in the form of equations of motion for a flight motion table. Key terms in these equations of motion include the mass matrix and the Coriolis vector.

The equations of motion for the flight table derived here are based on several assumptions:

(1) All rotations of coordinate axes are about a single, inertial point.

(2) Piece #1 and #3 are balanced and symmetric, therefore, the moments of inertia are centered about each component's symmetrical center of geometry.

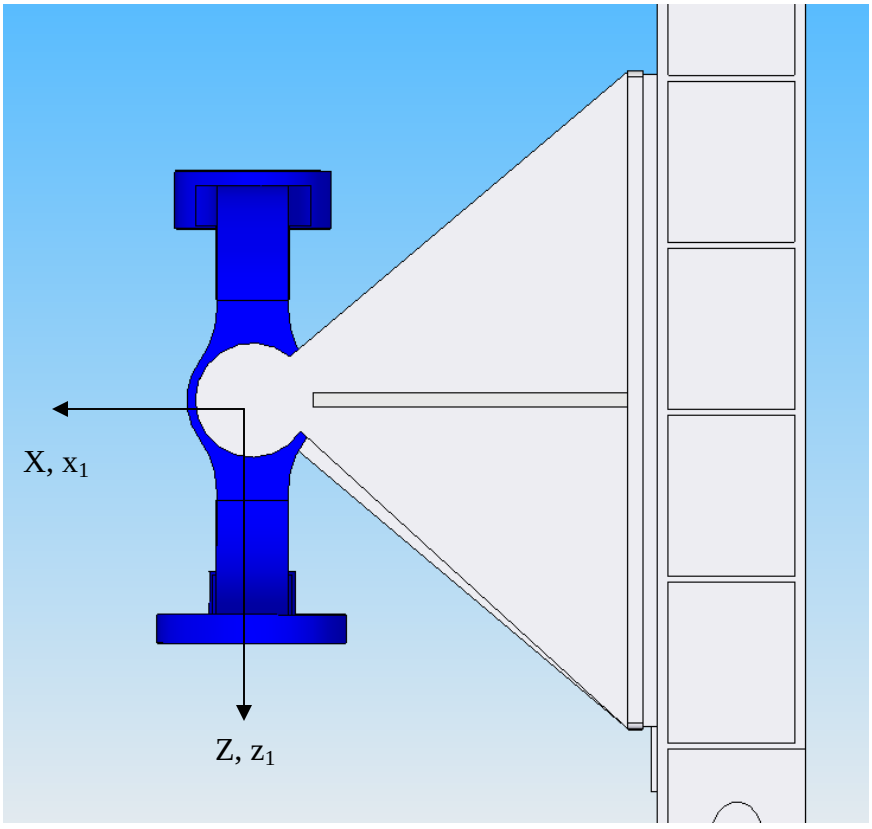
(3) Each component of the system analyzed is a rigid, non-flexible body.

(4) Friction and other applied forces in joints are negligible.

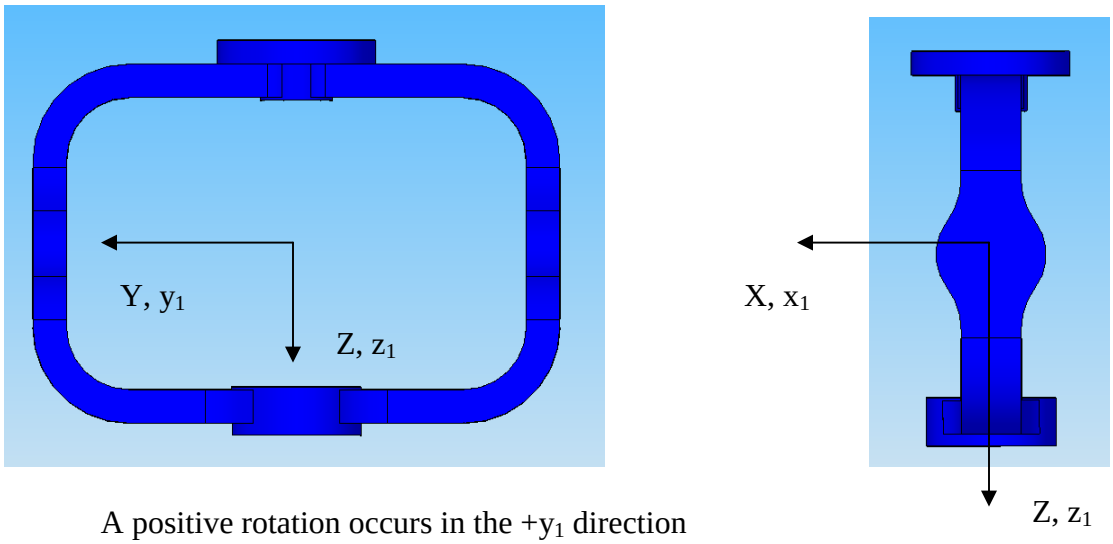
Describing the system kinematics begins with establishing an inertial coordinate system (CS) (X, Y, Z) and a separate, body-fixed coordinate system for each individual

component studied $(x_i, y_i, z_i, \text{for } i = 1,2,3)$. All inertial coordinates originate about the point of all rotations. The coordinate systems are shown in Figures 5 through 8. The figures of all the components and their set up were designed in Solid Edge. The pieces are also to scale with the flight table currently located at Auburn University. The coordinate systems in Figures 6 through 8 are all body-fixed; therefore they are attached to and rotate with the specific component.

Figure 5: Inertial CS aligned with Component #1 CS with the +Y-axis into the surface

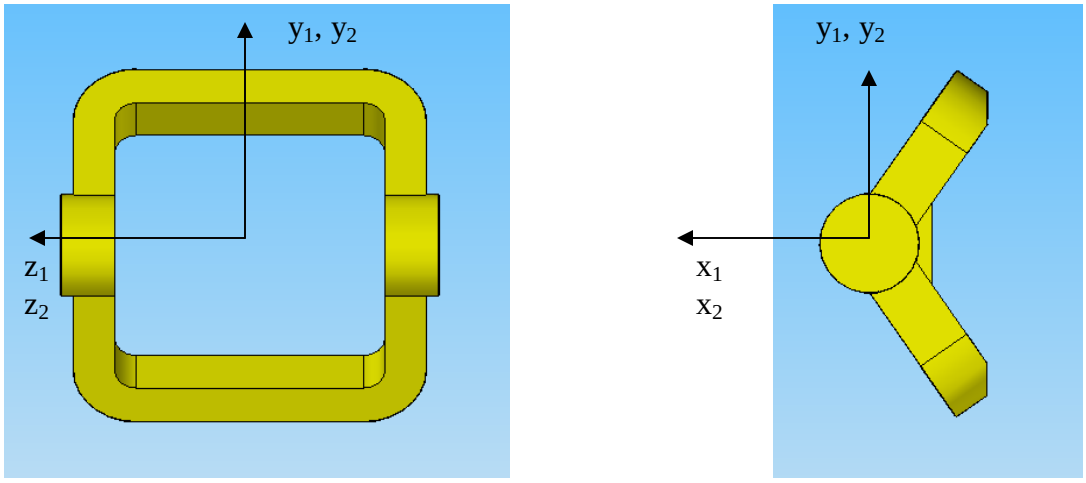


Figures 6a and 6b: Component #1 CS aligned with the Inertial CS



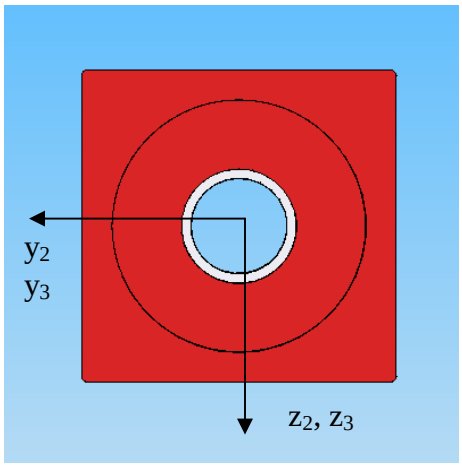
A positive rotation occurs in the $+y_1$ direction

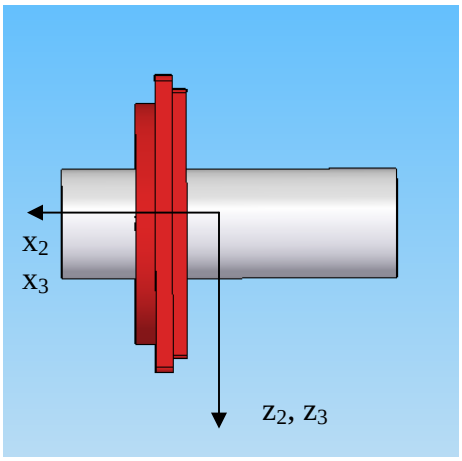
Figures 7a and 7b: Component #2 CS aligned with the Component #1 CS



A positive rotation occurs in the $+z_2$ direction

Figures 8a and 8b: Component #3 CS aligned with the Component #2 CS

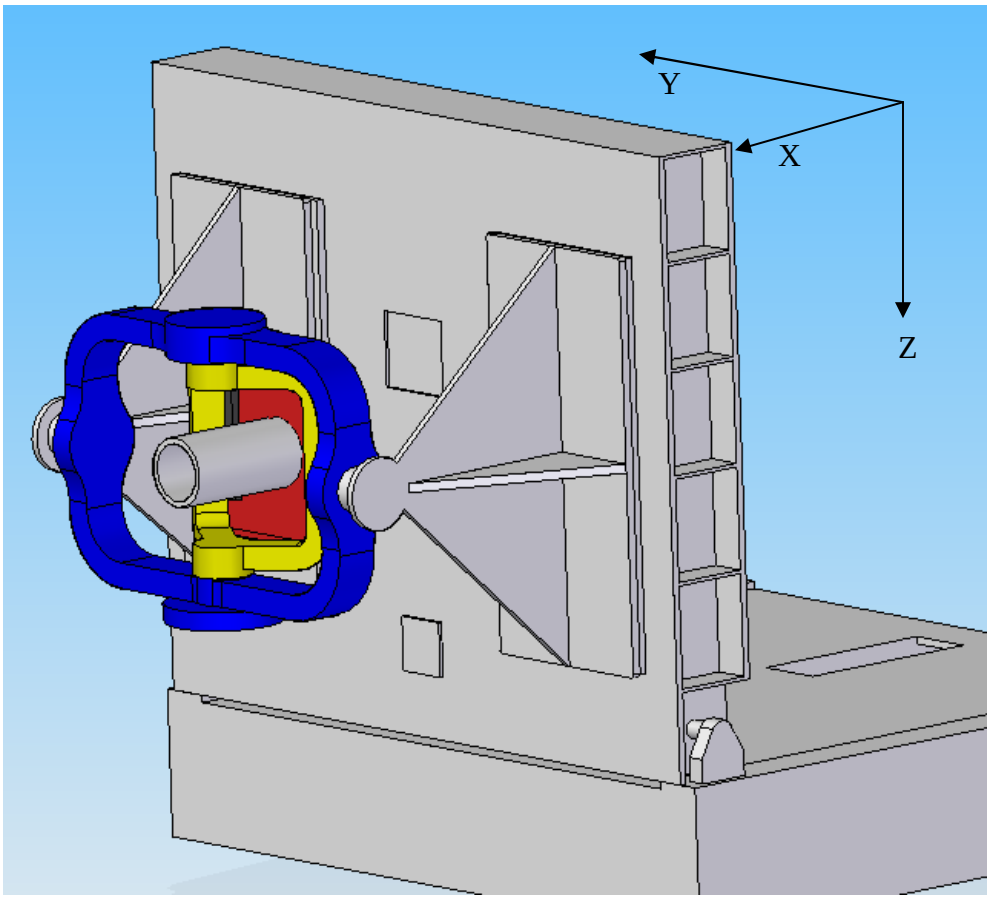




A positive rotation occurs in the $+x_3$ direction

The entire apparatus with all angles in the $\{x, y, z\}_{1,2,3}$ axes set to zero radians is shown in Figure 9. Note that in the actual experiment, this may not be the starting position of the components. This is only the reference position used in the equations of motion.

Figure 9: Gimbaled System set to Zero Deflections



To specify the orientations of the four coordinate systems, the rotation matrices were developed. A rotation matrix, in the case of a 3-dimensional system, is a 3x3 matrix that transforms the unit vector of one coordinate system into a corresponding vector in another coordinate system. In this case, the rotations are about one of the three axes. Equation 2a demonstrates the transformation from the coordinate system of component

#1 to the inertial coordinate system. Note that the rotation occurs about the $\overline{J}$ (also $\hat{j}_1$) axis, which is why a "1" is the multiplying factor in the $\overline{J}$ row.

$$\begin{Bmatrix} \overline{I} \\ \overline{J} \\ \overline{K} \end{Bmatrix} = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ 0 & 1 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 \end{bmatrix} \begin{Bmatrix} \hat{i}_1 \\ \hat{j}_1 \\ \hat{k}_1 \end{Bmatrix} \tag{2a}$$

Equations 2b and 2c are the transformation matrices for converting the coordinate system of component #2 to component #1 and the coordinate system of component #3 to component #2, respectively. In the #2 to #1 transformation, the rotation occurs about the $\hat{k}_1$ (also $\hat{k}_2$) axis. The rotation in the #3 to #2 transformation is about the $\hat{i}_2$ (also $\hat{i}_3$) axis.

$$\begin{Bmatrix} \hat{i}_1 \\ \hat{j}_1 \\ \hat{k}_1 \end{Bmatrix} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \hat{i}_2 \\ \hat{j}_2 \\ \hat{k}_2 \end{Bmatrix} \tag{2b}$$

$$\begin{Bmatrix} \hat{i}_2 \\ \hat{j}_2 \\ \hat{k}_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_3 & -\sin\theta_3 \\ 0 & \sin\theta_3 & \cos\theta_3 \end{bmatrix} \begin{Bmatrix} \hat{i}_3 \\ \hat{j}_3 \\ \hat{k}_3 \end{Bmatrix} \tag{2c}$$

Angular velocities of the coordinate systems can be obtained by inspection. A listing of the transformed angular velocities can be found in Appendix A.

The next step in deriving the equations of motion is to determine the energy in the rotational system. In his treatise the *Mécanique analytique*, Lagrange demonstrated laws of virtual work, which could be applied to the mechanics of both solids and fluids (Joseph). Rather than follow the work of D'Alembert and Euler by tracking the complete motion of a particle, he showed that "if we determine its configuration by a sufficient number of variables whose number is the same as that of the degrees of freedom possessed by the system, then the kinetic and potential energies of the system can be expressed in terms of those variables, and the differential equations of motion thence deduced by simple differentiation (Joseph)." The form of this equation is:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)-\frac{\partial T}{\partial \boldsymbol{q}}+\frac{\partial V}{\partial \boldsymbol{q}}=\boldsymbol{f} \tag{3}$$

where $\boldsymbol{q}=[\theta_1 \quad \theta_2 \quad \theta_3]^T$ and $\dot{\boldsymbol{q}}=[\dot{\theta}_1 \quad \dot{\theta}_2 \quad \dot{\theta}_3]^T$ are the generalized coordinates and generalized velocities, T is the kinetic energy of the system defined as the addition of the rotational kinetic energy of each component, V is the total potential energy, and $\boldsymbol{f}$ is the generalized external forces on the system. The energies are shown in Equation 4a-b as:

$$V = mg\left(a\sin\beta + L\cos\delta\left(1-\cos\theta_2\right)\right)$$

m = mass of component #2

L = distance of CG offset in component #2

where $a = L\sqrt{2\left(1-\cos\delta\right)}$ (4a)

$$\beta = \frac{\pi}{4} - \frac{1}{2}\theta_1$$

$$\delta = \frac{\pi}{2} - \theta_1$$

$$T = \frac{1}{2}\sum_{i=1}^{3} \boldsymbol{\omega}_i^T \boldsymbol{I}_i \boldsymbol{\omega}_i \quad \text{(4b)}$$

For component #2, estimations were made for the distance of the center of gravity offset.

In Equation 4b, ***I*** indicates the inertia of each body relative to the fixed center of rotation for the system. By substituting Equation 4 into Equation 3, one can obtain equations of motion that allow for the development of the mass matrix and the nonlinear Coriolis vector terms. The complete organization of these matrices and vectors can be found in Appendix A.

A unique, assumed characteristic of the flight motion table system considered here is that all parts of the system rotate about a single point. Also, it should be noted that component #3 is merely a hollowed out cylinder that can be rotated about its longitudinal axis. Therefore, it can be assumed that I_{3y} and I_{3z} are equivalent. (The actual code used in numerical simulation is written for the possible case of $I_{3y} \neq I_{3z}$ for future experiments). Equations 5 and 6 are the resultant mass matrix and nonlinear Coriolis

vector for the assumption of $I_{3y} = I_{3z}$. The moments of inertia and mass elements were calculated based on the geometry of the modeled part. The pieces were modeled as prisms, assumed to be solid casts. Table 3 is a representation of the moments of inertia for each component.

Table 3: Moments of Inertia for 3-Gimbaled System

Body Axis of Rotation	Component and Axis Rotation		
	Component #1: Pitch (234.9 kg)	Component #2: Yaw (202.2 kg)	Component #3: Roll (152.8 kg)
x	57.6 kg-m^2	11.9 kg-m^2	2.8 kg-m^2
y	20.4 kg-m^2	10.7 kg-m^2	6.8 kg-m^2
z	37.8 kg-m^2	9.9 kg-m^2	6.8 kg-m^2

The resulting equations of motion have the form $M\ddot{q} + h = f$. The terms $M\ddot{q}$ and h have the following form when $I_{3y} = I_{3z}$.

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} = \begin{bmatrix} I_{1y} + (I_{2x} + I_{3x})\sin^2\theta_2 + (I_{2y} + I_{3y})\cos^2\theta_3 & 0 & I_{3x}\sin\theta_2 \\ 0 & I_{2z} + I_{3y} & 0 \\ I_{3x}\sin\theta_2 & 0 & I_{3x} \end{bmatrix} \begin{Bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{Bmatrix}, \tag{5}$$

$$\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{Bmatrix} \dot{\theta}_2 \cos\theta_2 \left\{ 2\dot{\theta}_1 \left(I_{2x} - I_{2y} + I_{3x} - I_{3y} \right) \sin\theta_2 + \dot{\theta}_3 I_{3x} \right\} + \dfrac{\partial V}{\partial \theta_1} \\ -\dot{\theta}_1 \cos\theta_2 \left\{ \dot{\theta}_1 \left(I_{2x} - I_{2y} + I_{3x} - I_{3y} \right) \sin\theta_2 + \dot{\theta}_3 I_{3x} \right\} + \dfrac{\partial V}{\partial \theta_2} \\ \dot{\theta}_1 \dot{\theta}_2 I_{3x} \cos\theta_2 \end{Bmatrix} \qquad (6)$$

The generalized forces of the system are the three control torques exerted by the hydraulic/electric actuators of the flight table.

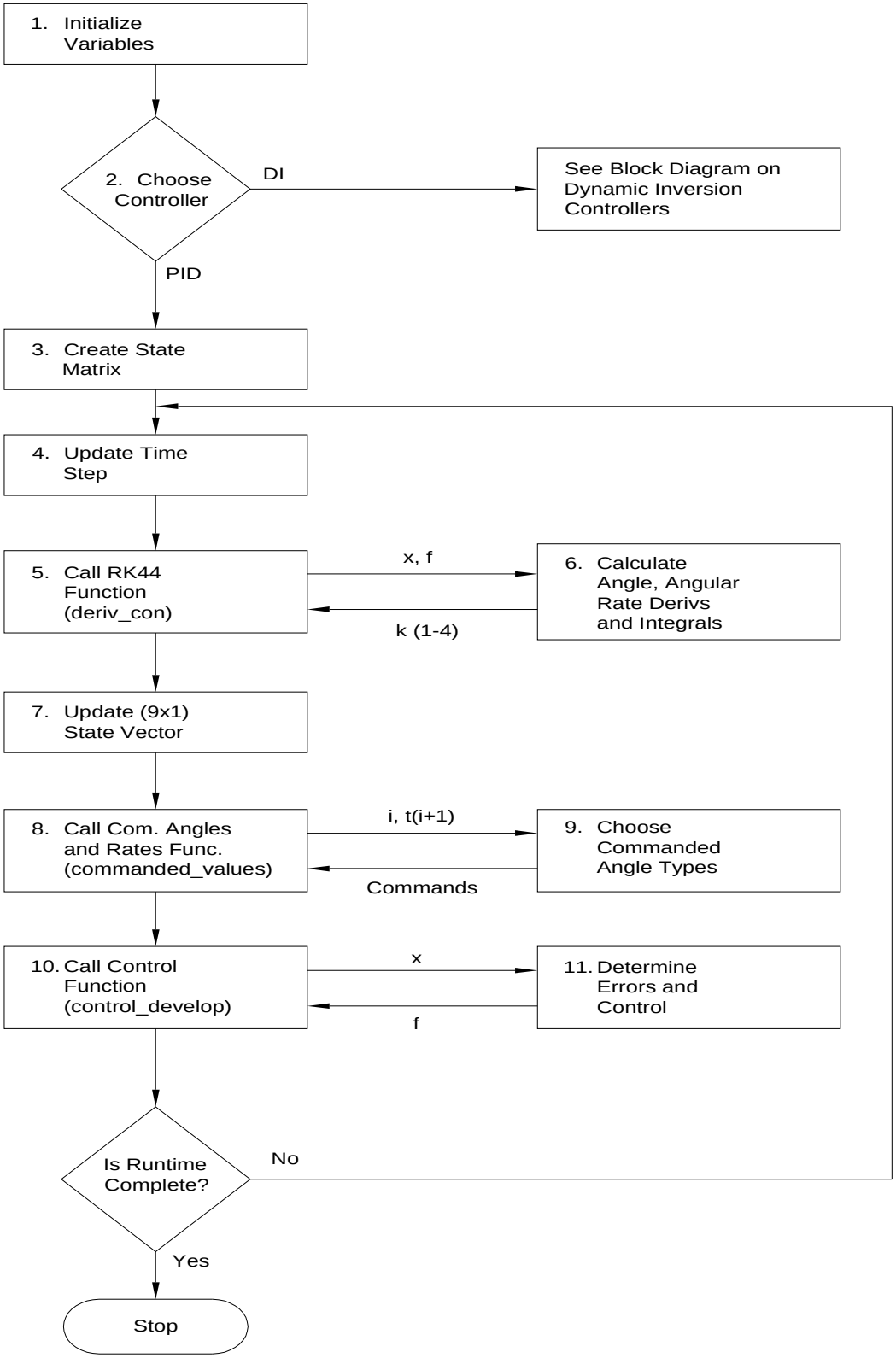
$$\boldsymbol{f} = \begin{Bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \end{Bmatrix} \qquad (7)$$

In operation, the torques are the control variables used to force the system motion to track the commanded motion.

PID Program Structure

The block diagram shown in Figure 10 demonstrates the structure of the Proportional-Integral-Derivative program. The following is an explanation of each block in the figure. The explanation includes the parameters involved, the objective of the structure and the output from the block.

Figure 10: PID Program Structure



1. Initialize Variables

Besides variables based on common knowledge, such as $g = 9.81\frac{m}{s^2}$, the section takes into account the initialization of the angles of each component on the fight table and the angular velocity of each. The physical geometry of each rotating piece, such as moments of inertia, distance of offset from the rotation point (used for potential energy) and the mass of the offset components are defined. The three basic variables involved with time are defined. The total time is the duration the simulation is allowed to run, the time step for each integration step, and the number of iterations allowed before a control adjustment are defined. For instance, if the system runs for 10 seconds, and each time step is 0.01 seconds, 1000 steps are analyzed. If a control is adjusted every 2 time steps, the system is running at 500 Hz. Finally, the initialization contains as experimental variables the gains for the proportional, integral and derivative terms.

2. Choose Desired Controller

This part of the program allows the user to select which type of controller they would like to examine; a PID or dynamic-inversion controller. Once

the user selects a controller, all the characteristics of that controller are applied for the remainder of the program. The controller not chosen is negated for the remainder of the runtime. In this chapter, the PID controller was selected.

3. Create State Vector

The state vector groups together all the components of the simulation that will be integrated across time. In the case of the PID controller, the state vector is a (9 x 1) vector consisting of angular positions for the 3 rotating components, the three angular rates associated with those components and the three errors associated with the integral gain in the controller itself. Grouping all nine components together assures that each cell is updated simultaneously with the other eight cells.

4. Update the System Time

The arrangement in question is a discrete time system. Therefore, the values at a specific time step are determined and then evaluated at a small forward step in the system time, Δt. When the words “updating the system time” are used, the program is not automatically changing all the times

associated with calculated variables to some new time, $t + \Delta t$, but it's storing the old and new variables which can then be used for comparison and error calculations.

5. Call Proportional Integral Derivative RK-44 Function (derive_con)

The PID program was written to take small time steps, integrate them across those time steps and update the state vector consisting of angles, angular rates, and integral error, all using a fourth-order Runge-Kutta integration method. The RK-44 uses the beginning, middle, and end point of each time step in order to calculate an average time rate of change of each parameter over that discrete time. The following equations represent the RK-44 for a second-order system:

$$\ddot{\boldsymbol{q}} = f(\boldsymbol{q}, \dot{\boldsymbol{q}})$$
$$\boldsymbol{k}_1 = \Delta t\left[f\left(\boldsymbol{x}_t(\boldsymbol{q}, \dot{\boldsymbol{q}}), \boldsymbol{u}_t\right)\right]$$
$$\boldsymbol{k}_2 = \Delta t\left[f\left(\boldsymbol{x}_t(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \frac{1}{2}\boldsymbol{k}_1, \boldsymbol{u}_t \right)\right]$$
$$\boldsymbol{k}_3 = \Delta t\left[f\left(\boldsymbol{x}_t(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \frac{1}{2}\boldsymbol{k}_2, \boldsymbol{u}_t \right)\right]$$
$$\boldsymbol{k}_4 = \Delta t\left[f\left(\boldsymbol{x}_t(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{k}_3, \boldsymbol{u}_t\right)\right]$$

The k-terms sent back into the "deriv_con" function, which represents the RK-44, are (9 x 1) vectors as well. Each cell in the vector corresponds to

its respective angle, angular rate or integral error which will be used to update the state vector, as shown in section 7.

6. Calculate Angle, Angular Rate Derivatives, and Error Integral

The function, "deriv_con", serves as the integration section of the RK-44. With the inputs of each portion of the time step, as shown in part 5, the function calculates the k and z (integral) terms. The k terms are also referred to as the $\dot{\theta}$ and $\ddot{\theta}$ terms. The equations of motion (derived in the upcoming sections and Appendix A):

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{f} \tag{8}$$

Provide the second time derivatives of the Euler angles.

$$\frac{\mathrm{d}\dot{\boldsymbol{q}}}{\mathrm{d}t} = \ddot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})^{-1}\left(\boldsymbol{f} - \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}})\right) \tag{9}$$

Although the mass matrix, *M*, is diagonally dominant and invertible, the "\" function was used in Matlab. This applies a Gaussian Elimination method for finding a matrix equation of the form $Ax = b$, where *A* is (nxn) and *x* and *b* are (nx1). The integral for each portion of the time step is also

calculated by taking difference between the commanded angles and the actual angles. The returned vector is a (9 x 1) vector in the form:

$$\boldsymbol{k}_{1-4} = \left[\dot{\boldsymbol{q}}_{(1x3)} \quad \ddot{\boldsymbol{q}}_{(1x3)} \quad \dot{\boldsymbol{z}}_{(1x3)}\right]^T$$

Note that all the components are the derivatives, or slope, of its integrated part. This vector is returned to the main program in order to update the state vector.

7. Update State Vector

Once all the derivatives of the angles, angular rates and integral errors are sent back to the main program, the last piece of the 4th Order Runge-Kutta is applied. This part of the RK-44 updates the angles, angular rates and error integral across the discrete time step using the equation:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{1}{6}\left(\boldsymbol{k}_1 + 2\left(\boldsymbol{k}_2 + \boldsymbol{k}_3\right) + \boldsymbol{k}_4\right) \qquad (10)$$

The updated state matrix will be used for calculating the errors present and the required controls for the next iteration of the main program.

8. Call the Commanded Angles and Rates Function (commanded_values)

This section of the program loop is one of the key factors for designing the experiment. The commanded angles, angular rates and angular accelerations are very similar to the simulated path of a target that the seeker is trying to follow. This portion of the program relays the position, speed, and the rate of change in speed the gimbals should be achieving. These commanded angles and rates are calculated in the "commanded_values" function. The variables sent to this function are the time step iteration number and the updated time of the system.

9. Determine Control Angle and Rate Types

The simulated motion can take many shapes and forms. The three types of simulations that are going to be examined are the constant direction and angular velocity, a constant position and angular velocity followed up with an impulsive change in position at a given time step, and a constantly changing (in this case sinusoidal) position and velocity. The constant position and angular velocity scenario takes a random gimbaled placement and tracks the motion of the positions of the interceptor and target as they maneuver to another position over the course of the simulation. The position and angular velocity with an impulsive change tracks the position

as it settles, instantaneously changes to another commanded position, and tracks the gimbals' motion for the duration of the simulation time. The sinusoidal position and velocity applies a commanded sine form of position, velocity and acceleration that the gimbals must track.

The main program will receive a (9 x 1) vector, represented as:

$$\boldsymbol{y}_{command} = [\boldsymbol{q}_{com} \quad \dot{\boldsymbol{q}}_{com} \quad \ddot{\boldsymbol{q}}_{com}]^T$$

The three scenarios are further described and demonstrated in the *Commanded Variables and Description of Functions* section.

10. Call the Control Function (control_develop)

Now stored in the program is the updated system time, a corresponding state vector and a commanded position and rate vector. However, as mentioned before, the values of these two vectors may not agree, creating some error in the results. To minimize this error, a control is applied to the necessary gimbals where it is required. This control is determined in the function "control_develop". The required inputs for this function are both the state vector and commanded angles and rates vector. The

function will calculate the control required for the next iteration of the main program.

11. Determine the Three PID Errors and Controls

As discussed in previous chapters, there are three types of errors to analyze in a PID controller system; the proportional, integral and derivative. Having a commanded angle and an angular rate relayed to the function, the position and speed each component should calculated. Also, after completion of the RK-44 and an update of the variables, an actual set of angles and angular rates are determined. Unless everything in the system was 100% perfect, there is bound to be some error in the nonlinear arrangement. Therefore, it is necessary to find the errors for each section of the PID controllers. The error used in the proportional controller is simply the difference between the commanded and actual position of each component. The integral controller error is represented by the "z" term found from integrating the difference between the commanded and actual angle of each component from the beginning to the current time of the simulation. The error used in the derivative portion of the controller is difference between the commanded and actual angular velocities of each component.

As discussed before, the basic equation used in the majority of PID controllers is represented by Equation 1:

$$\boldsymbol{u} = K_P \boldsymbol{e} + K_I \int \boldsymbol{e} \mathrm{d}t + K_D \frac{\mathrm{d}\boldsymbol{e}}{\mathrm{d}t}$$

Note that in the previous section, the three errors defined correspond, respectively, to the errors in the equation. The values of the gains have already been defined in the initialization of all parameters section. These gains can be adjusted in order permit the system to more precisely converge to the commanded angles and angular velocities. The value of the control, *u*, a (3 x 1) vector in this case, is then updated and applied back into Equation 9, until further updated. The control, *u*, is then relayed back to the main program.

12. Is Runtime Complete?

No matter how long the system runs, there is always going to be some oscillation error due to fact that a small time segment is used, rather than a continuous signal. However, after a certain time, the error in the system is considered negligible. The runtime value, defined in the initialization section, is used by the programmer to ensure the program will terminate

after a specific simulation time. Once a discrete value of Δt is added to the system time, the program checks to see if the termination time has been achieved. If so, the program terminates. If the time has not been reached, the program loops back to Section 2, and repeats all the steps leading up to Section 12.

Derivation of Dynamic-inversion Controller

When deriving the dynamic-inversion controller for a motion flight table, the problem is classified in the category of tracking controllers for nonlinear natural mechanical systems. Remember that a natural mechanical system is one where all the terms in the kinetic energy are quadratic in the generalized velocities, $\dot{\boldsymbol{q}}$.

Because of this property of natural mechanical systems, the kinetic energy of the system can be written as:

$$T(\boldsymbol{q},\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} = \frac{1}{2} M_{ij}\dot{q}_i\dot{q}_j \qquad (11)$$

where T is the kinetic energy and $\boldsymbol{M}$ is the mass matrix. In order to determine the equations of motion, as required for the dynamics of the flight table, Lagrange's equation

(Eq. 3) needs to be applied. This is done most conveniently in index notation. Expansion of Equation 8 leads to the equations of motion, represented as:

$$M_{ki}\ddot{q}_i + \frac{\partial M_{ki}}{\partial q_j}\dot{q}_j\dot{q}_i - \frac{1}{2}\frac{\partial M_{ij}}{\partial q_k}\dot{q}_i\dot{q}_j + \frac{\partial V}{\partial q_k} = f_k \tag{12a}$$

or

$$M_{ki}\ddot{q}_i + h_k = f_k \tag{12b}$$

$$\text{where } h_k = \left(\frac{\partial M_{ki}}{\partial q_j} - \frac{1}{2}\frac{\partial M_{ij}}{\partial q_k}\right)\dot{q}_i\dot{q}_j + \frac{\partial V}{\partial q_k} \tag{13}$$

In vector notation, Equation 12 can be represented as:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{f} \tag{14}$$

Based on this form of the equations of motion, a general form of a dynamic-inversion controller will be developed. The control law for the generalized forces, *f*, in design is desired to track the motion of the commanded inputs, $\boldsymbol{q}_d$ and $\dot{\boldsymbol{q}}_d$. For this, a Lyapunov function will be selected.

$$Q = \frac{1}{2}(\boldsymbol{q}-\boldsymbol{q}_d)^T(\boldsymbol{q}-\boldsymbol{q}_d) + \frac{1}{2}(\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_d)^T(\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_d) \tag{15}$$

A check point is to be sure $Q = 0$, when the actual motion identical to the commanded motion. Otherwise, $Q > 0$. Taking the derivative of *Q*, one gets:

$$\dot{Q} = (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d)^T (\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_d + \boldsymbol{q} - \boldsymbol{q}_d) \tag{16}$$

Here, $\ddot{\boldsymbol{q}}_d$ are the accelerations associated with the commanded motion. The condition of a global asymptotically stable system can be achieved by choosing:

$$\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_d + \boldsymbol{q} - \boldsymbol{q}_d = -(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) \tag{17}$$

The commanded motion can be substituted into the equations of motion to define $\boldsymbol{f}_d$, the generalized forces necessary to produce this motion.

$$\ddot{\boldsymbol{q}}_d = \boldsymbol{M}(\boldsymbol{q}_d)^{-1}(\boldsymbol{f}_d - \boldsymbol{h}(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d)) \tag{18}$$

A globally asymptotically stable controller used to track a desired motion for the natural mechanical system may be found by solving Equation 14 for $\ddot{\boldsymbol{q}}$ and substituting along with Equation 18 back into Equation 17.

$$\begin{aligned} \boldsymbol{f} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q}_d)^{-1} \boldsymbol{f}_d + \left(\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{M}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q}_d)^{-1}\boldsymbol{h}(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d)\right) \\ - \boldsymbol{M}(\boldsymbol{q})[(\boldsymbol{q} - \boldsymbol{q}_d) + (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d)] \end{aligned} \tag{19}$$

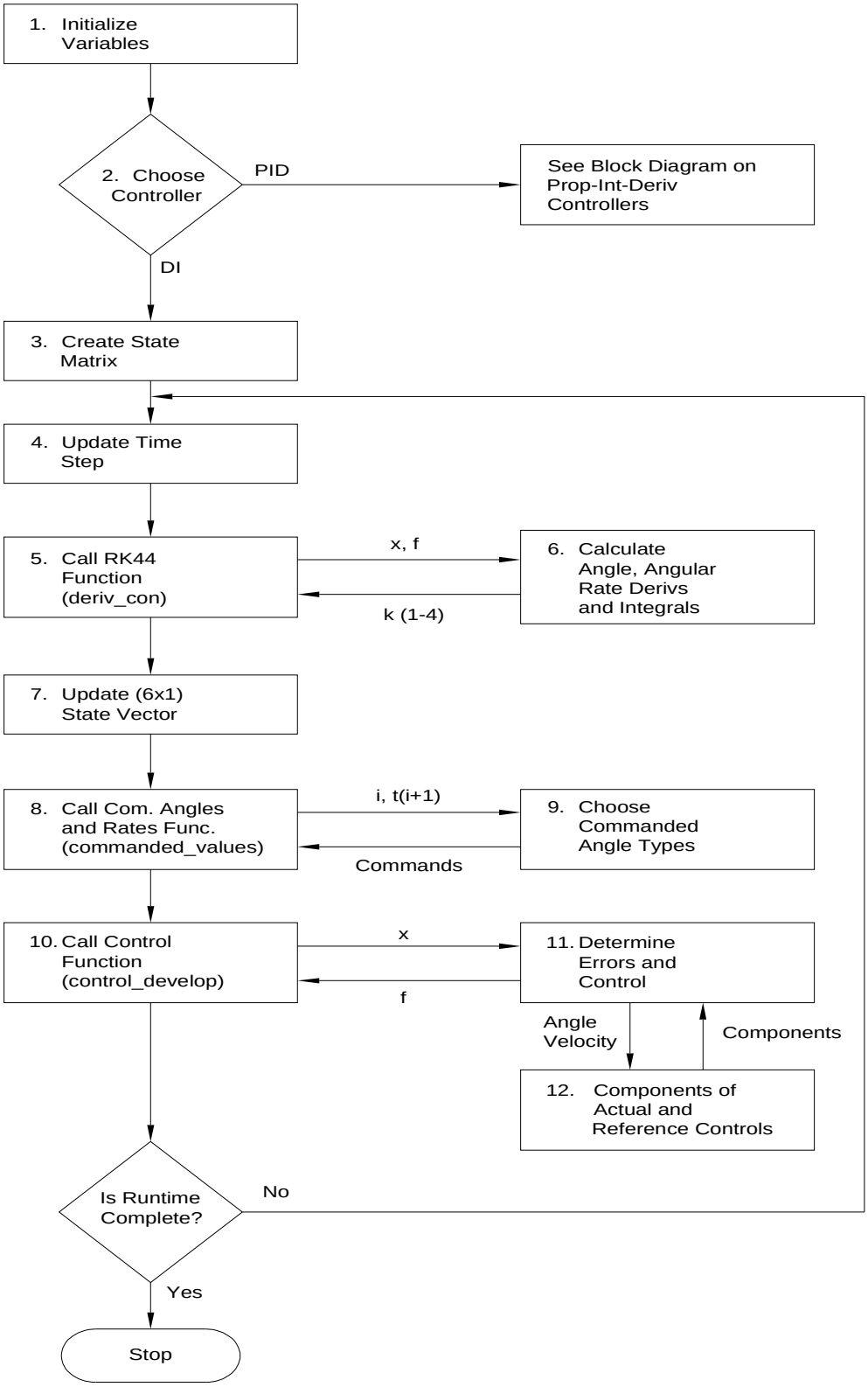
The performance of this nonlinear controller will be compared with a typical linear PID controller. A more detailed derivation can be found in Appendix B. From Equation 19, it can be seen that the feedback terms in this controller are a form of proportional-

derivative (PD) control. The PD controller can alleviate unmodeled dynamics and disturbances (Yan 199). Therefore, in comparison with the PID controller laid out in the previous section, the dynamic-inversion controller does not require error integral information. This is, however, a result of the choice of the Lyapunov function. The dynamic-inversion controller does require knowledge of commanded accelerations (in order to compute $\boldsymbol{f}_d$), which were not required by the PID controller. In implementation some means of computing $\ddot{\boldsymbol{q}}_d$ is necessary.

Dynamic-inversion Program Structure

The block diagram shown in Figure 11 demonstrates the structure of the dynamic-inversion program. The following is an explanation of each block in the figure. The explanation includes the parameters involved, the objective of the structure and the output from the block. Many of these sections are very similar to those in the PID description, mainly because the dynamic-inversion controller is the only difference in the system. A positive attribute to this program is the flexibility of examining whichever controller the user prefers. Therefore, the majority of the steps and processes are mirrored for each controller.

Figure 11: Dynamic-inversion Program Structure



1. Initialize Variables

To compare the PID arrangement to that of the dynamic-inversion in a fair manner, the same initial variables must be used. Therefore, the moments of inertia, mass of each component, distance of component offset, initial positions and angular velocities, and time step variables (including the frequency of control adjustments) must remain equal to those in the PID experiment. The only difference lies in the gains used in the dynamic-inversion. Only proportional and derivative gains are used and their values will be different following proper tuning to compliment the arrangement.

2. Choose Desired Controller

As mentioned before, the user has the ability to select which type of controller they chose to examine; a PID or dynamic-inversion controller. Once the user selects a controller, all the characteristics of that controller are applied for the remainder of the program. In this chapter, the dynamic-inversion controller was selected. Therefore, the PID controller characteristics and processes are negated for the remainder of the runtime.

3. Create the State Vector

The state vector combines all the components that will be integrated across the simulation time. In the case of the dynamic-inversion controller, the state vector is a (6 x 1) vector consisting of angular positions for the three rotating components and the three angular rates associated with those components: $\boldsymbol{x} = [\boldsymbol{q} \quad \dot{\boldsymbol{q}}]^T$. Unlike the PID state vector, there is no integral error present due to the form of the controller. The error across this controller is not integrated; therefore, the gain is not necessary. This set-up assures that all six variables will be updated and applied simultaneously throughout the simulation.

4. Update the System Time

The process of updating the system time by a discrete time step, Δt, serves the same purpose as the PID controller. This update is merely a process of storing and comparing the values of the state vector, command vector, and control vector for any given time step throughout the simulation. It allows the user to specify which value at a specific instant they would prefer to examine.

5. Call Dynamic-inversion RK-44 Function (deriv_con)

Similar to the PID program, the dynamic-inversion is analyzed over constant time steps of Δt, therefore integration across those discrete times is necessary. Once again, a fourth-order Runge-Kutta numerical integration is going to be applied to the system. All equations are going to be the same, with the exception of the k terms. Before, in the PID program, each returned value of k contained three angular derivatives, three angular velocity derivatives, and three error integrations. The error integrations are not necessary; therefore, the k values are a (6 x 1) vector, rather than a (9 x 1). The called function, "deriv_con", is the same function used for PID controller, only modified to cater to both controllers.

6. Calculate Angle and Angular Rate Derivatives

The function, "deriv_con", is the exact same function used for the PID controller, except in this situation, the calculation of the integral error over the specific time step is not applied. Both functions serve the purpose of determining the rate change in the angular velocity, as previously shown in Equation 9:

$$\frac{\mathrm{d}\dot{\boldsymbol{q}}}{\mathrm{d}t} = \ddot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})^{-1}(\boldsymbol{f} - \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}))$$

The resulting $\dot{q}$ and $\ddot{q}$ terms are returned to the main program as a (6 x 1) vector in the form of:

$$\boldsymbol{k}_{1-4} = \left[\dot{\boldsymbol{q}}_{(1x3)} \quad \ddot{\boldsymbol{q}}_{(1x3)}\right]^T$$

This vector is returned to the main program in order to update the state vector by completing the RK-44 integration.

7. Update the State Vector

Once all of the angular rates and angular accelerations are returned to the main program, the last function of the fourth-order Runge-Kutta is performed. This part of the RK-44 updates the angles and angular velocities across the discrete time step, Δt, using the same equation as that of the PID controller:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{1}{6}(\boldsymbol{k}_1 + 2(\boldsymbol{k}_2 + \boldsymbol{k}_3) + \boldsymbol{k}_4)$$

The updated state matrix will be used for calculating the errors in the commanded and actual angles and the required controls for the next iteration of the main program.

8. Call the Commanded Angles and Rates Function (commanded_values)

The commanded angles, angular rates, and angular accelerations are the motion that the controller is trying to track. This portion of the program relays the position, the speed, and the rate of change in speed the gimbals should be achieving. For comparison, the commanded values calculated in this section will be the same as those used in the PID experiments. These commanded angles and rates are calculated in the "commanded_values" function. The variables sent to this function are the time step iteration number and the updated time of the system.

9. Determine Control Angle and Rate Types

Three basic types of simulations were chosen to study the performance of the dynamic-inversion controller: (1) constant direction and angular velocity, (2) constant position and angular velocity followed up with an impulsive change in position at a given time step, and (3) constantly

changing (in this case sinusoidal) position and velocity. The same commanded angles and angular velocities will be used in the PID program to ensure an accurate comparison in the results of both controllers. The returned variable is a (9 x 1) vector, represented as:

$$\boldsymbol{y}_{command} = [\boldsymbol{q}_{com} \quad \dot{\boldsymbol{q}}_{com} \quad \ddot{\boldsymbol{q}}_{com}]^T$$

The three scenarios are further described and demonstrated in the *Commanded Variables and Description of Functions* section.

10. Call the Control Function (control_develop)

The variables required to calculate the updated errors and control for the t_{i+1} conditions are now saved into the state vector and command vector. Assuming the vectors do not agree, whether machine error or an error caused by a change in the commanded angles and velocity, the "control_develop" function allows for a correction control to be produced. The required inputs for this function are both the state vector and commanded angles and rates vector. The function will calculate the control required for the next iteration of the main program.

11. Call Commanded and Actual Components Function (model_components)

One of the unique characteristics of a dynamic-inversion controller is the comparison between the actual components of the equations of motion and some reference components based off commanded values in order to determine the control applied to the next time interval. The reference value components consist of the angles, angular rates, mass matrix, Coriolis terms and control. These terms are calculated in the function, "model_components" and returned to "control_develop" function for analysis. Once the terms are returned, they are applied in the control equation for this particular dynamic-inversion controller which will be derived in the following section:

$$\begin{aligned} \boldsymbol{f} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q}_d)^{-1}\,\boldsymbol{f}_d + \left(\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{M}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q}_d)^{-1}\boldsymbol{h}(\boldsymbol{q}_d,\dot{\boldsymbol{q}}_d)\right) \\ - \boldsymbol{M}(\boldsymbol{q})\left[(\boldsymbol{q}-\boldsymbol{q}_d)+(\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_d)\right] \end{aligned} \qquad (19)$$

The first part of the Equation [19] represents the feedback portion of the controller. The second condition of the equation is a nonlinear feedback term comparing the difference between the Coriolis terms of the reference and actual motions. The last term is a PD form of a controller that is a nonlinear feedback as well.

All the terms in Equation 19 are generated in “model_components”, with the exception of the commanded and actual angles and angular rates. The inputs for this function are the actual and commanded angles and angular velocities. Once the new control is calculated, it is returned to the main program in order to be applied to the next iteration, starting at Section 4.

12. Determine Commanded and Actual Value Components

The function, “model_components”, for the reference portion uses Equation 5 to determine the mass matrix, Equation 6 to determine the Coriolis terms, and Equation 8 to determine the reference control. All these values are in correspondence with the simulated motion of the system.

The actual value components consist of the angles, angular rates, mass matrix, and Coriolis terms. These values are determined in a similar fashion to the reference components, using the same order and equations, with the exception that the returned variables represent the actual characteristics of the system. The applied control is not calculated until the five characteristics of the motion at that particular time step are returned to the “control_develop” function.

13. Is Runtime Complete?

Just like in the PID controller, there is always going to be some sort of small error in the simulation, where after a certain time, the error in the system is negligible (if properly designed). Once a discrete value of Δt is added to the system time, the program checks to see if the termination time has been achieved, ceasing to program if it has. If the time has not been reached, the program loops back to Section 2, and repeats all the steps leading up to Section 13, only to be tested again.

VI. MODEL SIMULATIONS AND ANALYSIS

General Experimental Set-up

The tri-gimbaled system has three basic parameters which need to be considered when designing an accurate and precise response to a given command. The factors which will be examined are the type of controller, the type of commanded angles, angular velocities and angular accelerations, and the gain levels used in the simulation. As discussed in previous chapters, the controllers analyzed are of type PID and dynamic-inversion, and the commanded positions and rates are constant, steps and sinusoidal. The experiments will also examine how the system changes with increases in both the proportional and derivative gains.

Commanded Variables and Description of Functions

The commanded variables consist of constant variables, alternating variables (a step function), and a constantly adjusting sinusoidal command.

Prior to every simulation, some factors of the gimbals are set. All three gimbals are initially set to +0.4 radians from their "home" position. This offset will require an initial maneuver to be performed in order to approach the commanded position. The gimbals will all start from rest.

The controller must now track one of the following commanded parameters:

1. Constant Commanded Angles and Rates

   With all three gimbaled positions set to +0.4, an alternate set of gimbaled positions is commanded to force the controller to adjust all three controls simultaneously. The values for these gimbals after the commanded controls begin are:

$$\boldsymbol{\theta}_{com} = \begin{bmatrix} 0.1 & 0.1 & 0.2 \end{bmatrix}^T$$
$$\dot{\boldsymbol{\theta}}_{com} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$
$$\ddot{\boldsymbol{\theta}}_{com} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$

2. Constant Commanded Angles and Rates with a Step Function

   This commanded function is very similar to the constant commanded positioning function described in the previous section with one basic

difference. The commanded function travels along a constant angle for all components then impulsively maneuvers to a different angular position. For the experiments, the positions are:

$$t = 0 \rightarrow 5\,\text{sec} : \boldsymbol{q}_{com} = [0.1 \quad 0.1 \quad 0.2]^T$$
$$t = 5 \rightarrow 10\,\text{sec} : \boldsymbol{q}_{com} = [0.2 \quad 0.2 \quad 0.5]^T$$

Because this is a discrete time system and the time step is small, the angular velocity and angular acceleration expressions remain the same.

3. Consistent Sinusoidal Commanded Path

For this commanded motion, the angles of all three components are all going to be based off a sinusoidal wave with amplitude of 0.5 radians and frequency of $1/2\pi$ Hz. Because these angular positions are based on time, the angular velocities and accelerations can be represented as the first and second derivative of the position vector.

$$\boldsymbol{q}_{com} = [0.5\sin t \quad 0.5\sin t \quad 0.5\sin t]^T$$
$$\dot{\boldsymbol{q}}_{com} = \frac{\mathrm{d}\boldsymbol{q}_{com}}{\mathrm{d}t} = [0.5\cos t \quad 0.5\cos t \quad 0.5\cos t]^T$$
$$\ddot{\boldsymbol{q}}_{com} = \frac{\mathrm{d}^2\boldsymbol{q}_{com}}{\mathrm{d}t^2} = [-0.5\sin t \quad -0.5\sin t \quad -0.5\sin t]^T$$

This method of determining the rates is valid as long as the commanded position of each component is a function of time.

Gain Selections

It is typically assumed that the larger the gain in a system, the less robust (more sensitive to errors in the dynamic model) the system becomes. However, the experiments performed are not to find the optimal gains in the simulation; only to deduce the effect that a change in gains has on each controller.

After numerous trials of various gains, two sets of gains were selected as appropriate for the experiments. The first set was $K_P = 200$ and $K_D = 50$. This set of gains allows the motion and response to be more flexible. The second set is a more aggressive set with the gains set to $K_P = 400$ and $K_D = 100$. For each experiment run with the PID controller, the K_I gain was set equal to the K_P gain. The dynamic-inversion controller does not utilize the integral gain.

These two sets of gains were compared alongside the three commanded motion and the two different types of controllers, producing twelve sets of results.

Model Simulations

Three commanded motion, two sets of controller gains, and two different controllers are to be examined. Adjusting every parameter to perform a single study in order to find out the characteristics and effects of that parameter leads to twelve different situations to consider. This section will break those twelve simulations into three groups of four tests. Those three groups will be run with the types of commanded motion. Within each group, the changes in both the gains and controller will be observed through the angular motions of the gimbals. An analysis of the applied controls, errors, and run times will be addressed in later sections.

Input 1: Constant Commanded Angles and Rates

The first simulation tested will compare the PID and dynamic-inversion controllers with a change in gains. These will be conducted with the commanded angles, angular velocities and angular accelerations held constant throughout a 10 second test period. Figures 12a and 12b represent a PID and dynamic-inversion controller under the influence of a proportional gain equal to 200 and a derivative gain equal to 50.

After the initial jump from the +0.4 radian position, all components controlled through the PID show difficulties in convergence. The pitch component (piece

#1) shows the greatest error in convergence. However, the dynamic-inversion controller seems to converge in nearly every piece before one second has elapsed.

Figures 12c and 12d are for the same situation, except the gains have increased significantly. Note that the PID system still has convergence problems, but seems more in control due to the gain increase. The dynamic-inversion does not seem to have difficulties with the larger gains. The convergence time in the dynamic-inversion controllers remained about the same. With the larger gains applied, the PID controller began to respond similarly to the dynamic-inversion controller.

Figure 12: a) PID Model for Constant Control and $K_P = 200$, $K_D = 50$

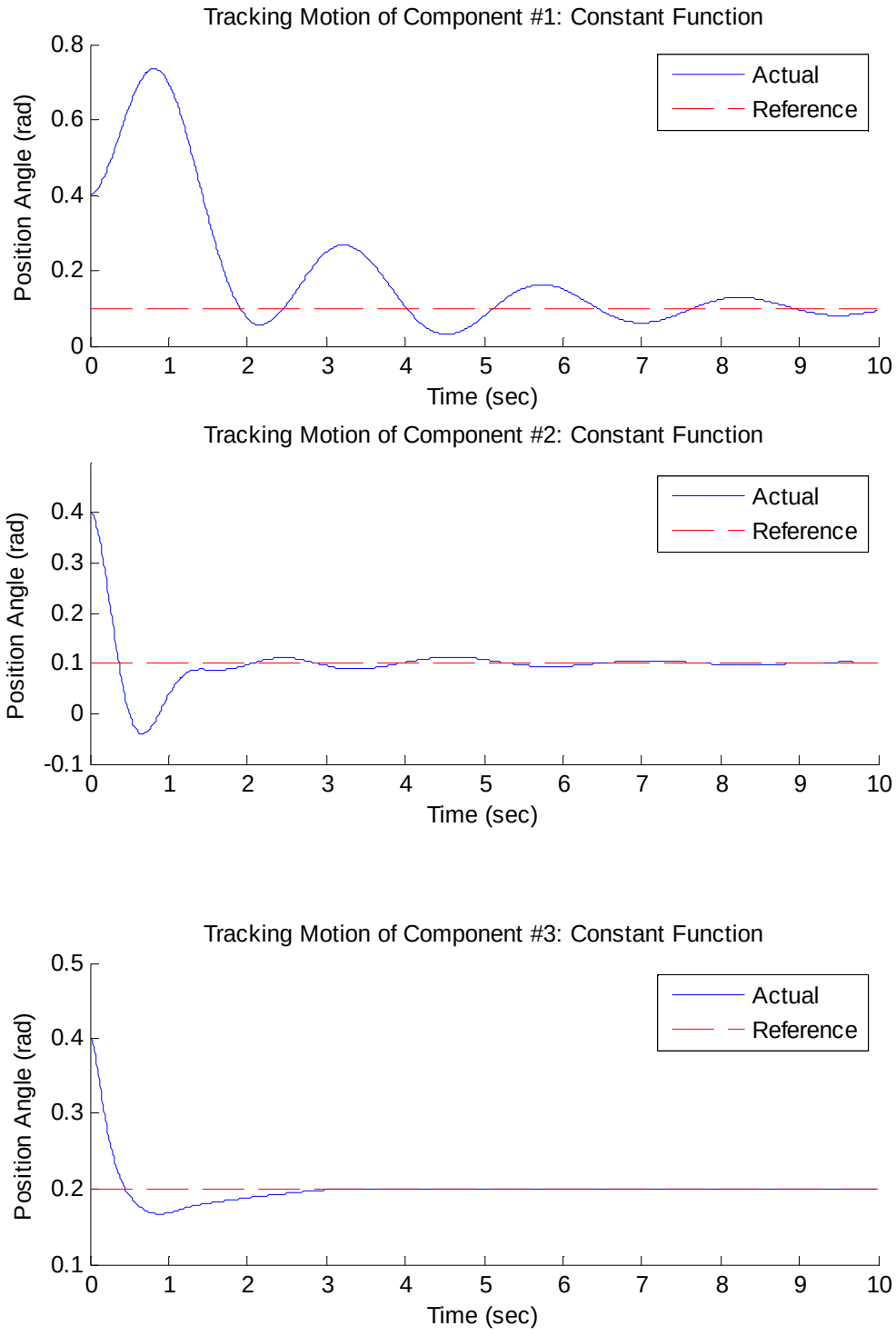


Figure 12: b) DI Model for Constant Control and $K_P = 200$, $K_D = 50$

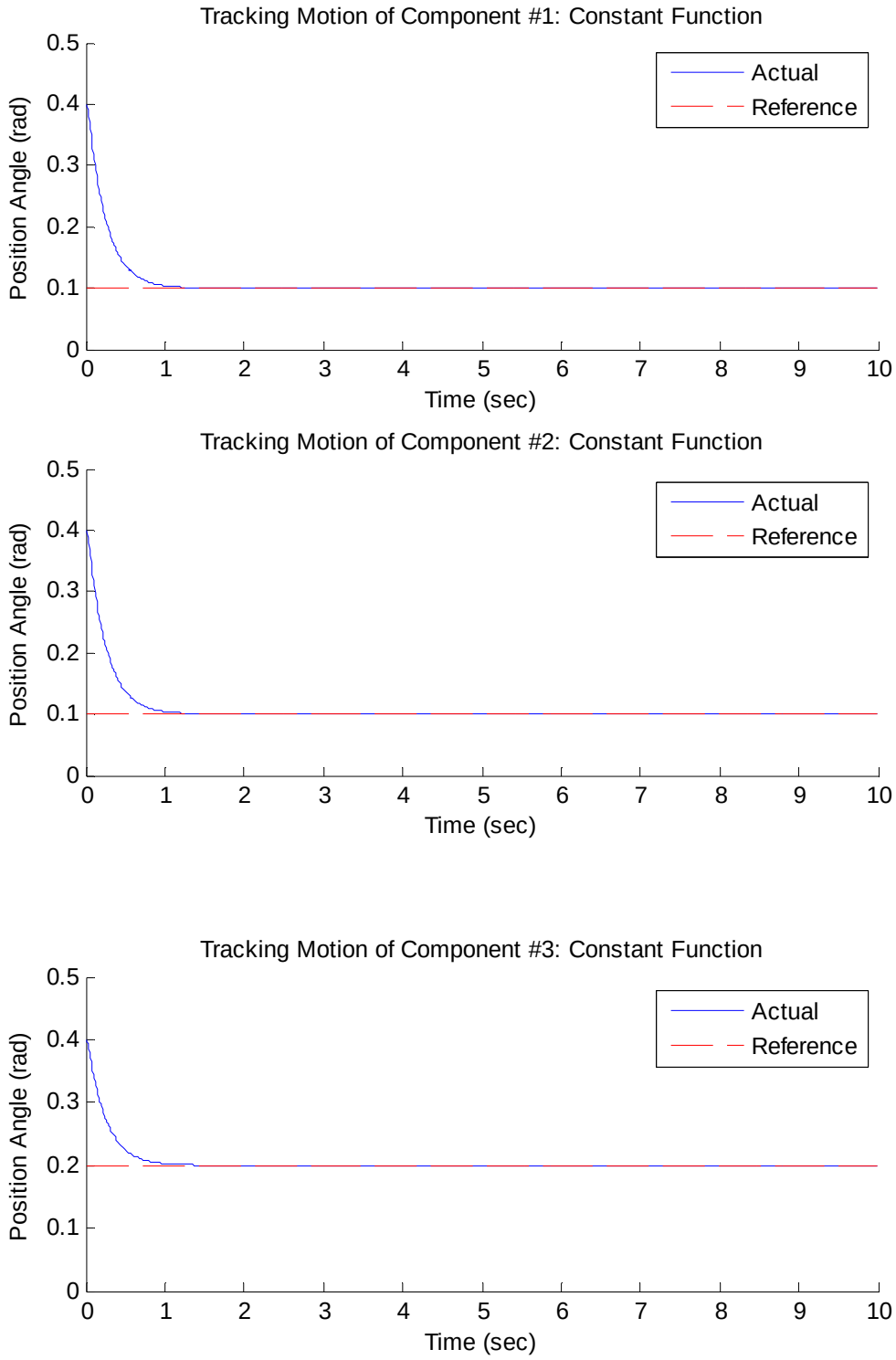
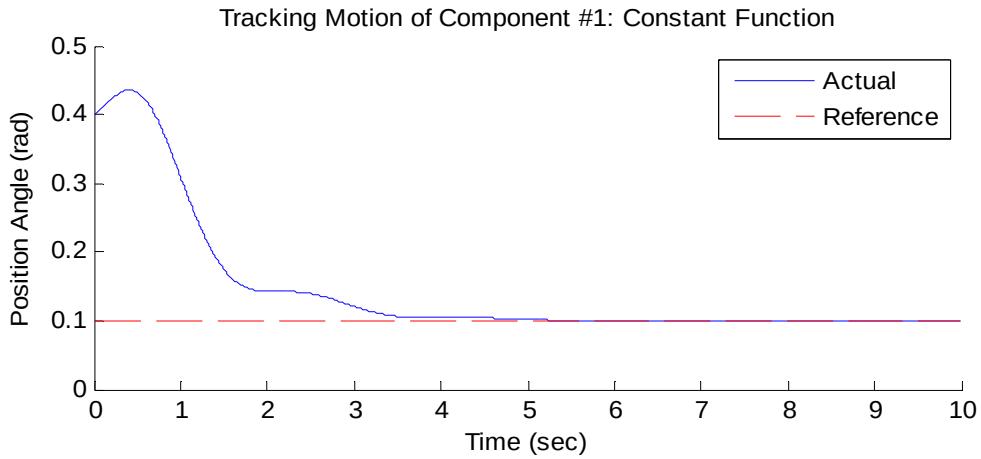
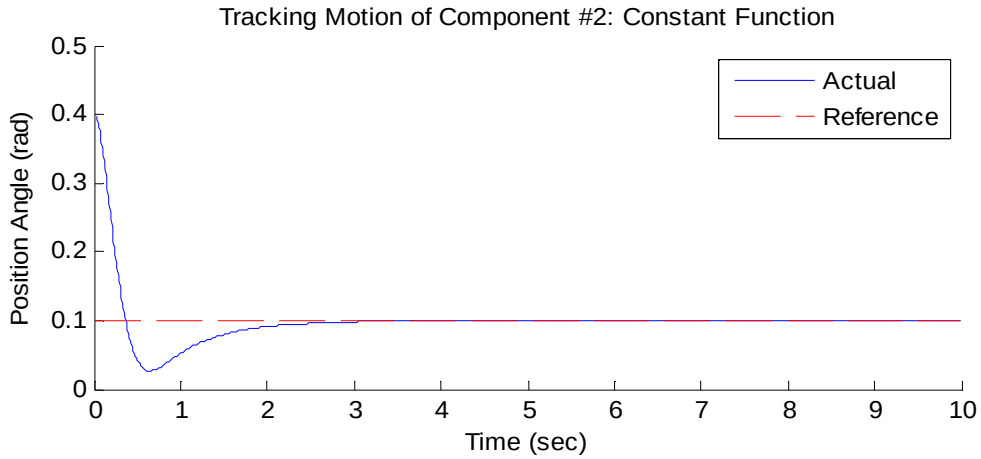


Figure 12: c) PID Model for Constant Control and K_P = 400, K_D = 100





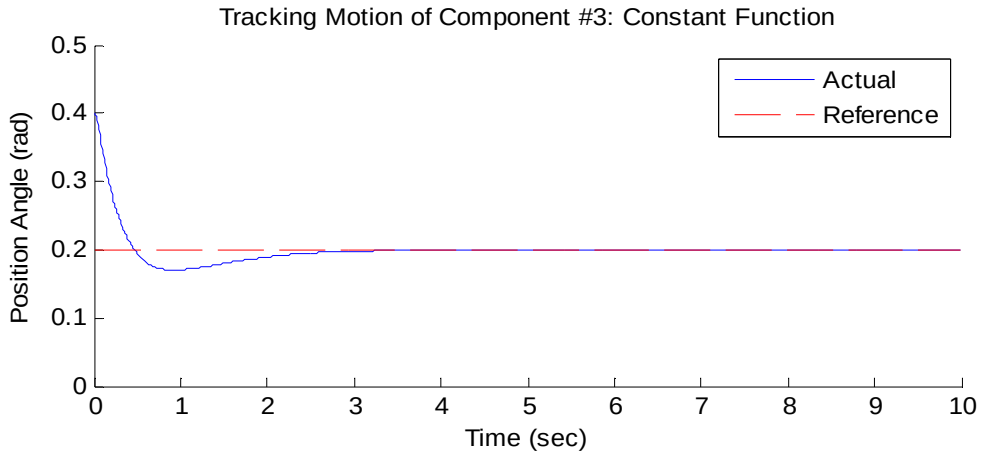
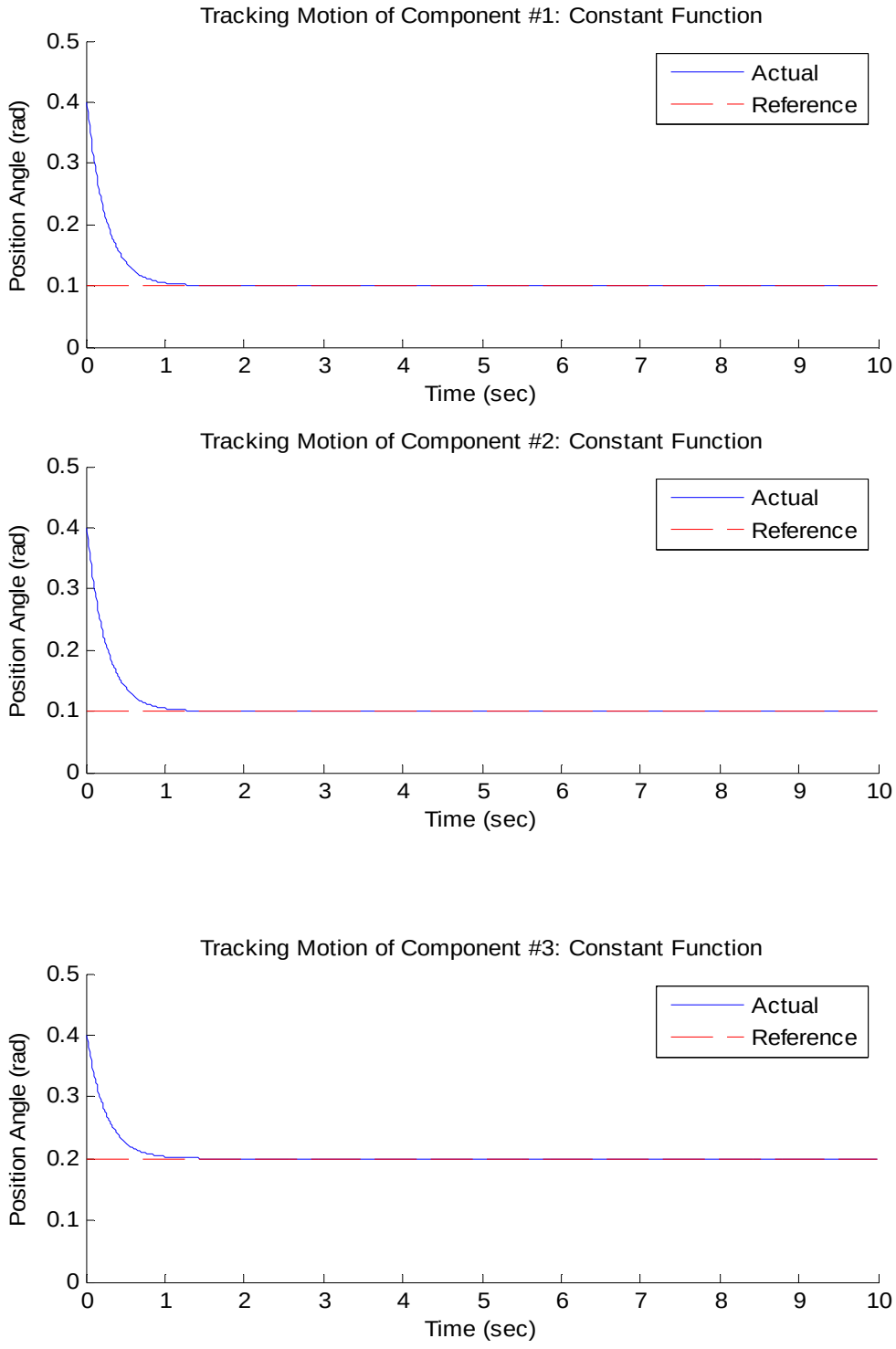


Figure 12: d) DI Model for Constant Control and K_P = 400, K_D = 100



Input 2: Constant Commanded Angles and Rates with a Step Function

The procedure of testing for this mode is nearly identical to that of Input 1 with the exception that a step function was applied at the 5 second mark. However, from 0 to 5 seconds and 5 to 10 seconds, the commanded angles and rates remain constant.

Figures 13a and 13b represent a PID and dynamic-inversion controller with a proportional gain equal to 200 and a derivative gain equal to 50. Figures 13c and 13d represent the same except that more aggressive K_P and K_D gains of 400 and 100 respectively were applied.

Similar to the previous mode, the dynamic-inversion has quicker reaction and convergence times than the PID controller. In fact, the PID controller only seems to be efficient when the gains are high. However, even after the gains are increased, there is still a significant amount of overshoot in the PID. The dynamic-inversion controller contains little to no overshoot and dampens out quickly in both cases.

Figure 13: a) PID Model for Step Control and K_P = 200, K_D = 50

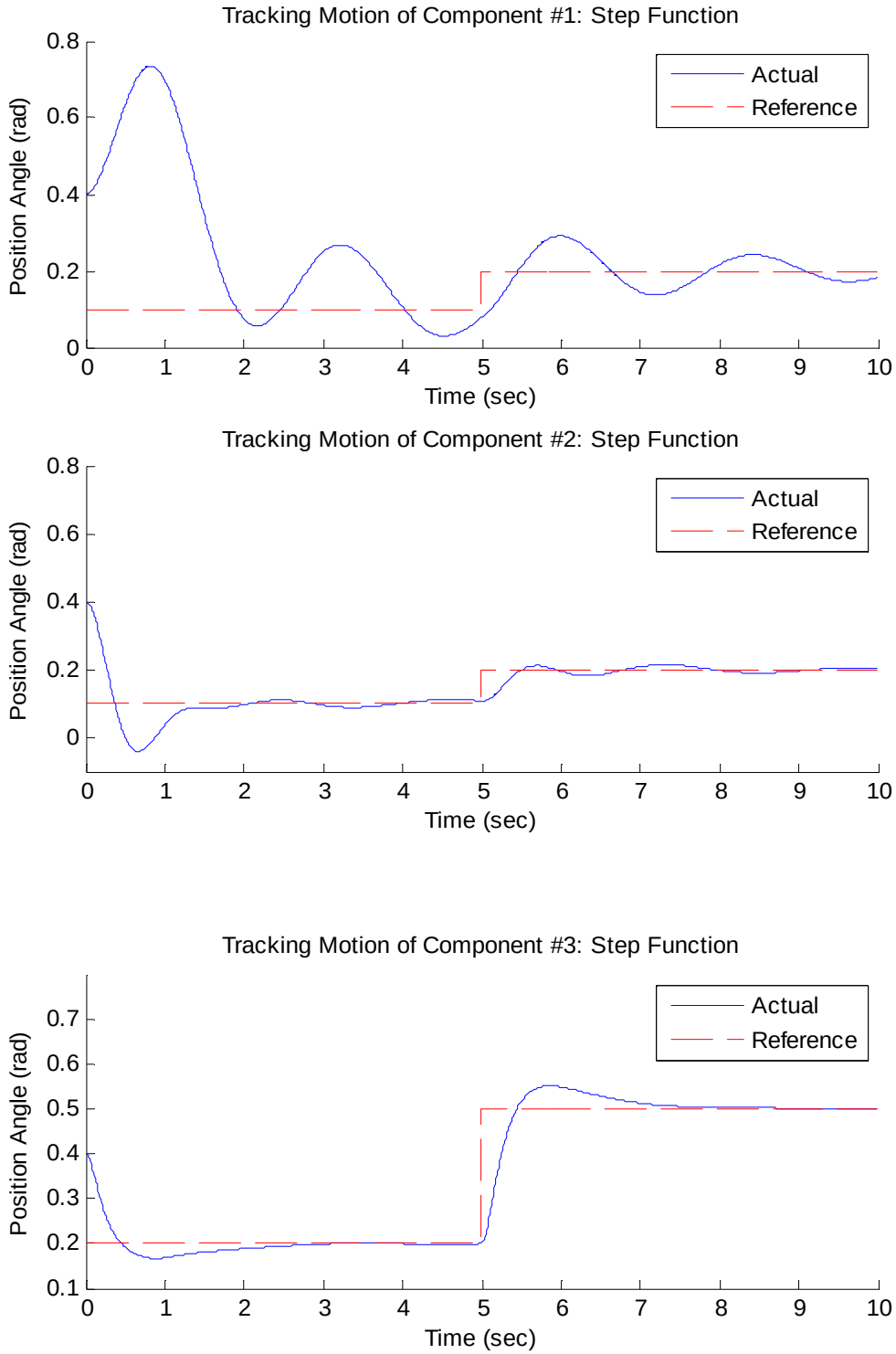


Figure 13: b) DI Model for Step Control and $K_P = 200$, $K_D = 50$

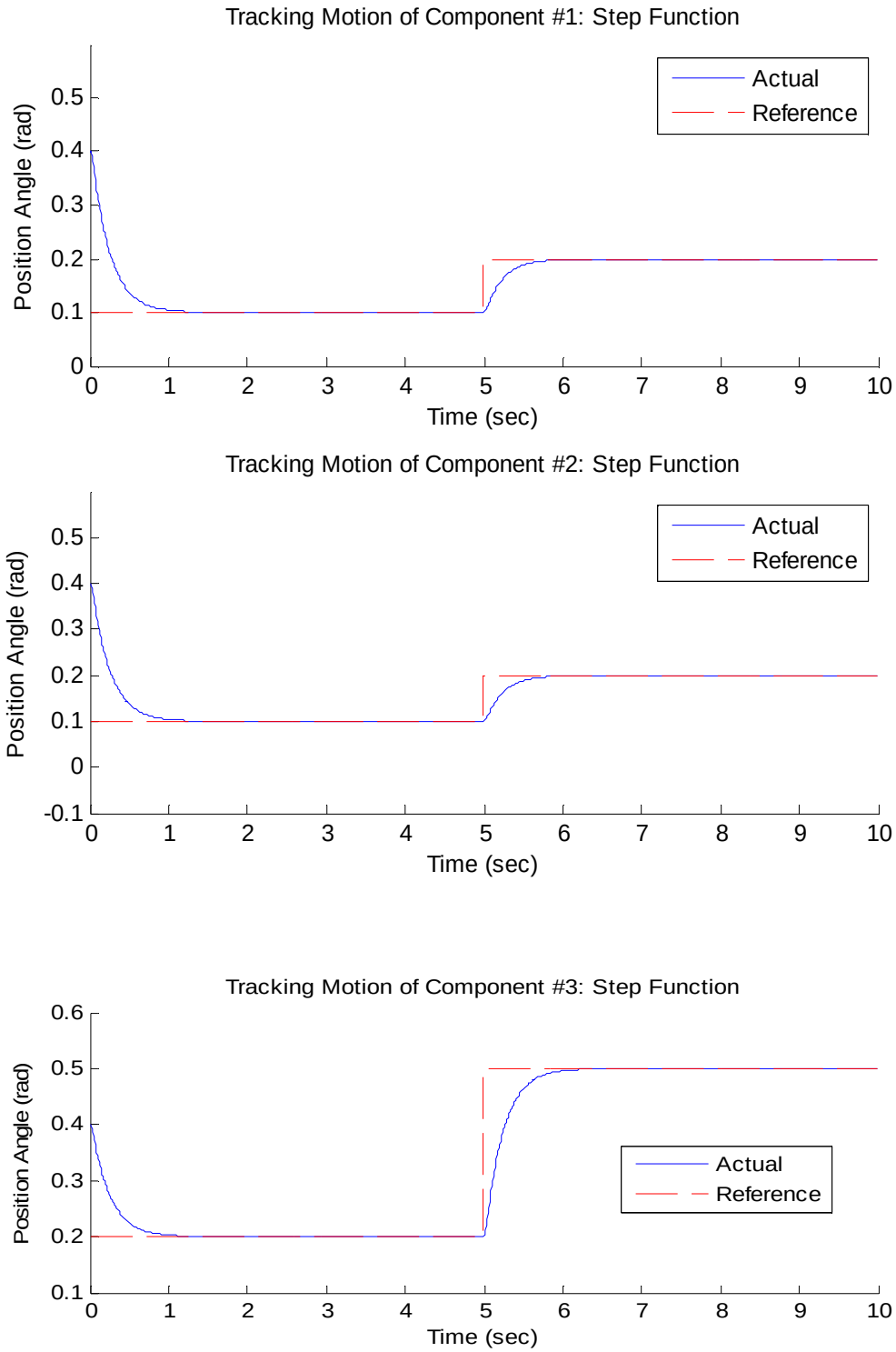


Figure 13: c) PID Model for $K_P = 400$, $K_D = 100$

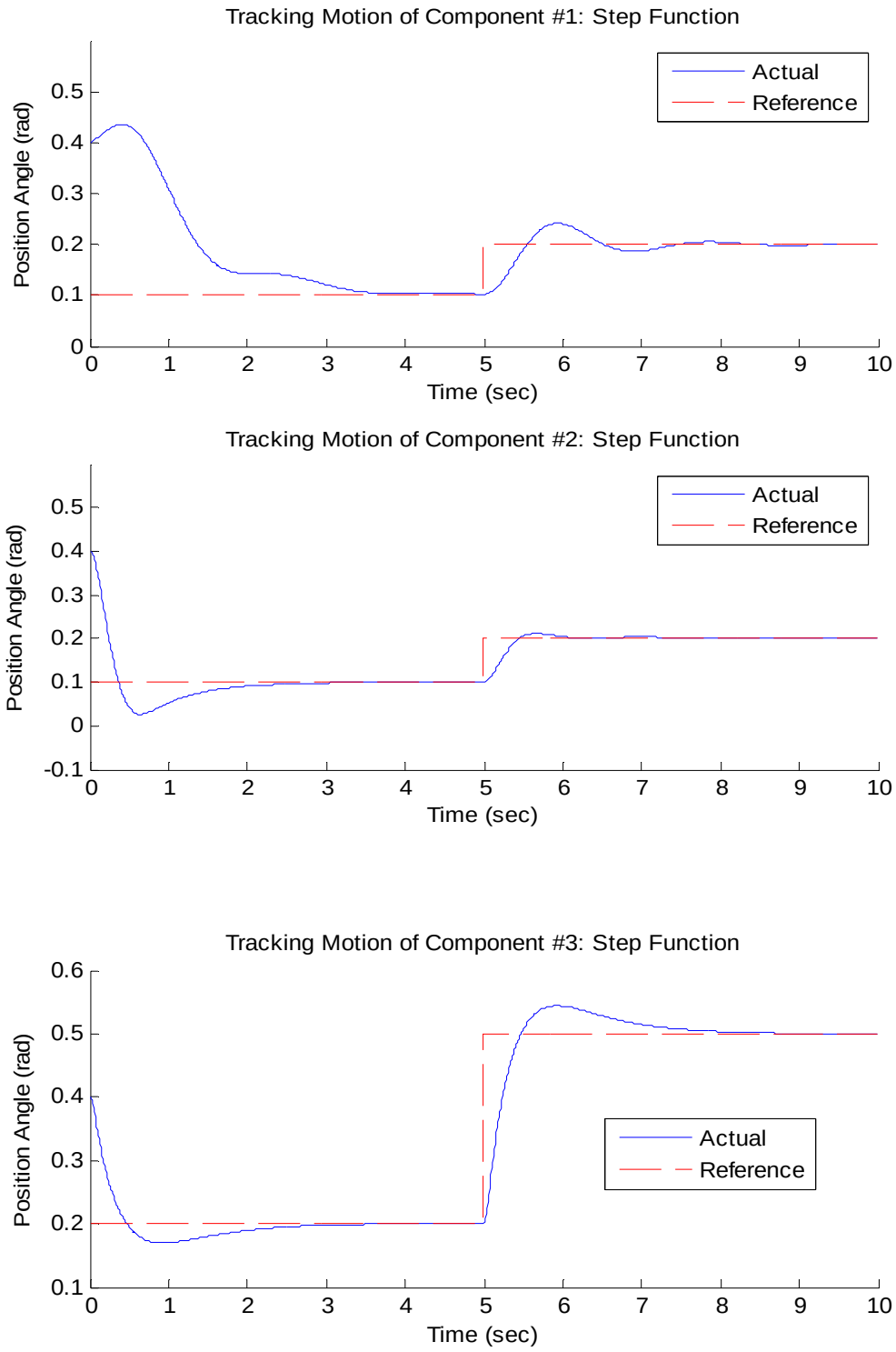
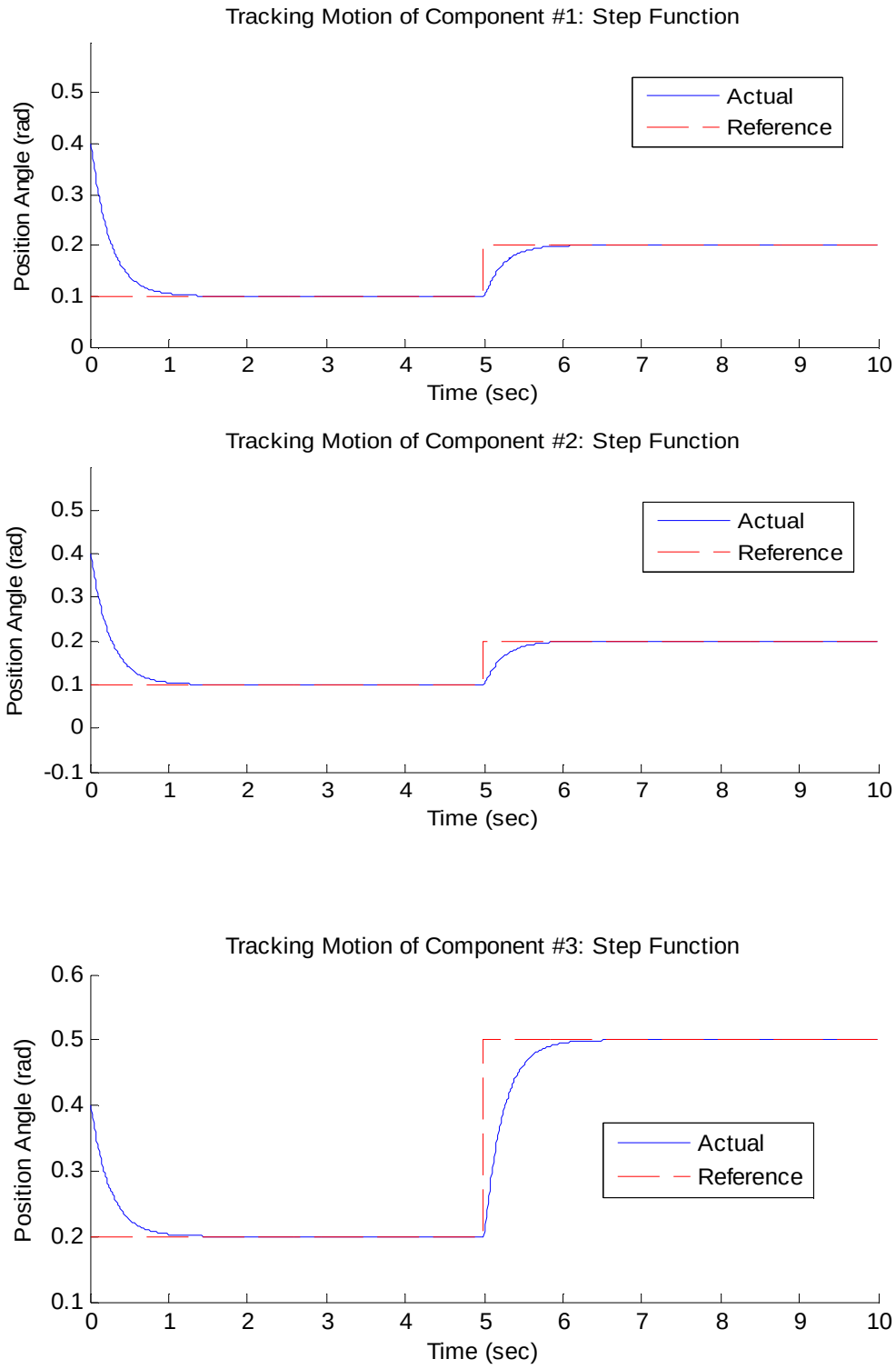


Figure 13: d) DI Model for $K_P = 400$, $K_D = 100$



Input 3: Sinusoidal Commanded Motion

The following mode is most similar to that of a guided missile simulation in an HWIL system. The relative motion between the seeker and target is more likely to be in the form of smooth curves getting sharper as the missile approaches the target rather than instantaneous steps in position. Therefore, the results from this mode are probably more relevant to flight motion table simulations. The motion is a simple sinusoidal wave function which oscillates for a total duration of 10 seconds.

Figures 14a and 14b represent a PID and dynamic-inversion controller with a proportional gain equal to 200 and a derivative gain equal to 50. Figures 14c and 14d represent the same except that more aggressive K_P and K_D gains of 400 and 100 respectively were applied.

In this mode, the dynamic-inversion has little to no overshoot in both cases while still managing to converge to the curve within first second of the simulation. Contrary to the dynamic-inversion results, the PID controller seemed to struggle in its convergence on the constantly changing curve, most likely due to the integral term. Figure 14a shows difficulty in the pitch and yaw, while Figure 14c shows difficulty converging at the peaks of the sine wave.

Figure 14: a) PID Model for Sinusoidal Command and $K_P = 200$, $K_D = 50$

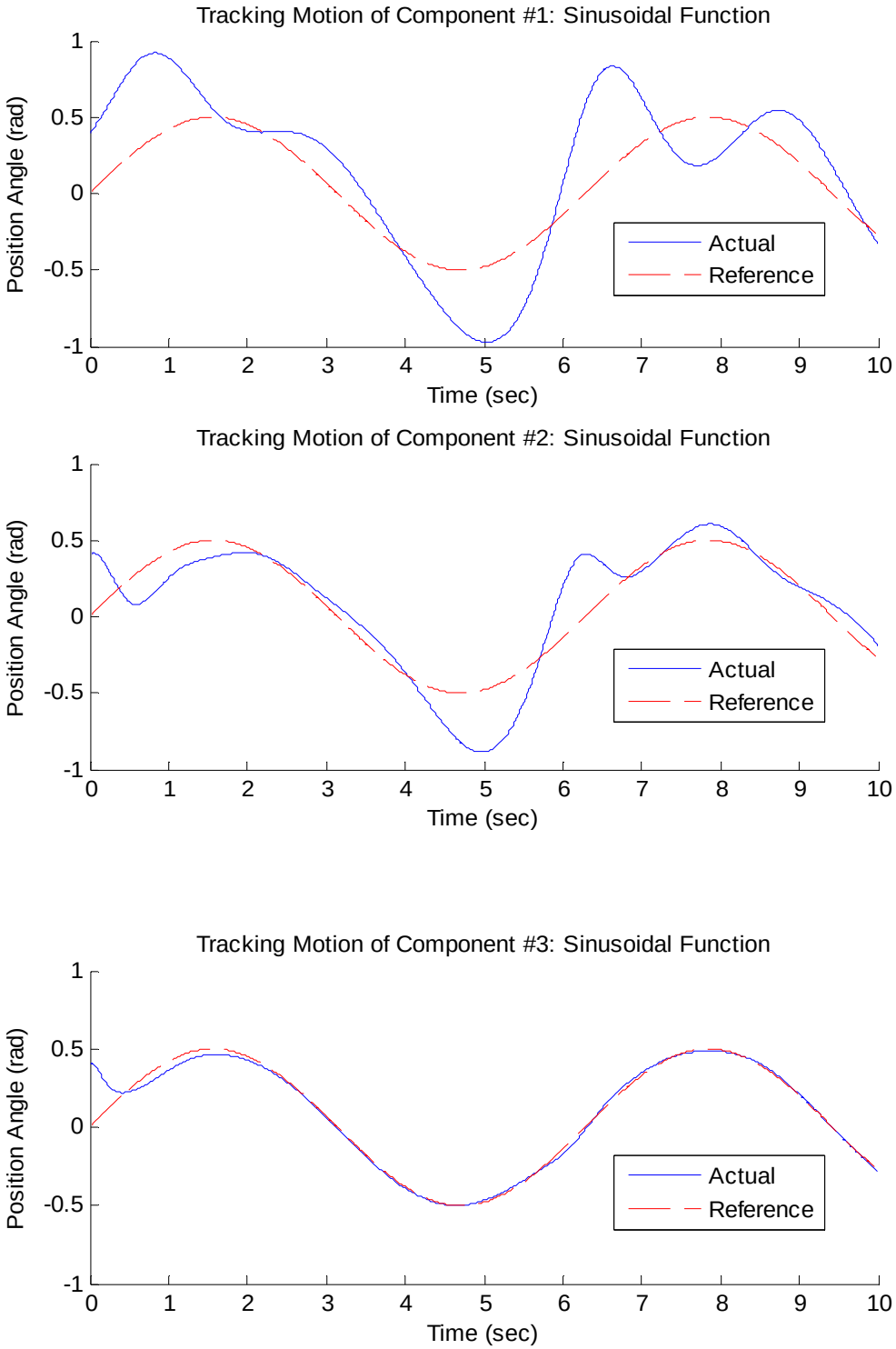


Figure 14: b) DI Model for Sinusoidal Command and K_P = 200, K_D = 50

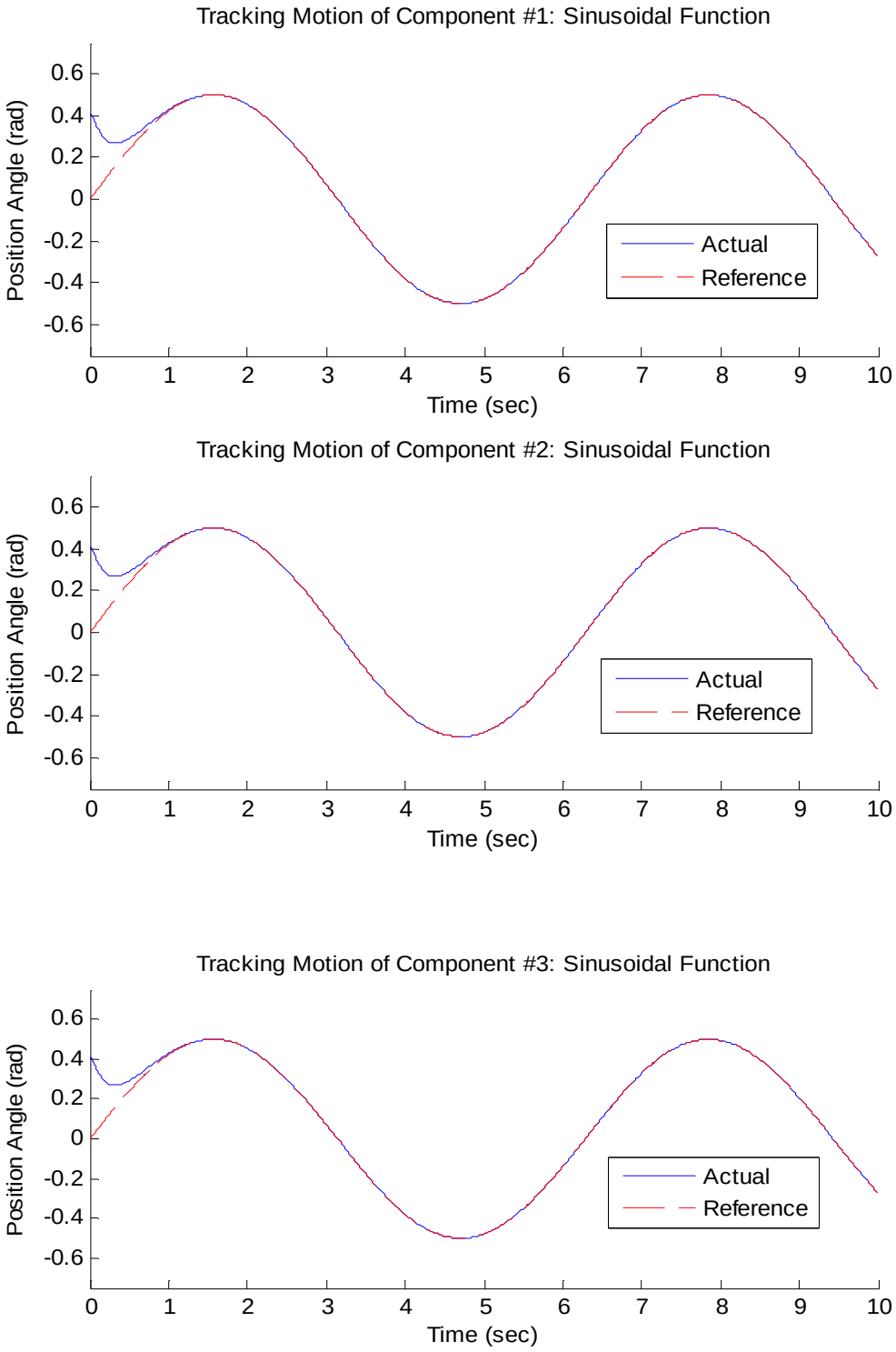


Figure 14: c) PID Model for Sinusoidal Command and K_P = 400, K_D = 100

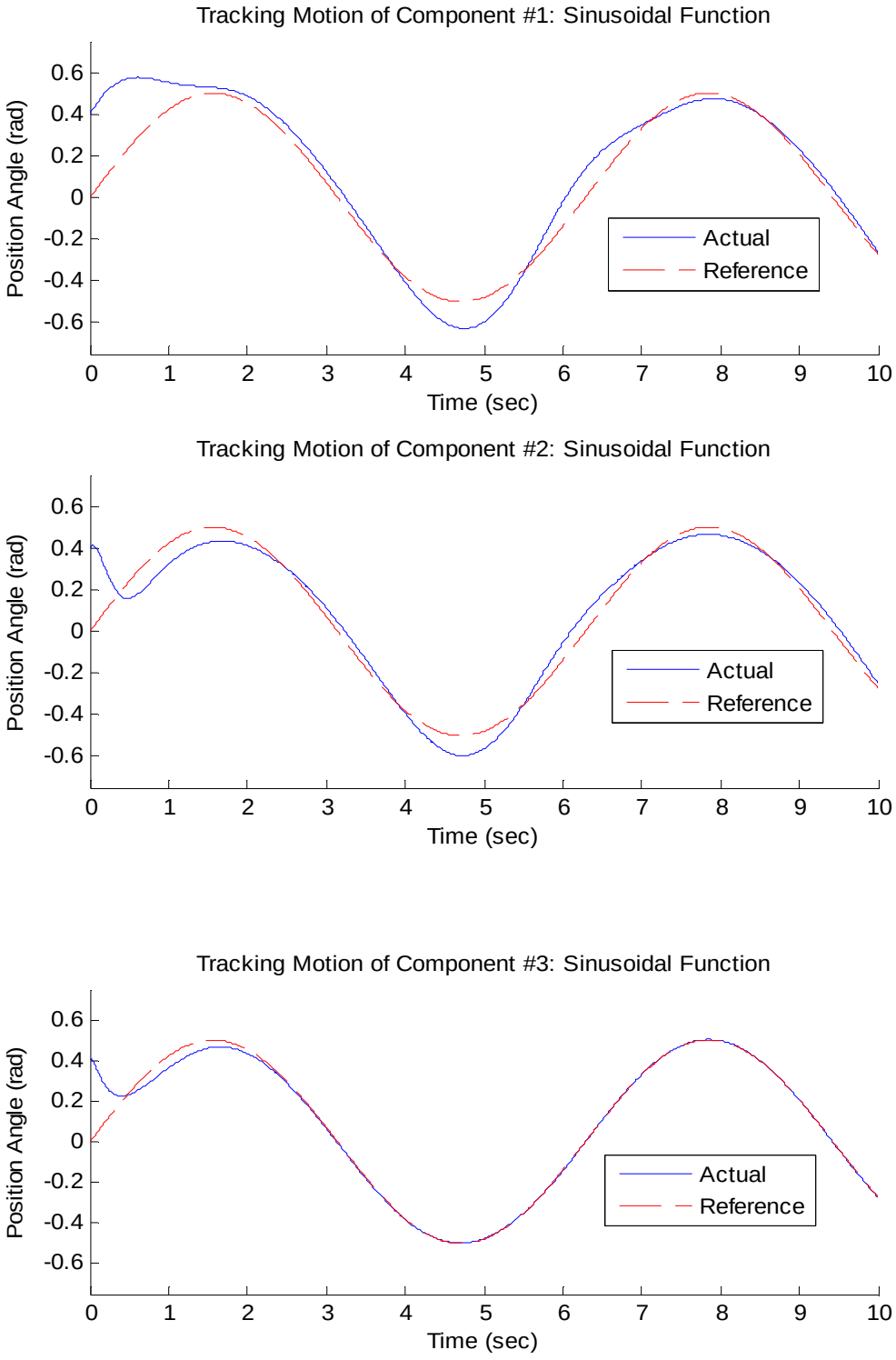
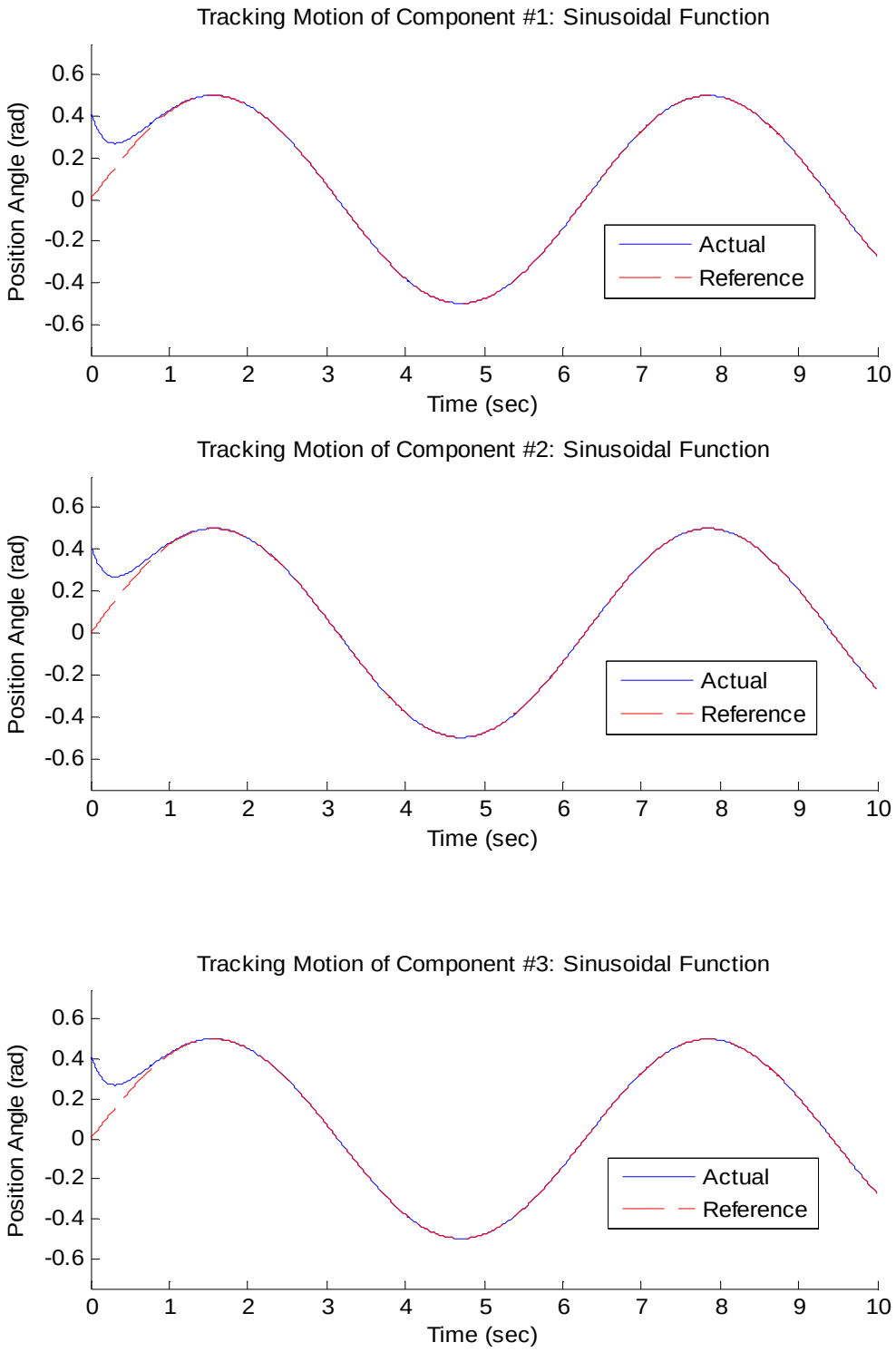


Figure 14: d) DI Model for Sinusoidal Command and K_P = 400, K_D = 100



VII. COMPARISON OF PID AND DYNAMIC-INVERSION CONTROLLERS

In addition to studying the gimbaled-angle histories, other methods can be used to compare the performance of the PID and dynamic-inversion controllers. As shown in the previous chapter, the PID controller consistently showed more fluctuation between the actual and commanded motion. This section describes an investigation of the control torques and processor time required by both the PID and dynamic-inversion controllers.

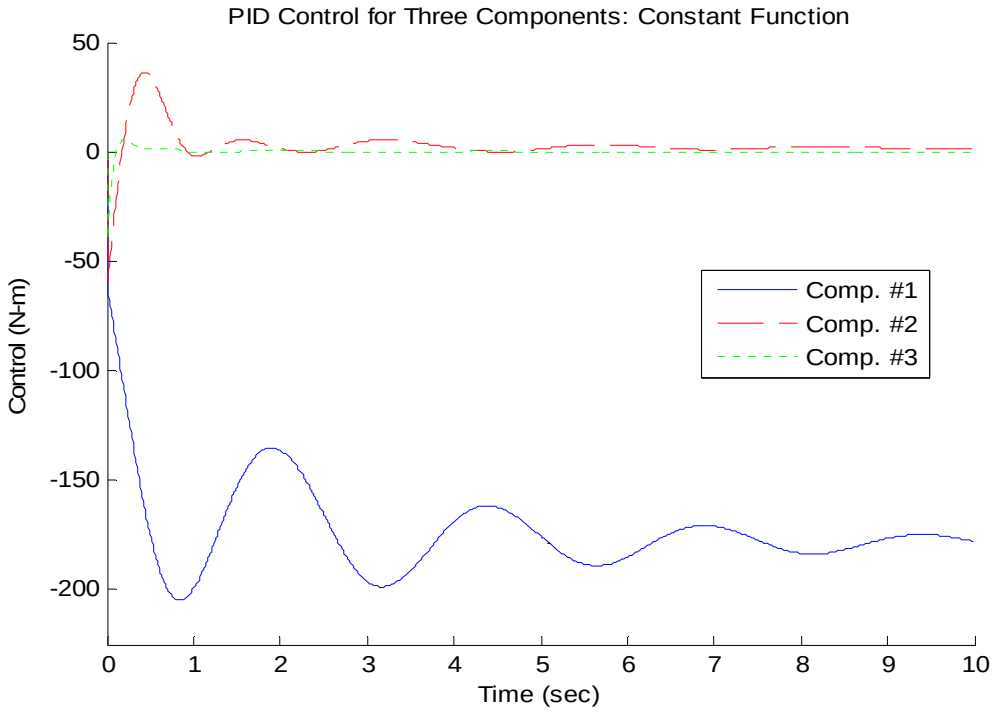
Comparison of Controls for each Input

Two factors being considered while looking at the control for each of the twelve systems are the settling time (how long is the controller required to operate before convergence) and the maximum control required. These two factors can play a role not only in the system performance, but also the life span of the flight table.

The controls for all three components, both controllers and all three commanded inputs are shown in Figures 15-17. A common trend is present in all results shown. Concerning the PID controller, the reaction is slower and the control is more oscillatory. However, the controls are very low. The maximum control applied to the system is around a

magnitude of 200 N-m. With the dynamic-inversion controller, the reaction is nearly instantaneous in all cases with little overshoot and a short damping time. However, the control approaches levels around a magnitude of 1200 N-m.

Figure 15: a) Control for Constant Command and $K_P = 200$, $K_D = 50$



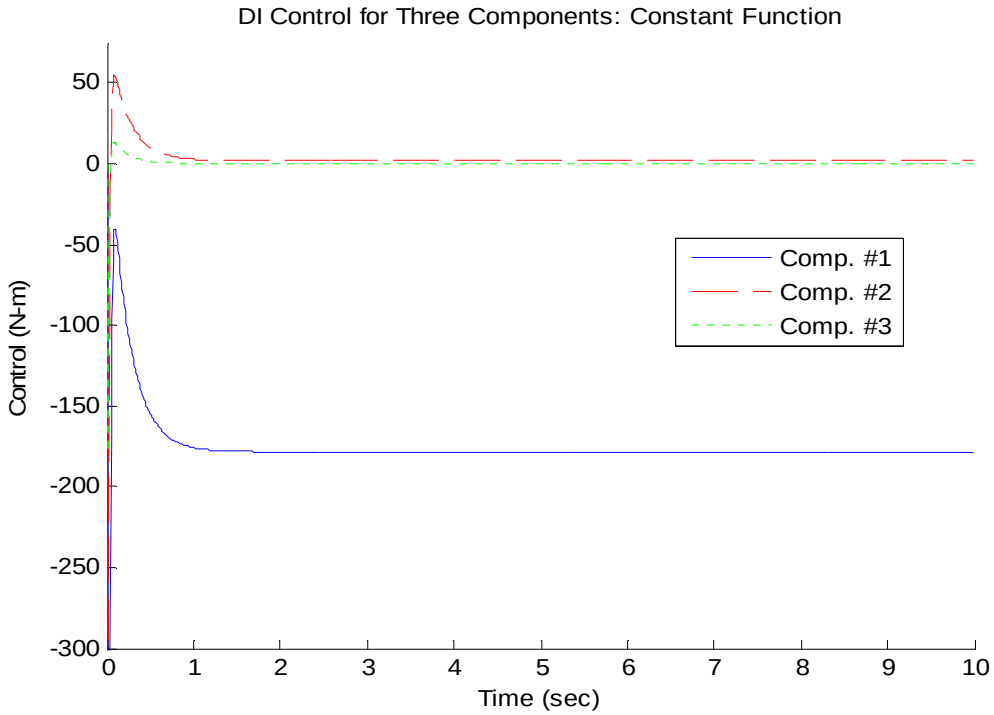
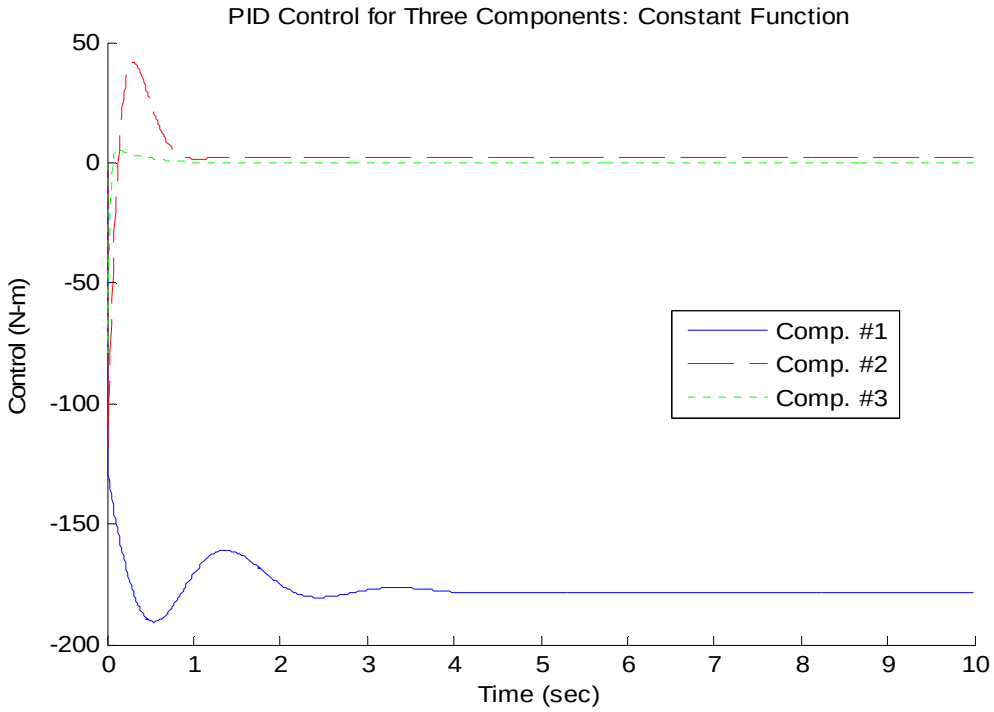


Figure 15: b) Control for Constant Function and $K_P = 400$, $K_D = 100$



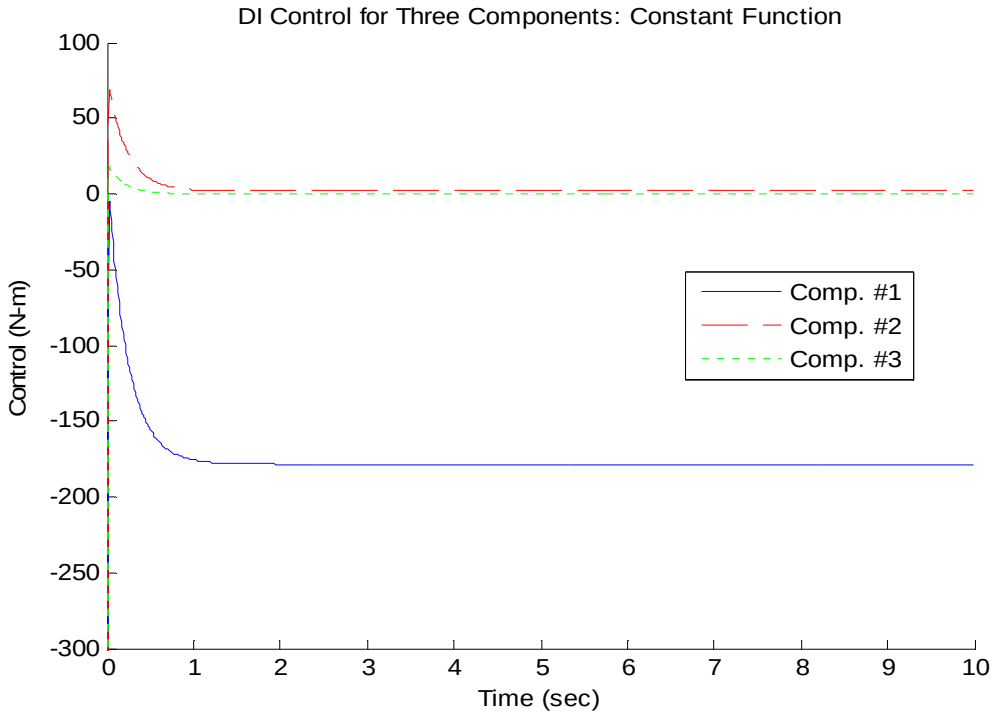


Figure 16: a) Control for Step Command and $K_P = 200$, $K_D = 50$

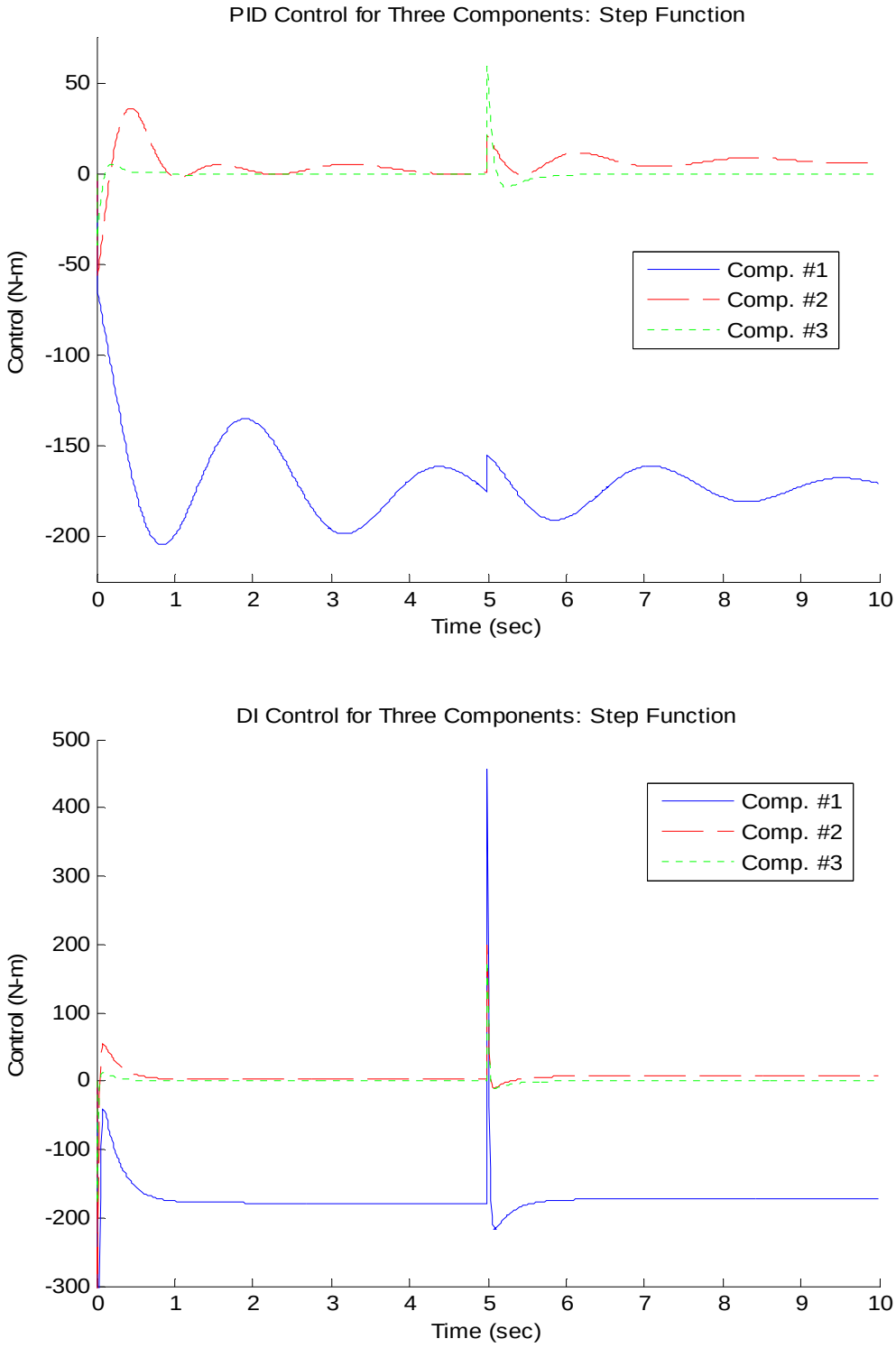


Figure 16: b) Control for Step Command and K_P = 400, K_D = 100

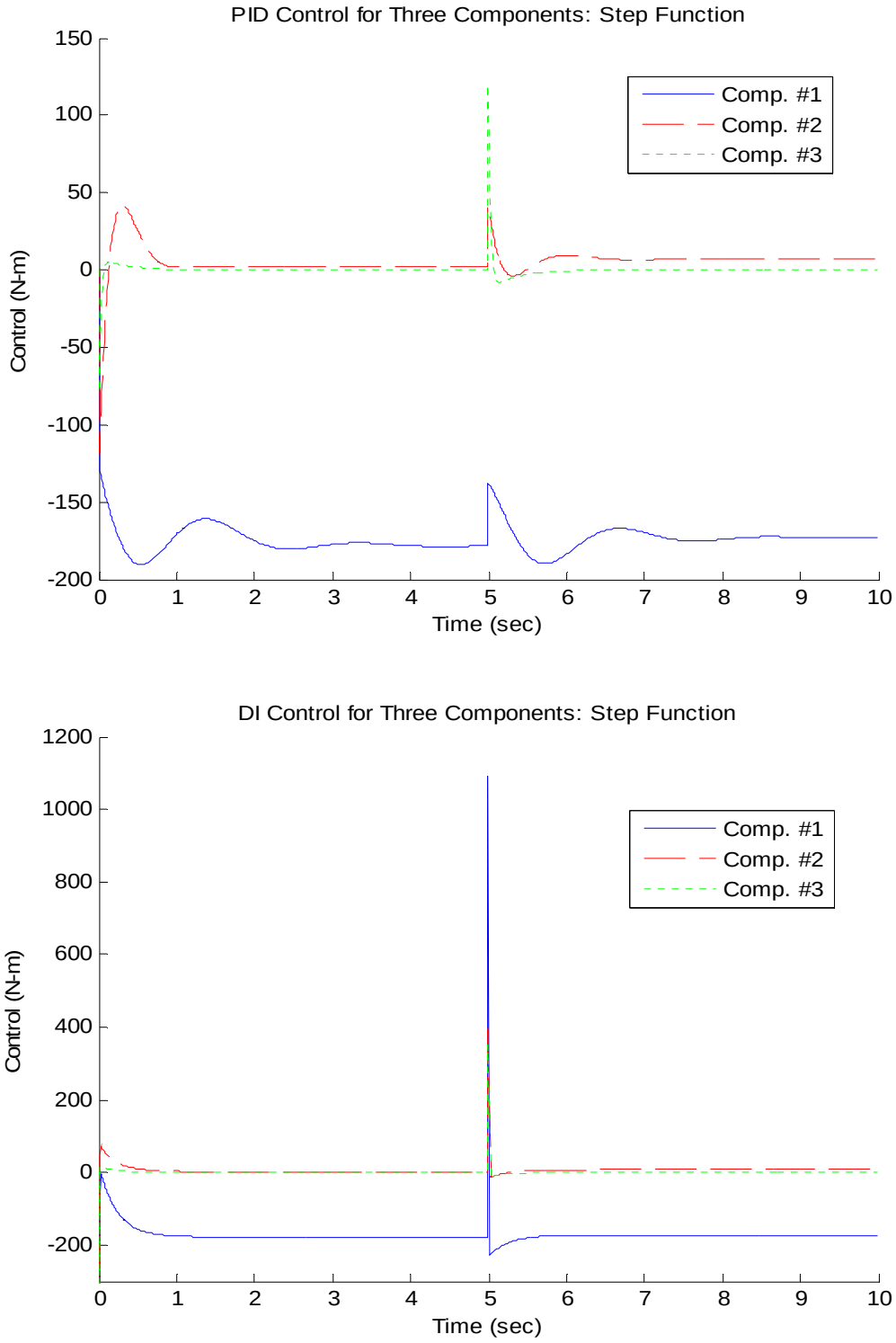


Figure 17: a) Control for Sinusoidal Command and $K_P = 200$, $K_D = 50$

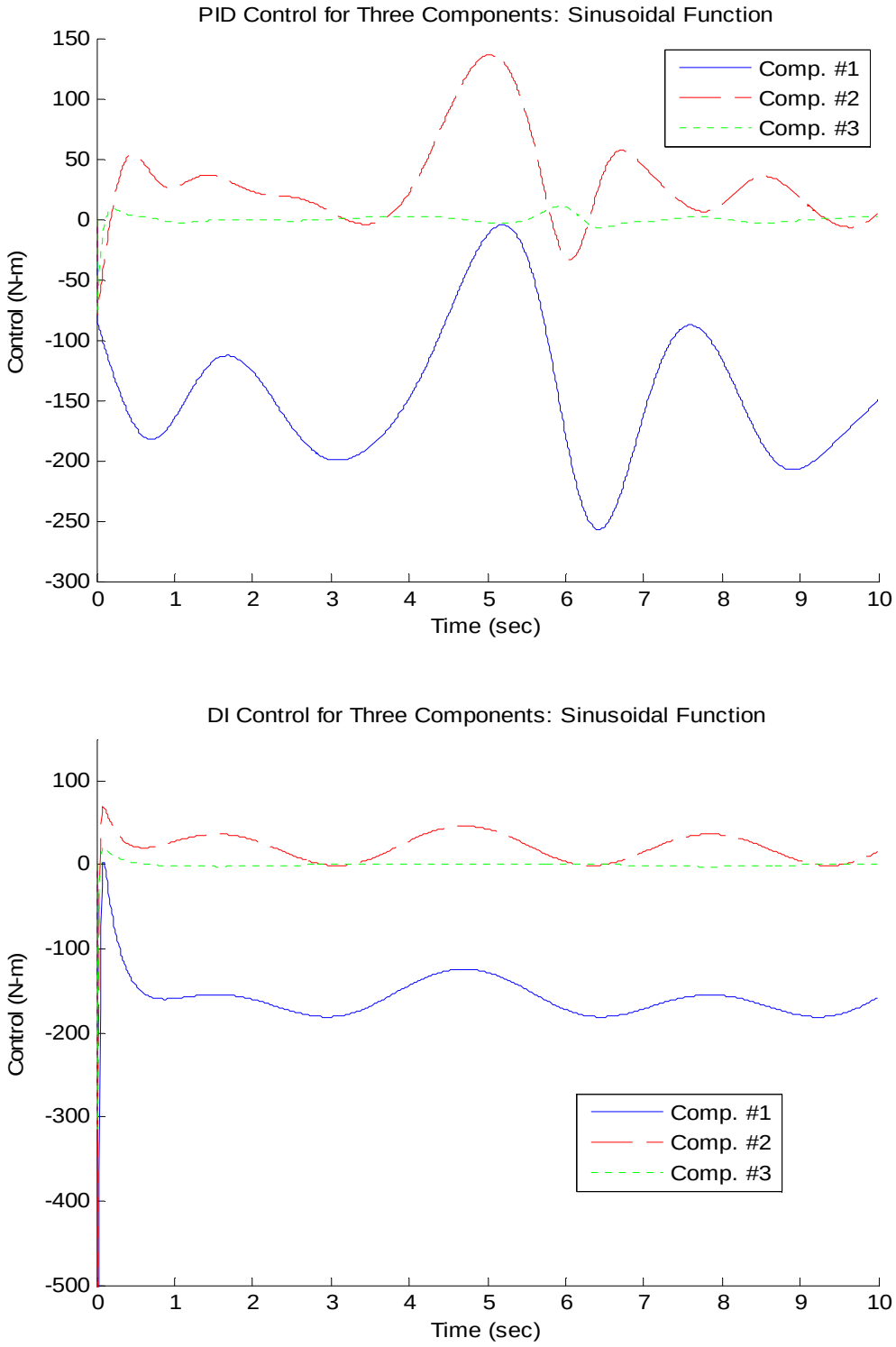
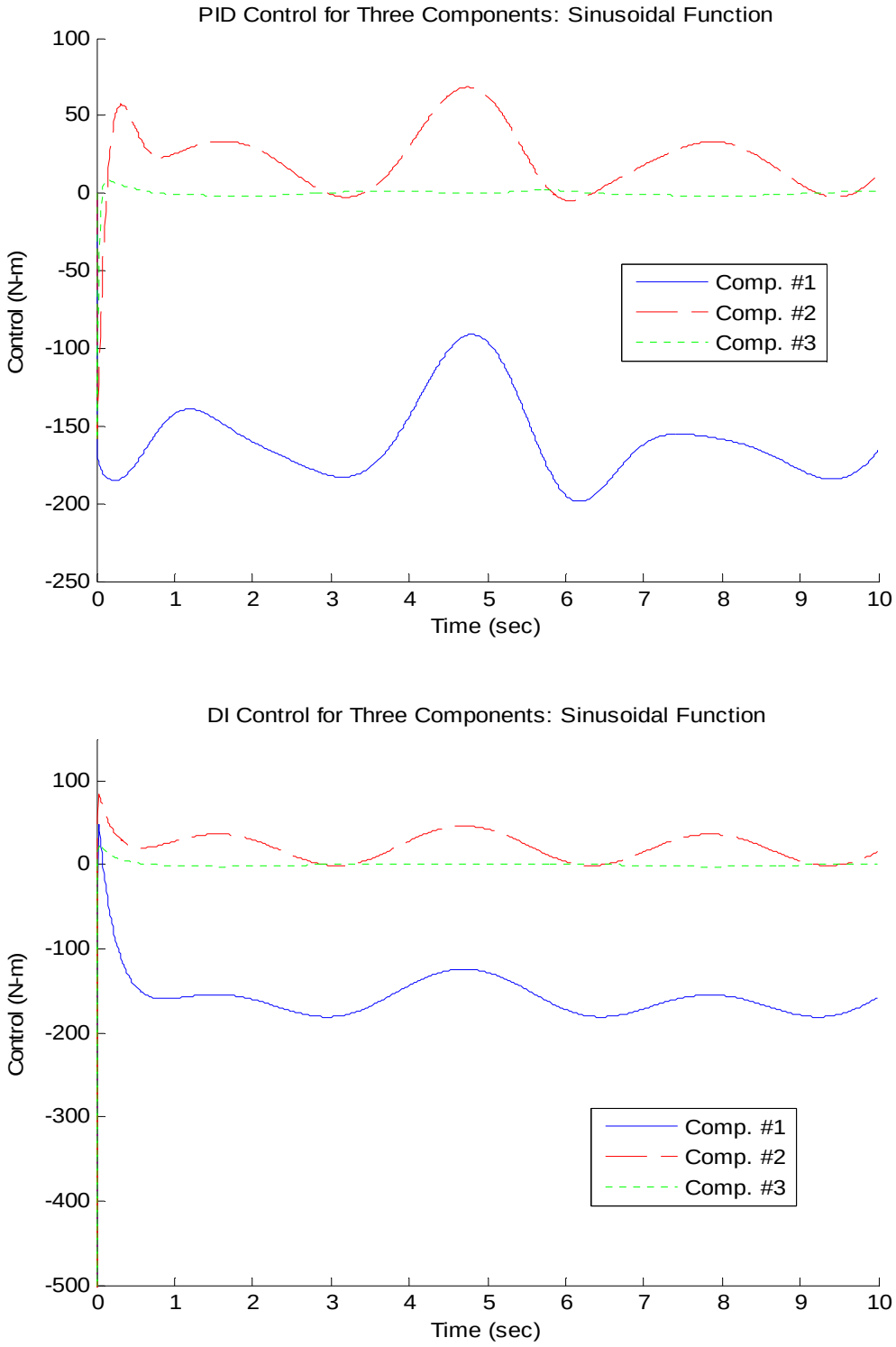


Figure 17: b) Control for Sinusoidal Function at K_P = 400, K_D = 100



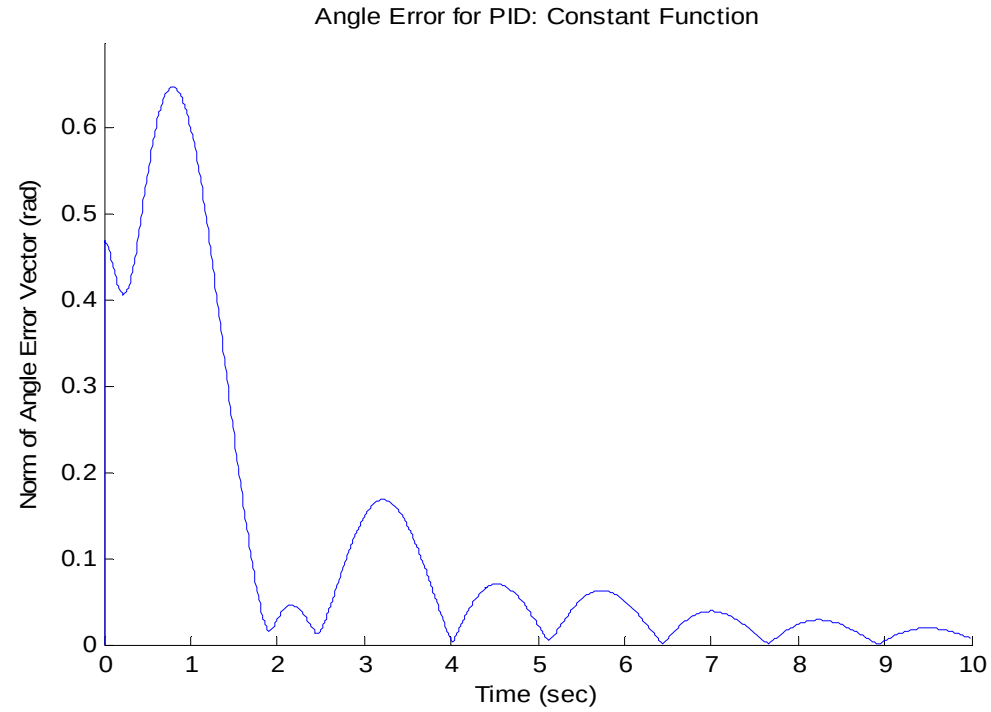
In comparing the controllers, the PID controller causes oscillatory reversing torques, while the dynamic-inversion controller requires greater peak torques. Depending on the application, it is not clear which is more desirable. However, other factors, such as the results of the error analysis and processing time also factor into the comparison decision.

Error Analysis for Each Input

Like in any system, one of the most effective ways to compare the performance of two controllers is to analyze the error produced between systems. Because there are three angles examined, all having some sort of error, the error analyzed in Figures 18a-c and 19a-c are the norm of each error vector, giving a scalar quantity to plot.

The first set of errors examined compares the angle errors throughout the entire simulation. Figures 18a-c compare the errors throughout the simulation for the lower gain settings (K_P = 200, K_D = 50). These gains are easy to compare due to the lack of convergence of the PID function after the initial angle jump. In every case, there is an oscillatory motion associated with the PID controllers. Every time the system oscillates, more error is added into the system. In all cases, the dynamic-inversion controller converges to zero error and remains relatively constant.

Figure 18: a) Error for Constant Command and $K_P = 200$, $K_D = 50$



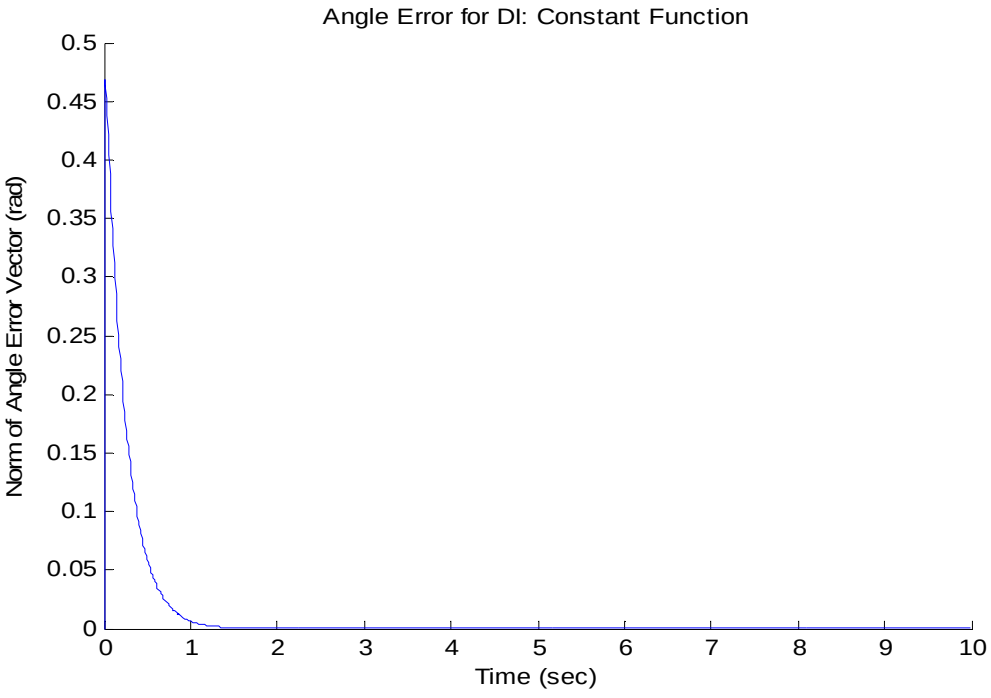


Figure 18: b) Error for Step Command and K_P = 200, K_D = 50

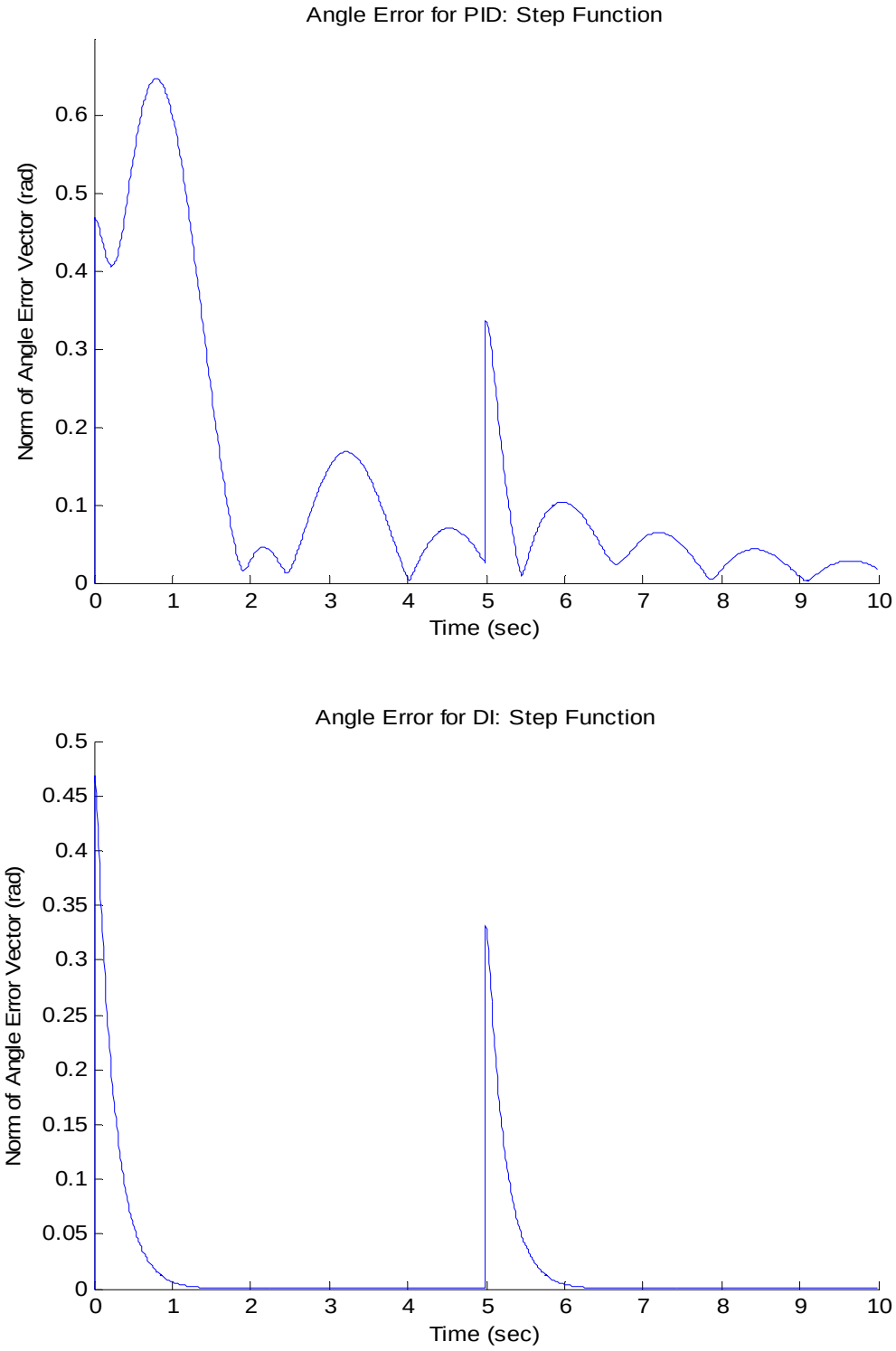
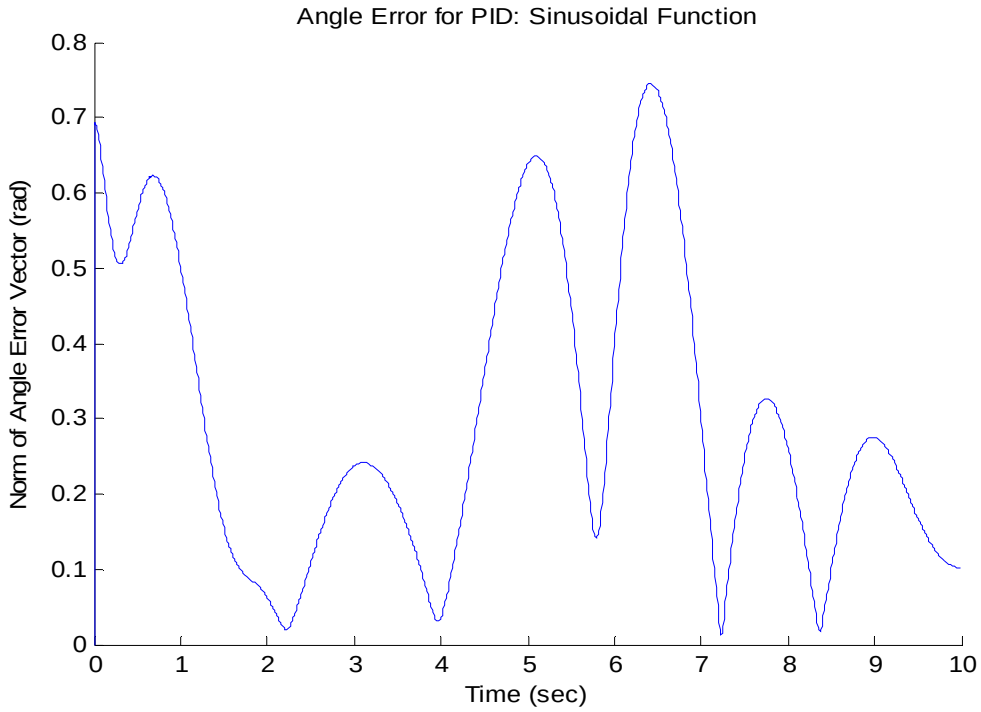
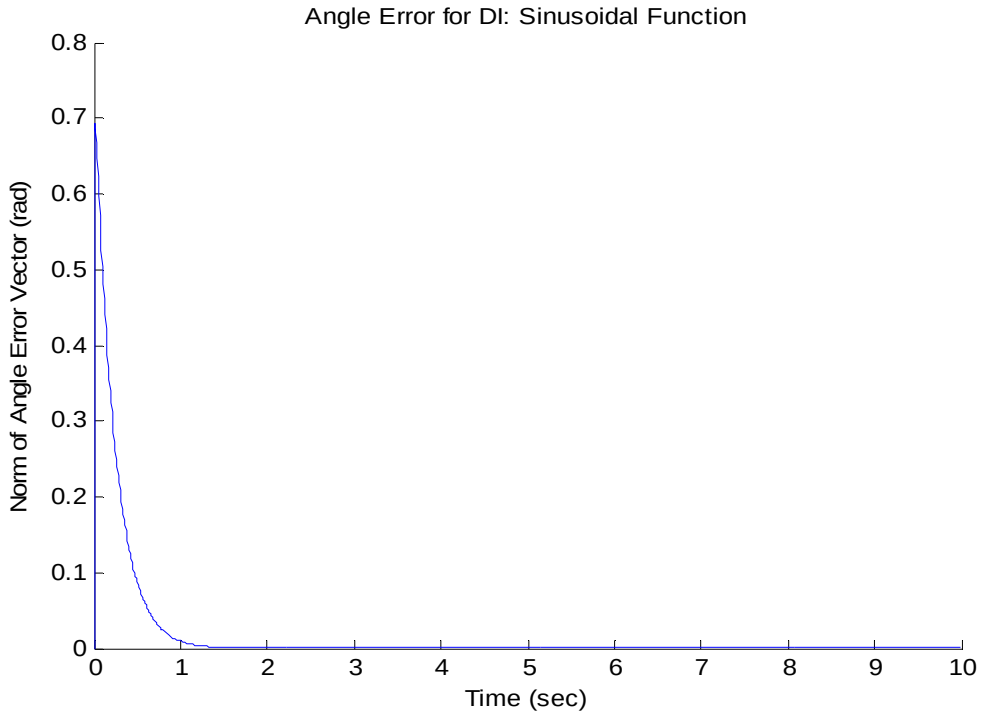


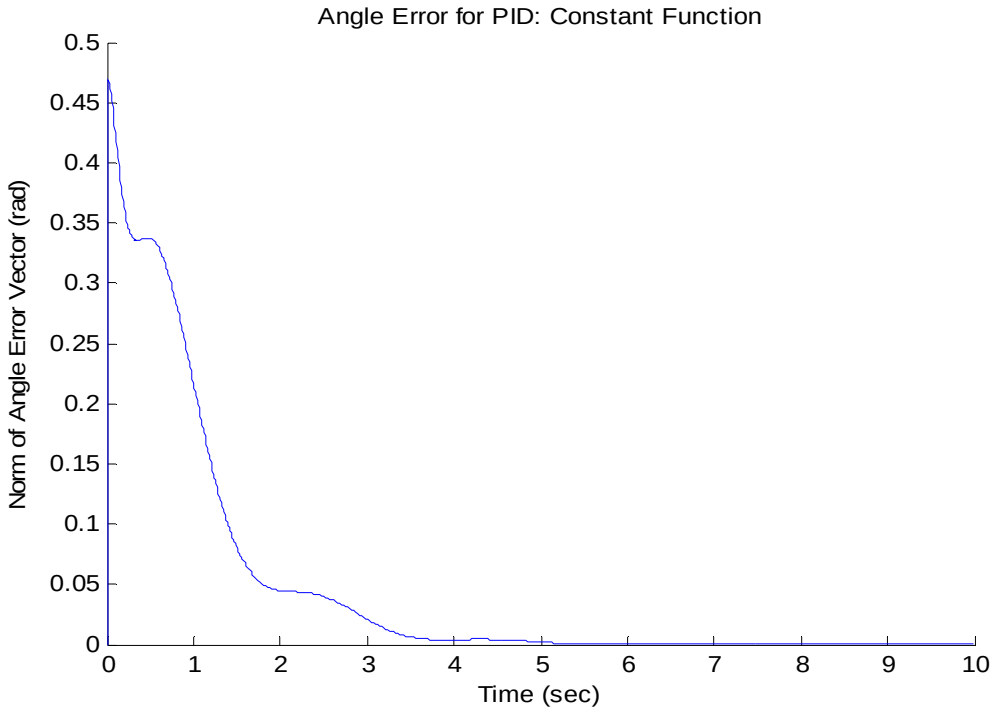
Figure 18: c) Error for Sinusoidal Command and K_P = 200, K_D = 50





After examining Figures 19a-c, one can compare the errors throughout the simulation for the higher gain settings (K_P = 400, K_D = 100). With the more aggressive gains used, the errors for both controllers converge with less overshoot, thereby reducing the error after the initial jump from the starting parameters to the commanded parameters.

Figure 19: a) Error for Constant Command and K_P = 400, K_D = 100



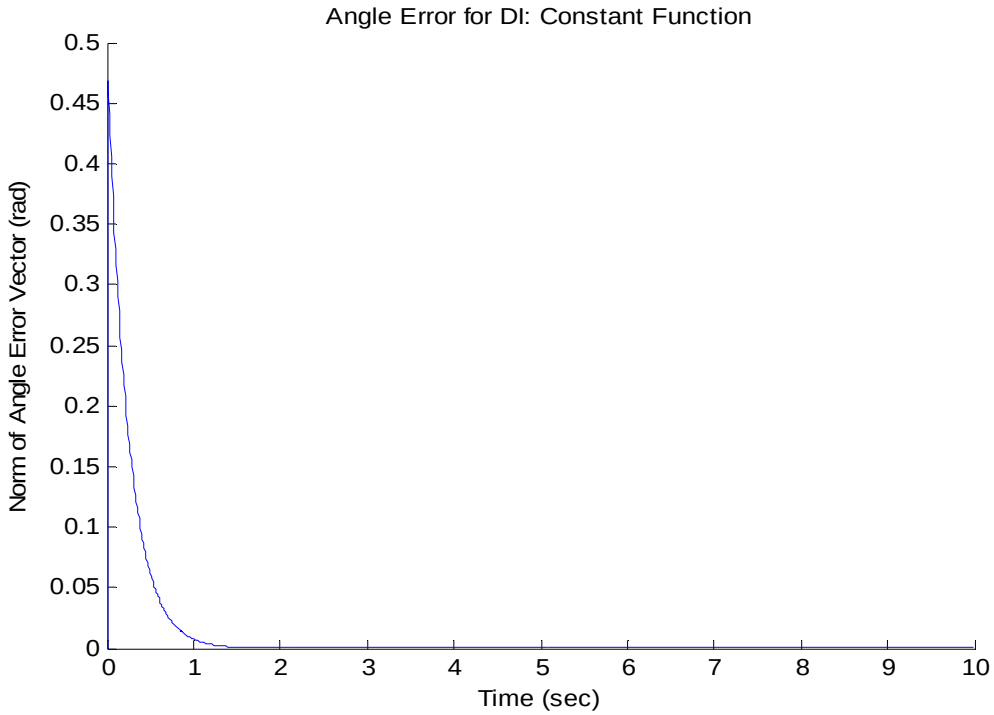


Figure 19: b) Error for Step Command and K_P = 400, K_D = 100

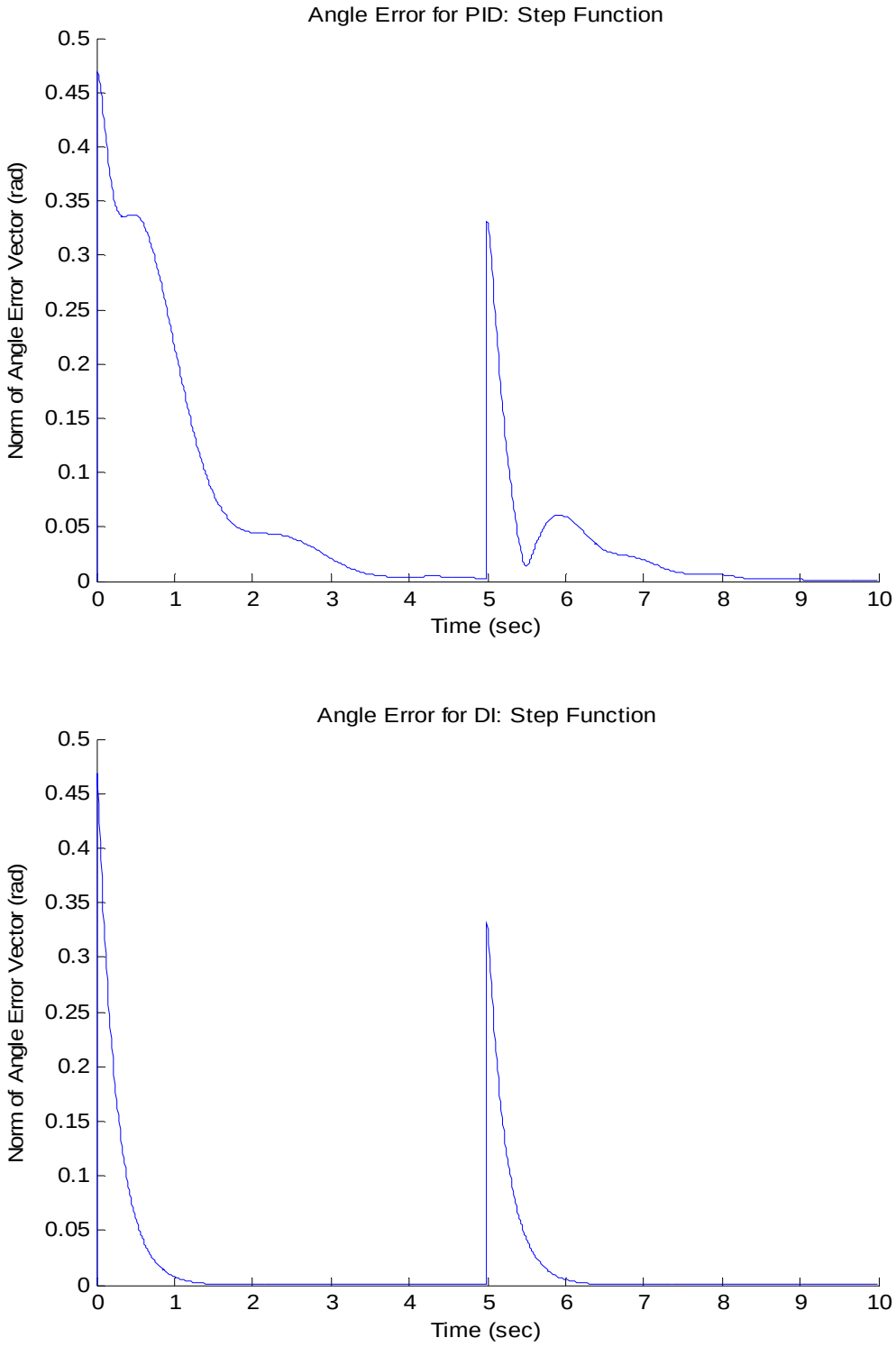
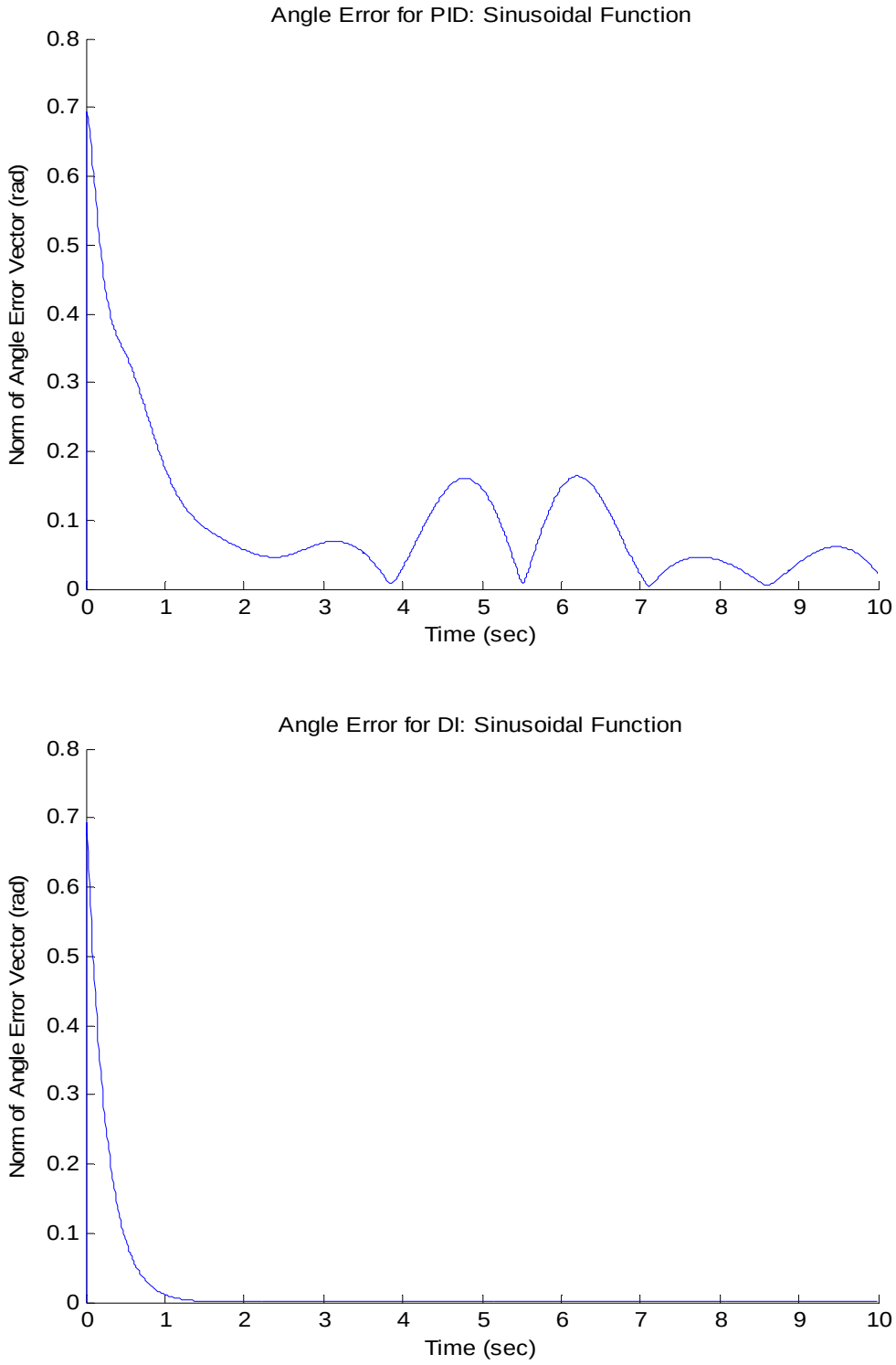


Figure 19: c) Error for Sinusoidal Command and K_P = 400, K_D = 100



In this case, the more aggressive gain settings resulted in less overshoot. Therefore, the majority of angle error is dependent upon the area under the curve from the initial maneuver to the commanded values. Also, as mentioned in previous chapters, more aggressive gains tend to lessen the oscillation in the PID controller, while increasing the rise time in the dynamic-inversion controller. In certain situations, mostly for short periods of time, it may be difficult to determine which controller produces less error. Therefore, Table 4 takes into account all twelve experiments and sums up the total error history over the course of each simulation. As shown, in every case the amount of error in the dynamic-inversion system is significantly less than that in the PID controller.

Table 4: Total Error Comparison of PID and DI controllers

Gains Used	Controller Type	Commanded Input Type		
		Constant Input	Step (Impulse) Input	Sinusoidal Input
$K_P = 200$ $K_D = 50$	PID	0.11450	0.12880	0.30250
	Dyn. Inv.	0.01170	0.02000	0.01740
$K_P = 400$ $K_D = 100$	PID	0.04790	0.06200	0.09890
	Dyn. Inv.	0.01170	0.02000	0.01730

The difference in the error from Table 4 can further be demonstrated in Figures 20a-b. One concept to note is the difference in the PID and dynamic controller errors for the simulation. As discussed earlier, as the gains increase, the amount of discrepancies

between the error in the PID and dynamic-inversion controller is lessened. In Figure 20a, the total error in the PID is approximately 1600-1700% greater than that in the dynamic-inversion controller for sinusoidal motion. However, when the gains are increased, such as in Figure 20b, the differences between the total errors in the simulation decrease significantly. The PID controller error is approximately 400-500% greater than the dynamic-inversion for sinusoidal motion. There is an obvious gap change between the two controller gain adjustments.

Figure 20: a) Error Comparison in PID and DI with K_P = 200, K_D = 50

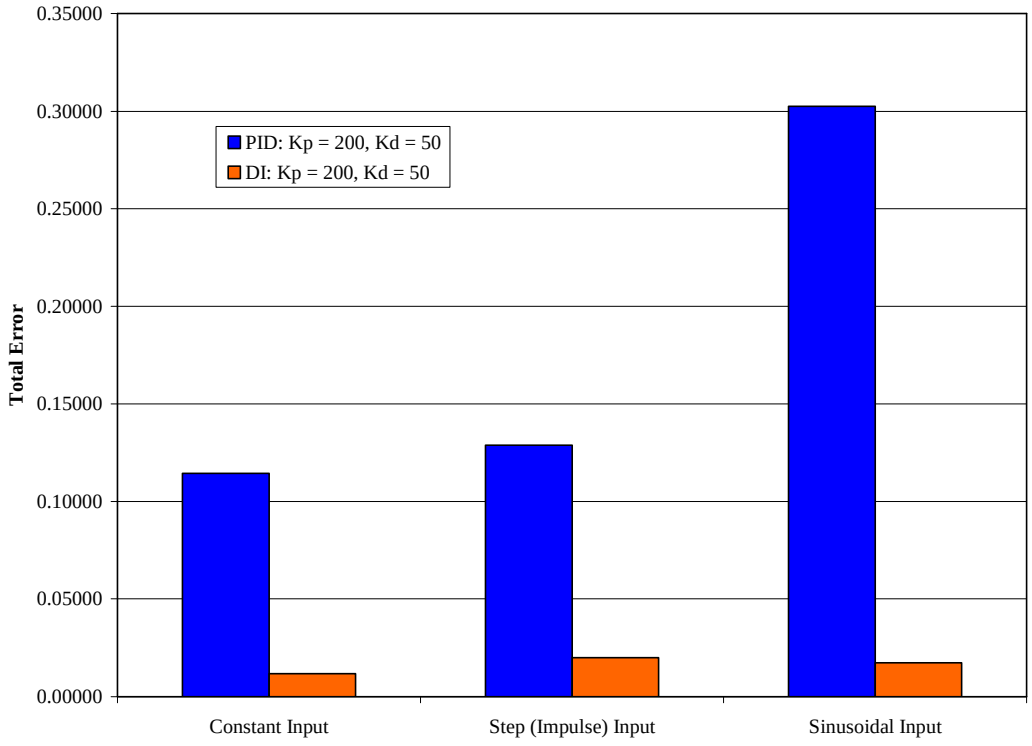
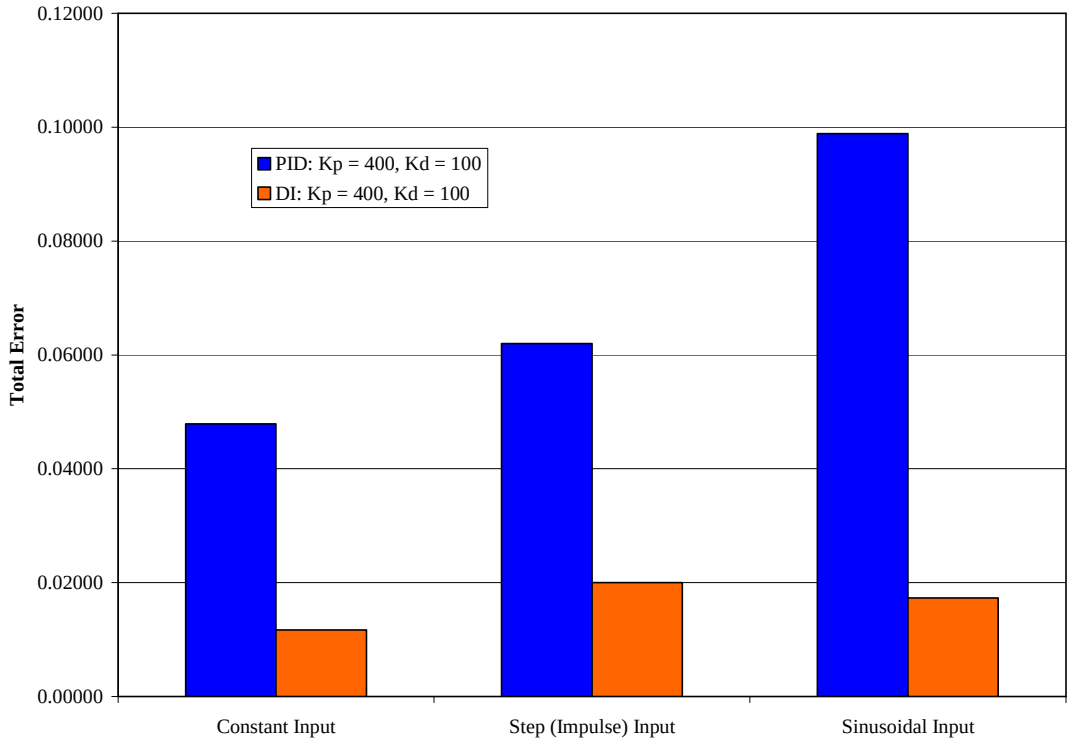


Figure 20: b) Error Comparison in PID and DI with K_P = 400, K_D = 100



Runtime Analysis for Each Input

A third characteristic of each controller to contemplate when deciding which to use in a flight table is the required processor time. A dynamic-inversion controller could converge faster than PID controller, but if it is hugely more computationally expensive, then the PID would be the obvious choice, and vice versa.

In order to analyze the amount of time it takes to complete a simulation of each controller, a “tic-toc” Matlab function was applied. The “tic”, where Matlab starts recording the time, was placed right before the for-loop in the main program. The “toc”, where Matlab stops timing the system, was placed directly after the for-loop was closed. Note that this sums the entire time required to simulate the system, not just the computational expense of evaluating the control. The other computations involved in each simulation should be similar, with the main difference being the controller evaluation. The placement ensured that the plotting of the variables is not taken into effect.

Table 5 demonstrates the results of the controller runtimes, while Figures 21a-b graph the data for each set of gains. Note that in all cases the dynamic-inversion controller runs around 30% longer than the PID controller.

It is the author’s belief that the structure of the dynamic-inversion controller program that causes the increase in runtime. When the PID controller calls the function “control_develop”, the control is directly calculated through linear equations and returned to the main program for future processing. However, when the dynamic-inversion controller calls “control_develop”, the function then must call the function “model_components”, compute relatively large nonlinear system components (mass matrix, Coriolis terms, etc.) for both the commanded and actual system, return those values to “control_develop” where another large equation is used to calculate the control

before it is sent back to the maim program for processing. These extra steps and large calculations attributed to the increase in runtime.

Table 5: Analysis of Runtimes

Gains Used	Controller Type	Commanded Input Type		
		Constant Input	Step (Impulse) Input	Sinusoidal Input
		10 sec	10 sec	10 sec
Kp = 200 Kd = 50	PID	1.20552	1.20392	1.22571
	Dyn. Inv.	1.57709	1.57208	1.56319
Kp = 400 Kd = 100	PID	1.22680	1.20592	1.20208
	Dyn. Inv.	1.56431	1.60439	1.59235

Figure 21: a) Runtime Comparison for PID and DI with K_P = 200, K_D = 50

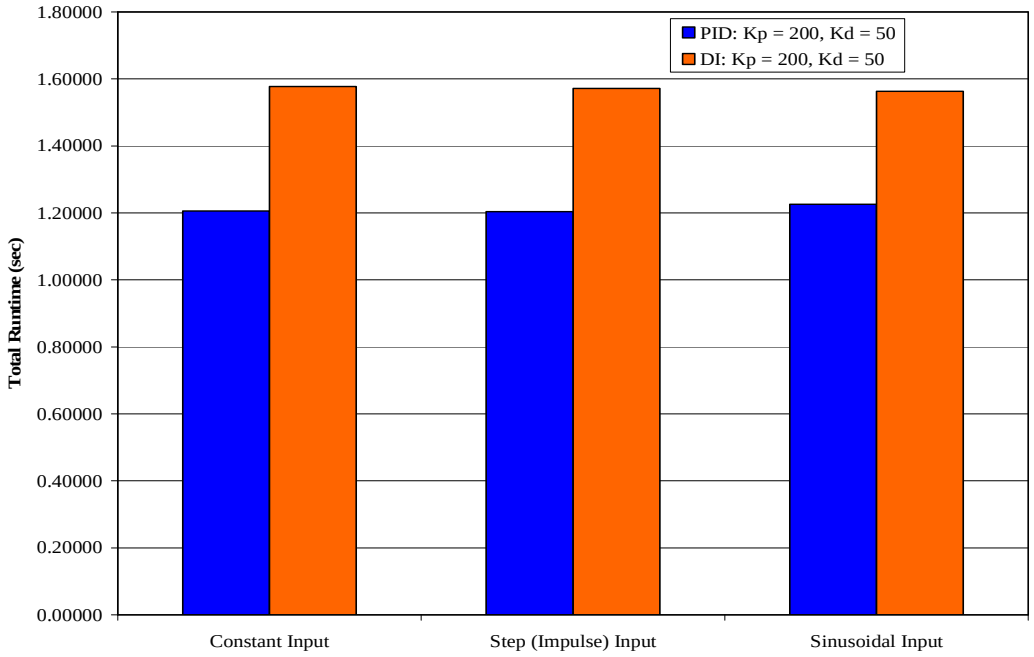
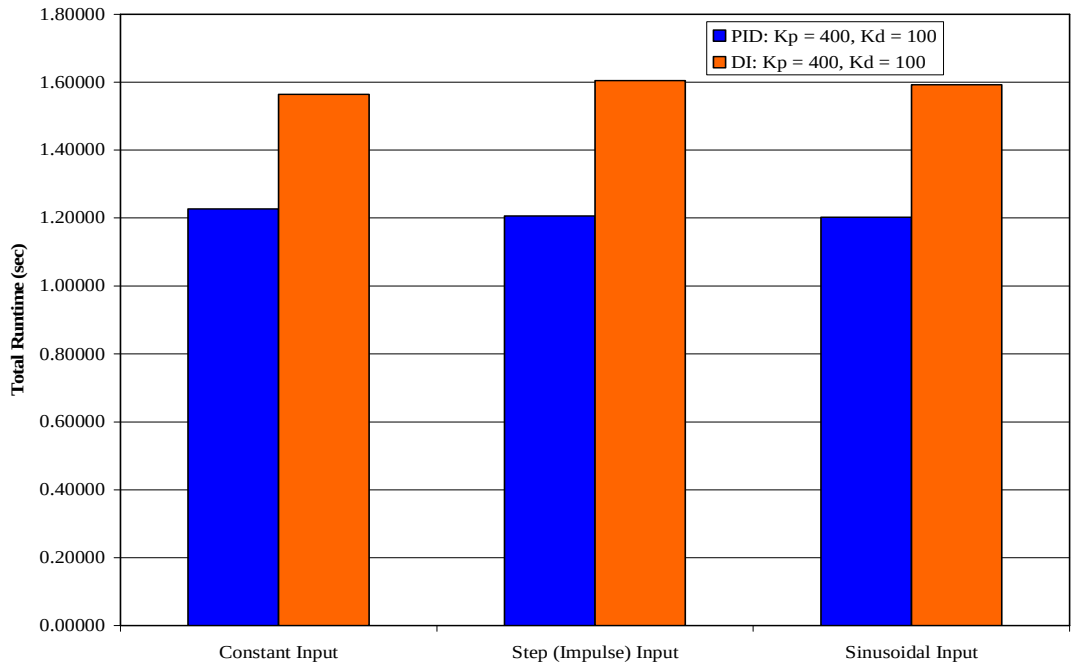


Figure 21: b) Runtime Comparison for PID and DI with K_P = 400, K_D = 100



On average, the time it takes to run the PID simulation is approximately 12% of real-time (it takes approximately 7.20 seconds to run a 60 second simulation). The time it takes the dynamic-inversion simulation to run is approximately 16% of real-time (approximately 9.60 seconds to run a 60 second simulation). HWIL systems work in real-time, as opposed to simulation time. For every minute long simulation, 2.40 seconds are lost due to the dynamic inversion program. This indicates that the dynamic-inversion controller could likely be implemented on a digital controller without great increase in cost. However, studies will have to be made to reduce the difference between the controller runtimes.

VIII. CONCLUSIONS

Because of their ability to function well enough and its user-friendly simplicity, PID controllers are employed in the majority of industrial design. However, when the industrial environment does not require simplicity in a system, is there a nonlinear controller that can be rendered more effective.

The dynamic-inversion controller developed in this thesis represents a controller capable of manipulating the movements of a three gimbaled, nonlinear system.

Derived from Lagrange's equations for kinetic and potential energies and Lyapunov global stability theory, the dynamic-inversion controller was tested. Results from these tests were compared with results obtained using a PID controller. The dynamic-inversion method consistently showed a higher quality of response for not only a wide range of gains, but also numerous types of commanded functions. The dynamic-inversion method showed less error through the simulation with faster convergence and larger damping in terms of the gimbaled motion and gain. The PID controller did execute faster than the dynamic-inversion, but the lag in the dynamic-inversion controller should not be enough to warrant a change in controllers.

Recommendations for future work would include applying and testing both PID and dynamic-inversion controllers to an actual flight table to analyze how the system results vary in the physical world. Also note that the experiments performed do not include a seeker head during the test. How would the extra mass and adjustment of the moments of inertial handle in a simulation and actual test site? One may also analyze the stresses present during a reverse torsion. The PID controller was consistently oscillating. For an active table, this would cause vibrations and noise. However, because the control in the dynamic-inversion was so large, would deceleration in the gimbals cause any unwarranted stress and strain on the system? Finally, the runtime of the dynamic-inversion was slightly higher than the run time of the PID controller. An examination on how that would affect performance may lead to a structure of the program that causes it to execute faster.

BIBLIOGRAPHY

Astrom, Karl Johan. *Control System Design: PID Control*. 2002. </www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf>

Carter, J. and K. Willis. "A History of Flight Motion Simulators used for Hardware-in-the-Loop Testing of Missiles." In *SPIE Conference on Technologies for Synthetic Environments: Hardware-in-the-Loop Testing III 1998, Orlando, FL*.

Collins, Jack A. *Mechanical Design of Machine Elements and Machines*. New York: John Wiley & Sons, Inc., 2003.

Gutirrez, J. "Proportional-integral-derivative Explained: Tuning PID Controls." *EDA Design Line* 13 Apr. 2007.

"Hardware in the Loop Laboratories (HWIL)." *EWSSA Integrated Products Team*. 15 Sep. 2003. NavAir Weapons Division. 20 June 2007. <www.nawcwd.navy.mil/ewssa/prod_srv/hwil_fac.htm>

"High Performance Computing Workshop." *Developmental Test Command*. 12 Aug. 2002. United States Army. 20 June 2007. <www.dtc.army.mil/hpcw/1999/burrough/index.html>

"Joseph Louis Lagrange (1736-1813)." *Mathematicians of the Seventeenth and Eighteenth Centuries*. Trinity College, Dublin. 22 May 2007 <www.maths.tcd.ie/pub/HistMath/People/Lagrange/RouseBall/RB_Lagrange.html>

Looye, G. and H.-D. Joos. "Design of Robust Dynamic-inversion Control Laws using Multi-Objective Optimization." In *AIAA Guidance, Navigation, and Control Conference and Exhibit 2001, Montreal CA*. (AIAA-2001-4285).

Marlin, Thomas E. *Process Control: Designing Processes and Control Systems for Dynamic Performance*. New York: McGraw-Hill, Inc., 1995.

"PID-Tutorial." *Control Tutorials for Matlab*. 26 Aug. 1997. Carnegie Mellon and the University of Michigan. 15 May 2007. <www.engin.umich.edu/group/ctm/PID/PID.html>

Plett, G. "Adaptive Inverse Control of Linear and Nonlinear Systems Using Dynamic Neutral Networks." In *IEEE Transactions on Neural Networks* Vol. 14, No. 2 (Mar. 2003): 360-76.

"Three Axis Flight Motion Simulator Series S-450R-3." *Hardware-in-the-Loop Flight Motion Simulator*. Acutronic. 30 May 2007. <www.acutronic.com/upload/pdf/Data_Sheet_S-450R-3.pdf>

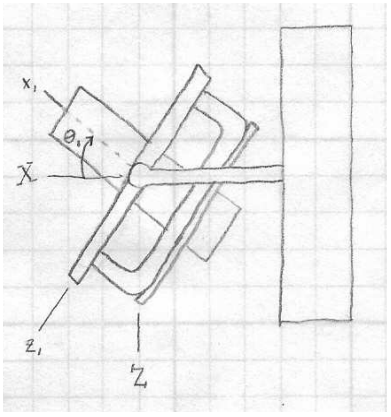
Visioli, A. and A. Piazzi. "On the Use of Dynamic-inversion for the Improvement of PID Control." In *16th IFAC World Congress on Automatic Control 2005, Prague CZ.*

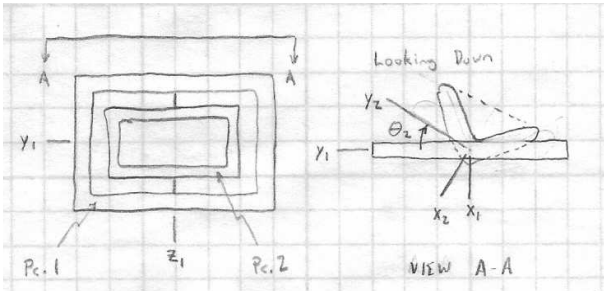
Yan, L. and C. James Li. "Robot Learning Control Based on Recurrent Neural Network Inverse Model." In *Journal of Robotic Systems* Vol. 14, No. 3 (1997): 199-212.

APPENDICES

APPENDIX A: Complete Derivation of Equations of Motion

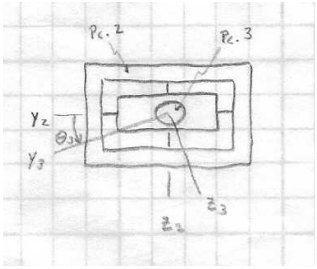
Moments of inertia defined as: $I_{\underline{\text{Pc. \#}}\ \underline{\text{Axis}}}$ therefore, I_{3z} would be z-axis of pc. 3.





$$\begin{Bmatrix} \bar{I} \\ \bar{J} \\ \bar{K} \end{Bmatrix} = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ 0 & 1 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 \end{bmatrix} \begin{Bmatrix} \hat{i}_1 \\ \hat{j}_1 \\ \hat{k}_1 \end{Bmatrix}$$

$$\begin{Bmatrix} \hat{i}_1 \\ \hat{j}_1 \\ \hat{k}_1 \end{Bmatrix} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \hat{i}_2 \\ \hat{j}_2 \\ \hat{k}_2 \end{Bmatrix}$$



$$\begin{Bmatrix} \hat{i}_2 \\ \hat{j}_2 \\ \hat{k}_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_3 & -\sin\theta_3 \\ 0 & \sin\theta_3 & \cos\theta_3 \end{bmatrix} \begin{Bmatrix} \hat{i}_3 \\ \hat{j}_3 \\ \hat{k}_3 \end{Bmatrix}$$

Calculate Angular Velocities:

$$\bar{\omega}_1 = \dot{\theta}_1 \hat{j}_1$$

$$\bar{\omega}_2 = \dot{\theta}_1 \hat{j}_1 + \dot{\theta}_2 \hat{k}_2$$
$$\bar{\omega}_2 = \dot{\theta}_1 \left(\sin\theta_2 \hat{i}_2 + \cos\theta_2 \hat{j}_2\right) + \dot{\theta}_2 \hat{k}_2$$
$$\bar{\omega}_2 = \dot{\theta}_1 \sin\theta_2 \hat{i}_2 + \dot{\theta}_1 \cos\theta_2 \hat{j}_2 + \dot{\theta}_2 \hat{k}_2$$

$$\bar{\omega}_3 = \dot{\theta}_1 \hat{j}_1 + \dot{\theta}_2 \hat{k}_2 + \dot{\theta}_3 \hat{i}_3$$
$$\bar{\omega}_3 = \dot{\theta}_1 \sin\theta_2 \hat{i}_2 + \dot{\theta}_1 \cos\theta_2 \hat{j}_2 + \dot{\theta}_2 \hat{k}_2 + \dot{\theta}_3 \hat{i}_3$$
$$\bar{\omega}_3 = \dot{\theta}_1 \sin\theta_2 \hat{i}_3 + \dot{\theta}_1 \cos\theta_2 \left(\cos\theta_3 \hat{j}_3 - \sin\theta_3 \hat{k}_3\right) + \dot{\theta}_2 \left(\sin\theta_3 \hat{j}_3 + \cos\theta_3 \hat{k}_3\right) + \dot{\theta}_3 \hat{i}_3$$
$$\bar{\omega}_3 = \left(\dot{\theta}_3 + \dot{\theta}_1 \sin\theta_2\right)\hat{i}_3 + \left(\dot{\theta}_1 \cos\theta_2 \cos\theta_3 + \dot{\theta}_2 \sin\theta_3\right)\hat{j}_3 + \left(\dot{\theta}_2 \cos\theta_3 - \dot{\theta}_1 \cos\theta_2 \sin\theta_3\right)\hat{k}_3$$

Determine Kinetic Energy:

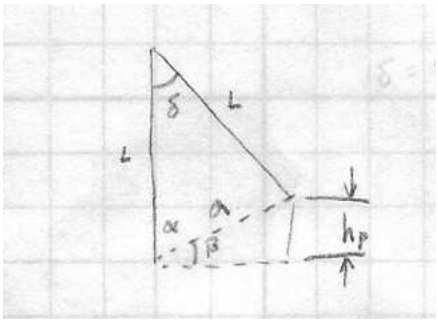
$$T = \frac{1}{2}\sum_{h=1}^{3} \boldsymbol{\omega}^T \boldsymbol{I} \boldsymbol{\omega}$$

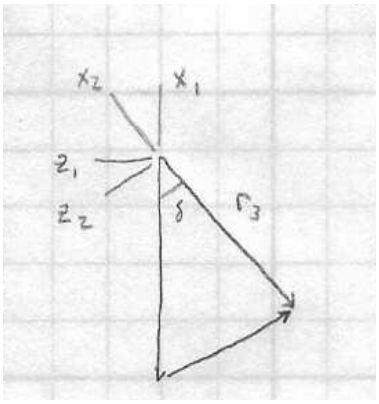
$$T_1 = \frac{1}{2}\left\{ \begin{bmatrix} 0 & \dot{\theta}_1 & 0 \end{bmatrix} \begin{bmatrix} I_{1x} & 0 & 0 \\ 0 & I_{1y} & 0 \\ 0 & 0 & I_{1z} \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta}_1 \\ 0 \end{bmatrix} \right\}$$
$$T_1 = \frac{1}{2}\dot{\theta}_1^{\,2} I_{1y}$$

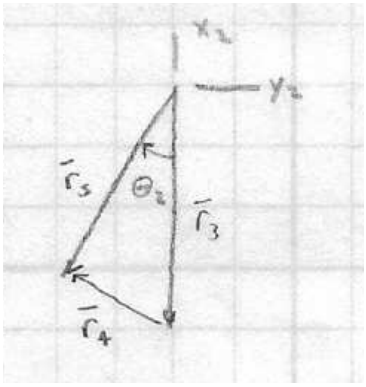
$$T_2 = \frac{1}{2}\left\{ \begin{bmatrix} \dot{\theta}_1 \sin\theta_2 & \dot{\theta}_1 \cos\theta_2 & \dot{\theta}_2 \end{bmatrix} \begin{bmatrix} I_{2x} & 0 & 0 \\ 0 & I_{2y} & 0 \\ 0 & 0 & I_{2z} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \sin\theta_2 \\ \dot{\theta}_1 \cos\theta_2 \\ \dot{\theta}_2 \end{bmatrix} \right\}$$
$$T_2 = \frac{1}{2}\left\{\dot{\theta}_1^{\,2} \sin^2\theta_2 I_{2x} + \dot{\theta}_1^{\,2} \cos^2\theta_2 I_{2y} + \dot{\theta}_2^{\,2} I_{2z}\right\}$$

$$T_3 = \frac{1}{2}\{\left(\dot{\theta}_3 + \dot{\theta}_1 \sin\theta_2\right)^2 I_{3x} + \left(\dot{\theta}_1 \cos\theta_2 \cos\theta_3 + \dot{\theta}_2 \sin\theta_3\right)^2 I_{3y} +$$
$$\left(\dot{\theta}_2 \cos\theta_3 - \dot{\theta}_1 \cos\theta_2 \sin\theta_3\right)^2 I_{3z}\}$$

Determine Potential Energy:







$$\delta = \frac{\pi}{2} - \theta_1$$

$$\alpha = \frac{\pi}{4} + \frac{1}{2}\theta_1$$

$$\beta = \frac{\pi}{4} - \frac{1}{2}\theta_1$$

$$\begin{bmatrix} \hat{i}_2 \\ \hat{j}_2 \\ \hat{k}_2 \end{bmatrix} = \begin{bmatrix} \cos\delta & 0 & -\sin\delta \\ 0 & 1 & 0 \\ \sin\delta & 0 & \cos\delta \end{bmatrix} \begin{bmatrix} \hat{i}_1 \\ \hat{j}_1 \\ \hat{k}_1 \end{bmatrix}$$

$$h_t = h_p + h_y$$

$$a^2 = L^2 + L^2 - 2L^2\cos\delta$$
$$a = \sqrt{2}L(1-\cos\delta)^{1/2}$$
$$h_p = a\sin\beta$$

$$\bar{r}_5 = \bar{r}_3 + \bar{r}_4 \therefore \bar{r}_4 = \bar{r}_5 - \bar{r}_3$$
$$\bar{r}_4 = L\left(-\cos\theta_2\hat{i}_2 - \sin\theta_2\hat{j}_2\right) - \left(-L\hat{i}_2\right)$$
$$\bar{r}_4 = L\left(-\cos\theta_2\left(\cos\delta\,\hat{i}_1 - \sin\delta\,\hat{k}_1\right) - \sin\theta_2\hat{j}_1\right) + L\left(\cos\delta\,\hat{i}_1 - \sin\delta\,\hat{k}_1\right)$$
$$h_y = -L\cos\theta_2\cos\delta + L\cos\delta$$

$$h_t = a\sin\beta + L\cos\delta(1-\cos\theta_2)$$
$$h_t = \sqrt{2}L\left(1-\cos\left(\frac{\pi}{2}-\theta_1\right)\right)^{1/2}\sin\left(\frac{\pi}{4}-\frac{1}{2}\theta_1\right) + L\cos\left(\frac{\pi}{2}-\theta_1\right)(1-\cos\theta_2)$$

$$V = mgh_t$$

Lagrange's Equations: $$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial T}{\partial \boldsymbol{q}} + \frac{\partial V}{\partial \boldsymbol{q}} = \boldsymbol{f}$$

For θ_1:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{\theta}_1}\right) - \frac{\partial T}{\partial \theta_1} + \frac{\partial V}{\partial \theta_1} = \Gamma_1$$

$$\Gamma_1 = \{\frac{\mathrm{d}}{\mathrm{d}t}[\dot{\theta}_1 I_{1y} + \dot{\theta}_1\sin^2\theta_2 I_{2x} + \dot{\theta}_1\cos^2\theta_2 I_{2y} + \left(\dot{\theta}_3 + \dot{\theta}_1\sin\theta_2\right)\sin\theta_2 I_{3x} +$$
$$\left(\dot{\theta}_1\cos\theta_2\cos\theta_3 + \dot{\theta}_2\sin\theta_3\right)\cos\theta_2\cos\theta_3 I_{3y} - \left(\dot{\theta}_2\cos\theta_3 - \dot{\theta}_1\cos\theta_2\sin\theta_3\right)\cos\theta_2\sin\theta_3 I_{3z}]\} -$$
$$\left\{mg\left[\left(-\sqrt{2}L/2\right)(1-\cos\delta)^{-1/2}\sin\delta\sin\beta - \left(\sqrt{2}L/2\right)(1-\cos\delta)^{1/2}\cos\beta + L(1-\cos)\sin\delta\right]\right\}$$

For θ_2:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{\theta}_2}\right)-\frac{\partial T}{\partial \theta_2}+\frac{\partial V}{\partial \theta_2}=\Gamma_2$$

$$\begin{aligned}\Gamma_2=\{&\frac{d}{dt}[\dot{\theta}_2 I_{2z}+\left(\dot{\theta}_1\cos\theta_2\cos\theta_3+\dot{\theta}_2\sin\theta_3\right)\sin\theta_3 I_{3y}+\left(\dot{\theta}_2\cos\theta_3-\dot{\theta}_1\cos\theta_2\sin\theta_3\right)\cos\theta_3 I_{3z}]\}-\\
&\{\dot{\theta}_1^{\,2}\cos\theta_2\sin\theta_2 I_{2x}-\dot{\theta}_1^{\,2}\cos\theta_2\sin\theta_2 I_{2y}+\left(\dot{\theta}_3+\dot{\theta}_1\sin\theta_2\right)\dot{\theta}_1\cos\theta_2 I_{3x}-\\
&\left(\dot{\theta}_1\cos\theta_2\cos\theta_3+\dot{\theta}_2\sin\theta_3\right)\dot{\theta}_1\sin\theta_2\cos\theta_3 I_{3y}+\left(\dot{\theta}_2\cos\theta_3-\dot{\theta}_1\cos\theta_2\sin\theta_3\right)\dot{\theta}_1\sin\theta_2\sin\theta_3 I_{3z}\}+\\
&\{mgL\sin\theta_2\cos\delta\}\end{aligned}$$

For θ_3:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{\theta}_3}\right)-\frac{\partial T}{\partial \theta_3}+\frac{\partial V}{\partial \theta_3}=\Gamma_3$$

$$\begin{aligned}\Gamma_3=\{&\frac{d}{dt}[\left(\dot{\theta}_3+\dot{\theta}_1\sin\theta_2\right)I_{3x}]\}-\{-\left(\dot{\theta}_1\cos\theta_2\cos\theta_3+\dot{\theta}_2\sin\theta_3\right)\dot{\theta}_1\cos\theta_2\sin\theta_3 I_{3y}+\\
&\left(\dot{\theta}_1\cos\theta_2\cos\theta_3+\dot{\theta}_2\sin\theta_3\right)\dot{\theta}_2\cos\theta_3 I_{3y}-\left(\dot{\theta}_2\cos\theta_3-\dot{\theta}_1\cos\theta_2\sin\theta_3\right)\dot{\theta}_2\sin\theta_3 I_{3z}-\\
&\left(\dot{\theta}_2\cos\theta_3-\dot{\theta}_1\cos\theta_2\sin\theta_3\right)\dot{\theta}_1\cos\theta_2\cos\theta_3 I_{3z}\}\end{aligned}$$

Combine the $\ddot{q}$ terms to find the mass matrix, ***M***, the $\dot{q}_i\dot{q}_j$ terms and the potential terms to find the Coriolis vector, ***h***, to give the equation form:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}+\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}})=\boldsymbol{f}$$

$$M(q)=\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{12} & M_{22} & M_{23} \\ M_{13} & M_{23} & M_{33} \end{bmatrix}$$

$$M_{11} = I_{1y} + \sin^2\theta_2\left(I_{2x} + I_{3x}\right) + \cos^2\theta_2\left(I_{2y} + \cos^2\theta_3 I_{3y} + \sin^2\theta_3 I_{3z}\right)$$
$$M_{22} = I_{2z} + \sin^2\theta_3 I_{3y} + \cos^2\theta_3 I_{3z}$$
$$M_{33} = I_{3x}$$
$$M_{12} = \cos\theta_2 \cos\theta_3 \sin\theta_3\left(I_{3y} - I_{3z}\right)$$
$$M_{13} = \sin\theta_2 I_{3x}$$
$$M_{23} = 0$$

$$h(q,\dot{q})=\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$h_1 = \dot{\theta}_2{}^2 \sin\theta_2 \cos\theta_3 \sin\theta_3\left(I_{3z} - I_{3y}\right) +$$
$$2\dot{\theta}_1\dot{\theta}_2 \cos\theta_2 \sin\theta_2\left(I_{2x} - I_{2y} + I_{3x} - \left(\cos^2\theta_3 I_{3y} + \sin^2\theta_3 I_{3z}\right)\right) +$$
$$2\dot{\theta}_1\dot{\theta}_3 \cos^2\theta_2 \cos\theta_3 \sin\theta_3\left(I_{3z} - I_{3y}\right) +$$
$$\dot{\theta}_2\dot{\theta}_3 \cos\theta_2\left(I_{3x} + \cos^2\theta_3\left(I_{3y} - I_{3z}\right) + \sin^2\theta_3\left(I_{3z} - I_{3y}\right)\right) +$$
$$mgL\left(\left(-\sqrt{2}/2\right)(1-\cos\delta)^{-1/2}\sin\delta\sin\beta - \left(\sqrt{2}/2\right)(1-\cos\delta)^{1/2}\cos\beta + (1-\cos\theta_2)\sin\delta\right)$$

$$h_2 = -\dot{\theta}^2 \cos\theta_2 \sin\theta_2\left(I_{2x} - I_{2y} + I_{3x} - \cos^2\theta_3 I_{3y} - \sin^2\theta_3 I_{3z}\right) -$$
$$\dot{\theta}_1\dot{\theta}_3 \cos\theta_2\left(I_{3x} + \sin^2\theta_3\left(I_{3y} - I_{3z}\right) + \cos^2\theta_3\left(I_{3z} - I_{3y}\right)\right) +$$
$$2\dot{\theta}_2\dot{\theta}_3 \cos\theta_3 \sin\theta_3\left(I_{3y} - I_{3z}\right) +$$
$$mgL\sin\theta_2\cos\delta$$

$$h_3 = \dot{\theta}_1{}^2 \cos^2\theta_2 \cos\theta_3 \sin\theta_3\left(I_{3y} - I_{3z}\right) + \dot{\theta}_2{}^2 \cos\theta_3 \sin\theta_3\left(I_{3z} - I_{3y}\right) +$$
$$\dot{\theta}_1\dot{\theta}_2 \cos\theta_2\left(I_{3x} + \cos^2\theta_3\left(I_{3z} - I_{3y}\right) + \sin^2\theta_3\left(I_{3y} - I_{3z}\right)\right)$$

APPENDIX B: Complete Derivation of Dynamic-inversion Controller

Kinetic Energy in Generalized Velocities

$$T(\boldsymbol{q},\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} = \frac{1}{2} M_{ij}\dot{q}_i\dot{q}_j$$

Apply Lagrange's Equations to Determine Mass Matrix and Coriolis Terms

$$L(\boldsymbol{q},\dot{\boldsymbol{q}}) = T(\boldsymbol{q},\dot{\boldsymbol{q}}) - V(\boldsymbol{q})$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} = f_k$$

$$\frac{d}{dt}\left(M_{ki}\dot{q}_i\right) - \frac{1}{2}\frac{\partial M_{ij}}{\partial q_k}\dot{q}_i\dot{q}_j + \frac{\partial V}{\partial q_k} = f_k$$

$$M_{ki}\ddot{q}_i + \frac{\partial M_{ki}}{\partial q_j}\dot{q}_j\dot{q}_i - \frac{1}{2}\frac{\partial M_{ij}}{\partial q_k}\dot{q}_i\dot{q}_j + \frac{\partial V}{\partial q_k} = f_k$$

Equations of Motion in Index and Vector Notation

$$M_{ki}\ddot{q}_i + h_k = f_k \quad \text{where} \quad h_k = \left(\frac{\partial M_{ki}}{\partial q_j} - \frac{1}{2}\frac{\partial M_{ij}}{\partial q_k}\right)\dot{q}_i\dot{q}_j + \frac{\partial V}{\partial q_k}$$

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{f}$$

Selected Lyapunov Function

$$Q = \frac{1}{2}(\boldsymbol{q}-\boldsymbol{q}_d)^T(\boldsymbol{q}-\boldsymbol{q}_d) + \frac{1}{2}(\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_d)^T(\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_d)$$

Find Derivative to Determine if Global Asymptotically Stable $\left(\dot{Q} \le 0\right)$

$$\dot{Q} = (\boldsymbol{q} - \boldsymbol{q}_d)^T (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) + (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d)^T (\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_d)$$

$$\dot{Q} = (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d)^T (\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_d + \boldsymbol{q} - \boldsymbol{q}_d)$$

$$\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_d + \boldsymbol{q} - \boldsymbol{q}_d = -(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) \qquad \text{(Eq. B1)}$$

$$\ddot{\boldsymbol{q}}_d = \boldsymbol{M}(\boldsymbol{q}_d)^{-1} (\boldsymbol{f}_d - \boldsymbol{h}(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d)) \qquad \text{(Eq. B2)}$$

Substitute Desired Motion into the Equations of Motion

$$\boldsymbol{f}_d = \boldsymbol{M}(\boldsymbol{q}_d)\ddot{\boldsymbol{q}}_d + \boldsymbol{h}(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d) = \boldsymbol{M}_d \ddot{\boldsymbol{q}}_d + \boldsymbol{h}_d$$

$$\ddot{\boldsymbol{q}}_d = \boldsymbol{M}_d^{-1} (\boldsymbol{f}_d - \boldsymbol{h}_d) \qquad \text{(Eq. B3)}$$

Substituting Eq. B2 and Eq B3 into Eq. B1

$$\boldsymbol{M}^{-1}(\boldsymbol{f} - \boldsymbol{h}) - \boldsymbol{M}_d^{-1}(\boldsymbol{f}_d - \boldsymbol{h}_d) + \boldsymbol{q} - \boldsymbol{q}_d = -(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d)$$

$$\boldsymbol{f} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q}_d)^{-1} \boldsymbol{f}_d + \left(\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{M}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q}_d)^{-1} \boldsymbol{h}(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d)\right) - \boldsymbol{M}(\boldsymbol{q})[(\boldsymbol{q} - \boldsymbol{q}_d) + (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d)]$$

APPENDIX C: PID and Dynamic-inversion Controller Program

```
global g d I1 I2 I3 m dt time_intervals ang_com angrate_com
angaccel_com Kp Ki Kd ang_com_temp angrate_com_temp

% Variables for the surrounding system
g = 9.81;     %m/sec^2

% Variables for the Flight Table Geometry
d = 0.20;      %m - distance of component offset from point of rotation
I1x = 57.6257;                   %kg-m^2
I1y = 20.4346;                   %kg-m^2
I1z = 37.8028;                   %kg-m^2
I2x = 6.6928;                    %kg-m^2
I2y = 3.7280;                    %kg-m^2
I2z = 3.0442;                    %kg-m^2
I3x = 2.8279;                    %kg-m^2
I3y = 6.7884;                    %kg-m^2
I3z = 6.7884;                    %kg-m^2
I1 = [I1x 0 0;0 I1y 0;0 0 I1z];
I2 = [I2x 0 0;0 I2y 0;0 0 I2z];
I3 = [I3x 0 0;0 I3y 0;0 0 I3z];
m = 91.8523;        %kg - mass of component 2 (yaw)
f = [0;0;0];   %set the initial control = 0 for all components

% Time Characteristics for Simulation
i = 1;                   %Counter for the "for-loop"
iter = 1;
ti(i)=0.0;               %sec - initial start time of the system
tf = 5.0;                %sec - final time of the system
dt = 0.01;                %sec - time steps for RK44 integration
time_intervals = 1;      %number of steps till control is adjusted
tsteps=(tf-ti)/dt;       %total number of iterations

% Initial Position and Angular Rates of Gimbals (Commanded and Actual)
ang_com_temp = [0.1;0.1;0.2];
angrate_com_temp = [0;0;0];
[commands] = commanded_values(i,0);
ang_com  = commands(1:3);
angrate_com = commands(4:6);
angaccel_com = commands(7:9);
Ang_Com(:,i) = ang_com;
AngRate_Com(:,i) = angrate_com;
the1_0 = 0.4;          %radians - Initial Position, Component 1
the2_0 = 0.4;          %radians - Initial Position, Component 2
the3_0 = 0.4;          %radians - Initial Position, Component 3
u_0 = angrate_com(1);  %radians/sec - Initial Velocity, Component 1
v_0 = angrate_com(2);  %radians/sec - Initial Velocity, Component 2
w_0 = angrate_com(3);  %radians/sec - Initial Velocity, Component 3
the1(i) = the1_0;      %Angle of Component 1, for graphing
the2(i) = the2_0;      %Angle of Component 2, for graphing
the3(i) = the3_0;      %Angle of Component 3, for graphing
```

```
u(i) = u_0;            %Angular Velocity of Component 1, for graphing
v(i) = v_0;            %Angular Velocity of Component 2, for graphing
w(i) = w_0;            %Angular Velocity of Component 3, for graphing

% PID Augmented Integration Term
z = [0;0;0];

% Defining State Vector for Various Controllers
controller = input('Enter "1" for PID, "2" for DI:  ');
if controller == 1
    x = [the1(i);the2(i);the3(i);u(i);v(i);w(i);z];
elseif controller == 2
    x = [the1(i);the2(i);the3(i);u(i);v(i);w(i)];
end;

% Gains
if controller == 1
    Kp = 400;
    Ki = 400;
    Kd = 100;
elseif controller == 2
    Kp = 400;
    Ki = 0;
    Kd = 100;
end;

%Graphing Parameters
time(i) = 0.0;
ang_1(i) = x(1);
ang_2(i) = x(2);
ang_3(i) = x(3);
control_1(i) = f(1);
control_2(i) = f(2);
control_3(i) = f(3);
error_mag(i) = 0;
error_norm_total = 0;

% Integration Process for the RK44
for i = 1:tsteps
    ti(i+1) = ti(i) + dt;

    k1 = dt*deriv_con(controller,x,f);
    k2 = dt*deriv_con(controller,x+k1/2,f);
    k3 = dt*deriv_con(controller,x+k2/2,f);
    k4 = dt*deriv_con(controller,x+k3,f);

    x = x + (1/6)*(k1+2*(k2+k3)+k4);

    [commands] = commanded_values(i,ti(i+1));
    ang_com = commands(1:3);
    angrate_com = commands(4:6);
    angaccel_com = commands(7:9);
    Ang_Com(:,i+1) = ang_com;
    AngRate_Com(:,i+1) = angrate_com;
```

```
    [f,error_norm] = control_develop(controller,i,x);

    error_mag(i+1) = error_norm;
    error_norm_total = error_norm_total + error_norm;
    ang_1(i+1) = x(1);
    ang_2(i+1) = x(2);
    ang_3(i+1) = x(3);
    control_1(i+1) = f(1);
    control_2(i+1) = f(2);
    control_3(i+1) = f(3);
    time(i+1) = ti(i+1);
end;

error_norm_avg = error_norm_total/tsteps


**************************************************************

function [k]=Controller_Main(controller,x,f)

global g d I1 I2 I3 m dt ang_com angrate_com

I1x = I1(1,1);
I1y = I1(2,2);
I1z = I1(3,3);
I2x = I2(1,1);
I2y = I2(2,2);
I2z = I2(3,3);
I3x = I3(1,1);
I3y = I3(2,2);
I3z = I3(3,3);

the1 = x(1);
the2 = x(2);
the3 = x(3);
t1d = x(4);
t2d = x(5);
t3d = x(6);

M11 = I1y + (I2x+I3x)*(sin(the2))^2 + (I2y + I3y*(cos(the3))^2 +
I3z*(sin(the3))^2)*(cos(the2))^2;
M12 = (I3y-I3z)*cos(the2)*cos(the3)*sin(the3);
M21 = M12;
M13 = I3x*sin(the2);
M22 = I3y*(sin(the3))^2 + I3z*(cos(the3))^2 + I2z;
M23 = 0;
M31 = M13;
M32 = M23;
M33 = I3x;
M = [M11 M12 M13;
     M21 M22 M23;
     M31 M32 M33];
```

```
h1 = (t2d^2)*sin(the2)*cos(the3)*sin(the3)*(I3z-I3y)...
     + t1d*t2d*2*cos(the2)*sin(the2)*(I2x-I2y+I3x
     -(I3y*(cos(the3))^2+I3z*(sin(the3))^2))...
     + t1d*t3d*(-2*(I3y-I3z)*cos(the3)*sin(the3)*(cos(the2))^2)...
     + t2d*t3d*cos(the2)*(I3x+I3y*((cos(the3))^2
     -(sin(the3))^2)+I3z*((sin(the3))^2-(cos(the3))^2))...
     + m*g*(-(sqrt(2)/2)*d*(((1-cos((pi/2)-the1))^(-1/2))*sin((pi/2)
     -the1)*sin((pi/4)-0.5*the1)+sqrt(1-cos((pi/2)-the1))*cos((pi/4)
     -0.5*the1))+d*(1-cos(the2))*sin((pi/2)-the1));
h2 = (t1d^2)*cos(the2)*sin(the2)*(I2y-I2x
     -I3x+I3y*(cos(the3))^2+I3z*(sin(the3))^2)...
     + t1d*t3d*cos(the2)*(I3y*((cos(the3))^2
     -(sin(the3))^2)+I3z*((sin(the3))^2-(cos(the3))^2)-I3x)...
     + t2d*t3d*2*(I3y-I3z)*cos(the3)*sin(the3)...
     + m*g*d*sin(the2)*cos((pi/2)-the1);
h3 = (t1d^2)*(I3y-I3z)*(cos(the2))^2*cos(the3)*sin(the3)...
     + (t2d^2)*(I3z-I3y)*cos(the3)*sin(the3)...
     + t1d*t2d*cos(the2)*(I3x+I3y*((sin(the3))^2
     -(cos(the3))^2)+I3z*((cos(the3))^2-(sin(the3))^2));

h = [h1;h2;h3];

ang_rate_dot = M\(f - h);

if controller == 1
    ang = [the1;the2;the3];
    error = ang_com - ang;
    int_error = error;
    k = [t1d;t2d;t3d;ang_rate_dot;int_error];
elseif controller == 2
    k = [t1d;t2d;t3d;ang_rate_dot];
end;


**************************************************************

function [commands]=Controller_Main(i,t)

global dt ang_com_temp angrate_com_temp

% Commanded Angles and Rates for Sinusoidal Motion
% ang_com = [0.5*sin(t);0.5*sin(t);0.5*sin(t)];
% angrate_com = [0.5*cos(t);0.5*cos(t);0.5*cos(t)];
% angaccel_com = [-0.5*sin(t);-0.5*sin(t);-0.5*sin(t)];

% Commanded Angles and Rates for Constant Motion
ang_com = [0.1;0.1;0.2];
angrate_com = (ang_com - ang_com_temp)/dt;
angaccel_com = (angrate_com - angrate_com_temp)/dt;

ang_com_temp = ang_com;
angrate_com_temp = angrate_com;
% Step Function after a Given Time Period
```

```
% if i>=500
%     ang_com = [0.2;0.2;0.5];
% end;

commands = [ang_com;angrate_com;angaccel_com];


************************************************************

function [ftemp,error_norm]=Controller_Main(controller,i,x)

global ang_com angrate_com Kp Ki Kd

ang = [x(1);x(2);x(3)];
vel = [x(4);x(5);x(6)];
angle_error = ang_com - ang;
if i >= 0
    error_norm = norm(angle_error);
else error_norm = 0;
end;

if controller == 1
    e = ang_com - ang;
    de = angrate_com - vel;
    z_vec = [x(7);x(8);x(9)];
    ftemp = Kp*e + Ki*z_vec + Kd*de;

elseif controller == 2
    [qr,qdr,Mr,hr,fr,q,qd,M,h]=model_components(ang,vel);
    ftemp = M*(Mr\fr) + (h - M*(Mr\hr)) - M*(Kp*(q-qr)+Kd*(qd-qdr));
end;


************************************************************

function [qr,qdr,Mr,hr,fr,q,qd,M,h]=control_develop(the,vel)

global g d I1 I2 I3 m dt ang_com angrate_com angaccel_com

I1x = I1(1,1);
I1y = I1(2,2);
I1z = I1(3,3);
I2x = I2(1,1);
I2y = I2(2,2);
I2z = I2(3,3);
I3x = I3(1,1);
I3y = I3(2,2);
I3z = I3(3,3);
```

```
%%%%%%%%%%%%%%%%%%%%%%  Reference Values  %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Positional Variables
the1r = ang_com(1);
the2r = ang_com(2);
the3r = ang_com(3);
qr = [the1r;the2r;the3r];

% Velocity Variables
t1dr = angrate_com(1);
t2dr = angrate_com(2);
t3dr = angrate_com(3);
qdr = [t1dr;t2dr;t3dr];

% Acceleration Variables
t1ddr = angaccel_com(1);
t2ddr = angaccel_com(2);
t3ddr = angaccel_com(3);
tddr = [t1ddr;t2ddr;t3ddr];

% Returned Variables
M11 = I1y + (I2x+I3x)*(sin(the2r))^2 + (I2y + I3y*(cos(the3r))^2 +
      I3z*(sin(the3r))^2)*(cos(the2r))^2;
M12 = (I3y-I3z)*cos(the2r)*cos(the3r)*sin(the3r);
M21 = M12;
M13 = I3x*sin(the2r);
M22 = I3y*(sin(the3r))^2 + I3z*(cos(the3r))^2 + I2z;
M23 = 0;
M31 = M13;
M32 = M23;
M33 = I3x;
Mr = [M11 M12 M13;
      M21 M22 M23;
      M31 M32 M33];

h1 = (t2dr^2)*sin(the2r)*cos(the3r)*sin(the3r)*(I3z-I3y)...
     + t1dr*t2dr*2*cos(the2r)*sin(the2r)*(I2x-I2y+I3x-
       (I3y*(cos(the3r))^2+I3z*(sin(the3r))^2))...
     + t1dr*t3dr*(-2*(I3y-I3z)*cos(the3r)*sin(the3r)*(cos(the2r))^2)...
     + t2dr*t3dr*cos(the2r)*(I3x+I3y*((cos(the3r))^2-
       (sin(the3r))^2)+I3z*((sin(the3r))^2-(cos(the3r))^2))...
     + m*g*(-(sqrt(2)/2)*d*(((1-cos((pi/2)-the1r))^(-1/2))*sin((pi/2)
     -the1r)*sin((pi/4)-0.5*the1r)+sqrt(1-cos((pi/2)
     -the1r))*cos((pi/4)-0.5*the1r))+d*(1-cos(the2r))*sin((pi/2)
     -the1r));
h2 = (t1dr^2)*cos(the2r)*sin(the2r)*(I2y-I2x
     -I3x+I3y*(cos(the3r))^2+I3z*(sin(the3r))^2)...
     + t1dr*t3dr*cos(the2r)*(I3y*((cos(the3r))^2
     -(sin(the3r))^2)+I3z*((sin(the3r))^2-(cos(the3r))^2)-I3x)...
     + t2dr*t3dr*2*(I3y-I3z)*cos(the3r)*sin(the3r)...
     + m*g*d*sin(the2r)*cos((pi/2)-the1r);
h3 = (t1dr^2)*(I3y-I3z)*(cos(the2r))^2*cos(the3r)*sin(the3r)...
     + (t2dr^2)*(I3z-I3y)*cos(the3r)*sin(the3r)...
     + t1dr*t2dr*cos(the2r)*(I3x+I3y*((sin(the3r))^2
```

```
     -(cos(the3r))^2)+I3z*((cos(the3r))^2-(sin(the3r))^2));

hr = [h1;h2;h3];
fr = Mr*tddr + hr;

%%%%%%%%%%%%%%%%%%  Actual Values  %%%%%%%%%%%%%%%%%%%
% Positional Variables
the1 = the(1);
the2 = the(2);
the3 = the(3);
q = [the1;the2;the3];

% Velocity Variables
t1d = vel(1);
t2d = vel(2);
t3d = vel(3);
qd = [t1d;t2d;t3d];

% Returned Variables
M11 = I1y + (I2x+I3x)*(sin(the2))^2 + (I2y + I3y*(cos(the3))^2 +
      I3z*(sin(the3))^2)*(cos(the2))^2;
M12 = (I3y-I3z)*cos(the2)*cos(the3)*sin(the3);
M21 = M12;
M13 = I3x*sin(the2);
M22 = I3y*(sin(the3))^2 + I3z*(cos(the3))^2 + I2z;
M23 = 0;
M31 = M13;
M32 = M23;
M33 = I3x;
M = [M11 M12 M13;
     M21 M22 M23;
     M31 M32 M33];

h1 = (t2d^2)*sin(the2)*cos(the3)*sin(the3)*(I3z-I3y)...
     + t1d*t2d*2*cos(the2)*sin(the2)*(I2x-I2y+I3x
     -(I3y*(cos(the3))^2+I3z*(sin(the3))^2))...
     + t1d*t3d*(-2*(I3y-I3z)*cos(the3)*sin(the3)*(cos(the2))^2)...
     + t2d*t3d*cos(the2)*(I3x+I3y*((cos(the3))^2
     -(sin(the3))^2)+I3z*((sin(the3))^2-(cos(the3))^2))...
     + m*g*(-(sqrt(2)/2)*d*(((1-cos((pi/2)-the1))^(-1/2))*sin((pi/2)
     -the1)*sin((pi/4)-0.5*the1)+sqrt(1-cos((pi/2)-the1))*cos((pi/4)
     -0.5*the1))+d*(1-cos(the2))*sin((pi/2)-the1));
h2 = (t1d^2)*cos(the2)*sin(the2)*(I2y-I2x
     -I3x+I3y*(cos(the3))^2+I3z*(sin(the3))^2)...
     + t1d*t3d*cos(the2)*(I3y*((cos(the3))^2
     -(sin(the3))^2)+I3z*((sin(the3))^2-(cos(the3))^2)-I3x)...
     + t2d*t3d*2*(I3y-I3z)*cos(the3)*sin(the3)...
     + m*g*d*sin(the2)*cos((pi/2)-the1);
h3 = (t1d^2)*(I3y-I3z)*(cos(the2))^2*cos(the3)*sin(the3)...
     + (t2d^2)*(I3z-I3y)*cos(the3)*sin(the3)...
     + t1d*t2d*cos(the2)*(I3x+I3y*((sin(the3))^2
     -(cos(the3))^2)+I3z*((cos(the3))^2-(sin(the3))^2));
h = [h1;h2;h3];
```