ABSTRACT

REEDY, JACOB DAVID. Proof of Concept for Novel Method of Quantifying Stability Margins for Non-Linear Phase-plane Controllers. (Under the direction of Dr. Andre P. Mazzoleni.)

This thesis proposes an original method for quantifying the stability of a phase-plane controller called the Time Domain Analysis of Phase-Plane Stability, or TDAPPS. Phase-plane controllers are a type of on-off, or ‘bang-bang,’ control system in which the vehicle’s reaction control system is either active or inactive. It is difficult to come up with gain and phase margins or otherwise determine the stability boundaries for a non-linear system. Current methods of analysis are abstract, time-consuming, and yield a gain margin in the frequency domain. The proposed TDAPPS method does not require in-depth knowledge of non-linear analysis tools and provides gain and phase margins in the time domain. This thesis describes how a phase-plane controller works, compares the proposed method to current practices, and tests its application to the phase-plane control system for a space vehicle reminiscent of the Apollo capsule. The results of this test are then confirmed via simulation testing to prove that the TDAPPS method is a valuable analysis tool for non-linear control systems.

Keywords: Phase-plane controller, Non-linear controls, Time domain, Stability analysis
Proof of Concept for Novel Method of Quantifying Stability Margins for Non-Linear Phase-plane Controllers.

by
Jacob David Reedy

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science Aerospace Engineering

Raleigh, North Carolina 2018

APPROVED BY:

______________________________  ______________________________
Dr. Andre P. Mazzoleni  Dr. Larry M. Silverberg
Chair of Advisory Committee

______________________________
Dr. Matthew Bryant
DEDICATION

To my wife, Sarah, and parents, Marie and Dave, for constant love and support of all kinds.
Without their encouragement and motivation, I would not be where I am today.
BIOGRAPHY

Jacob D.P. Reedy was born in Reading, Pennsylvania and moved to Charlotte, North Carolina at the age of ten. He attended North Carolina State University for his undergraduate education where he received a Bachelor’s Degree in Aerospace Engineering. To pursue his goal of attaining a career in the space industry, he immediately enrolled in the Master’s program at NC State. Under the guidance of Dr. Andre Mazzoleni, he studied advanced dynamics, researched methods of active trajectory control for high-altitude balloons, and developed a prototype Mars rover. Prior to receiving his Master’s Degree, he accepted a job with Adaptive Aerospace Group, Inc. in Hampton, VA. He is currently working there tackling projects such as simulating the entry, descent, and landing phases of manned space capsules, performing controller stability analyses, and conducting Monte Carlo analysis on small UAS. In his free time, he enjoys reading, playing games and sports with his family and friends, and closely following the commercial space industry.
ACKNOWLEDGMENTS

I have many people to profusely thank for many different forms of support and encouragement. I would like to start off by thanking my wife, Sarah, parents, Marie and Dave, and siblings Sam, Olivia, and Ben for their constant love, strength, and support. I must also profusely thank Dr. Mazzoleni for accepting me as a graduate student, mentoring me on several projects, and arming me with knowledge and experience that has already served me very well in my young career. Next, I owe a lot of gratitude to Keith Hoffler and Bruce Jackson of Adaptive Aerospace Group for providing me with a fantastic opportunity to start my career and mentoring me over the last few years. I would like to specifically thank Bruce Jackson for all the help in developing TDAPPS and editing this document.

I also owe a thank you to many of my friends that helped me survive graduate school. First, thank you to Chris Yoder for volunteering his time to help improve this thesis. I would like to specifically thank Chris Yoder, Sachin Kelkar, Rajmohan Waghela, Jon Gatlin, Andrew Davis, Michael Brueno, Keifer Ewing, and Zachary Tulbert for being great friends and supporters throughout my academic career.
TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ vi
LIST OF FIGURES ........................................................................................................ vii

1. Project Description .................................................................................................. 1
   2. The Phase-plane Controller ............................................................................... 3
      2.1 Background ................................................................................................... 3
      2.2 Description .................................................................................................. 4

3. TDAPPS Method .................................................................................................... 15
   3.1 Description ..................................................................................................... 15
   3.2 Comparison with Describing Function Method .............................................. 26
   3.3 Simulink® Simulation Description .................................................................. 28

4. TDAPPS Application ............................................................................................... 33
   4.1 Apply to generic re-entry capsule .................................................................... 33
      4.1.1 3-axis attitude control .............................................................................. 35
      4.1.2 RCS Jet cross-coupling ............................................................................ 44
   4.2 Discussion of TDAPPS results ...................................................................... 54

5. Conclusions ............................................................................................................ 64

6. References ............................................................................................................. 66

APPENDIX .................................................................................................................. 68
   Appendix A ........................................................................................................... 69
**LIST OF TABLES**

Table 3.1 Equations for each of the firing curves in terms of controller parameters................. 16

Table 3.2 Jet state flag value assignment based on the command from vehicle guidance.... 21

Table 4.1 Nominal Apollo Command Module mass properties at entry interface................. 34

Table 4.2 Initial Euler angle attitude and angular rates......................................................... 35

Table 4.3 RCS torque and corresponding angular acceleration magnitude in each axis........... 36

Table 4.4 Initial Euler angle attitude and angular rates for the cross-coupling simulation..... 44

Table 4.5 RCS jet thrust alignment unit vectors................................................................. 45

Table 4.6 RCS jet torque vectors......................................................................................... 46
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Space Shuttle phase-plane space</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Phase-plane space plot for a simple phase-plane controller</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Top – Total Error vs. time; Bottom – Thrust versus time</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Phase-plane plot with hysteresis</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>Top – Total Error vs. time; Bottom – Thrust versus time</td>
<td>10</td>
</tr>
<tr>
<td>2.6</td>
<td>Single axis phase-plane controller with an applied external torque</td>
<td>11</td>
</tr>
<tr>
<td>2.7</td>
<td>Phase-plane space with numbered intersection point to show a full analysis cycle</td>
<td>12</td>
</tr>
<tr>
<td>2.8</td>
<td>Unstable phase-plane controller</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Example of phase-plane space with key terminology labels</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>TDAPPS method flow chart</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>Visualize aid for describing curve projection</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>Classic control loop</td>
<td>27</td>
</tr>
<tr>
<td>3.5</td>
<td>Top level of <code>three_axis_phase_plane_exp.slx</code></td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Plant model for the three-axis phase plane controller</td>
<td>30</td>
</tr>
<tr>
<td>3.7</td>
<td>Top level of the phase plane controller subsystem</td>
<td>30</td>
</tr>
<tr>
<td>3.8</td>
<td>Splitting phase plane controller between upper and lower bounds</td>
<td>31</td>
</tr>
<tr>
<td>3.9</td>
<td>Phase plane controller upper level deadband logic model</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Depiction of CM body reference frame and general RCS jet locations</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>Roll axis phase plane plot with no RCS cross-coupling</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Total error and torque time histories for the roll axis</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>TDAPPS results for the roll axis</td>
<td>39</td>
</tr>
</tbody>
</table>
Figure 4.5  Pitch axis phase plane plot with no RCS cross-coupling............................... 41

Figure 4.6  TDAPPS results for the pitch axis................................................................. 42

Figure 4.7  Yaw axis phase plane plot with no RCS cross-coupling............................... 43

Figure 4.8  TDAPPS results for the yaw axis.................................................................. 43

Figure 4.9  Roll-axis phase plane response of vehicle with cross-coupling.................... 46

Figure 4.10 Roll-axis total error and torque time histories............................................. 47

Figure 4.11 TDAPPS analysis for roll-axis cross-coupling simulation........................... 48

Figure 4.12 Pitch-axis phase plane response of vehicle with cross-coupling.................... 49

Figure 4.13 Pitch-axis total error and torque time histories............................................ 50

Figure 4.14 TDAPPS analysis for pitch-axis cross-coupling simulation......................... 51

Figure 4.15 Yaw-axis phase plane response of vehicle with cross-coupling....................... 52

Figure 4.16 Yaw-axis total error and torque time histories............................................. 53

Figure 4.17 TDAPPS analysis for yaw-axis cross-coupling simulation.......................... 54

Figure 4.18 Roll axis phase plane plot with 0.5 seconds of delay.................................... 56

Figure 4.19 Roll axis total error and torque with 0.5 seconds of delay............................. 57

Figure 4.20 Yaw axis phase plane space from the doubled RCS torque simulation........... 58

Figure 4.21 Yaw axis total error and torque plots from the doubled RCS torque simulation................................................................. 59

Figure 4.22 Yaw axis phase plane space from the quartered RCS torque simulation......... 60

Figure 4.23 Yaw axis total error and torque plots from the quartered RCS torque simulation....................................................................................................................... 61

Figure 4.24 Yaw axis phase plane plot with gain of 0.3 applied to RCS torque................. 62
1. **Project Description**

A phase-plane controller is a type of ‘bang-bang controller’ often used to achieve spacecraft attitude control [1,2,3]. It is a non-linear control system because the vehicle’s actuators only have two states: on and off. For typical non-linear control systems, analysis is performed by developing an error-dependent linear model of the controller and determining remaining stability margin from the closed-loop gains at which the controller would become unstable. This method has been used to analyze phase-plane controllers in the past but it requires several assumptions, is abstract, time-consuming, and provides only a stability margin for gains in the frequency domain [1,2,3]. While the result is certainly valid, it is not immediately intuitive as to how the gain margin in the frequency domain will affect the actual dynamics of the vehicle. Nor does it provide insight into the effect of time/phase delay on the system. It also requires an in-depth understanding of the non-linear analysis method to successfully conduct in addition to interpreting the results.

This thesis project describes and examines an original method for determining the phase and gain margins of a closed-loop system using a phase-plane controller in the time domain. The time domain is more closely linked to the vehicle dynamics than the frequency domain, and it can be simulated and visualized with relative ease. The proposed method is called the Time Domain Analysis of Phase-Plane Stability (TDAPPS). The TDAPPS method is comprised of a programmed algorithm that projects whether the system is stable or not at any given point along its trajectory, and what the minimum time (phase) delay and/or change to the system gain would cause an unstable response. Ultimately, the result is a time history of phase and gain margins along a particular plant trajectory or maneuver. This makes it easier to identify problem areas where further analysis may be required. The goal of this research is to show that the TDAPPS method can provide valuable information on the stability of a spacecraft’s control system in all three axes of rotational
freedom. Included will be studies of simulations across several different controller parameters and target attitudes. This analysis would then aid engineers in designing and analyzing a system’s phase-plane controller.

To accomplish this goal, this paper provides a brief, historical background on the use of phase-plane controllers in general and on actual spacecraft and describe how a basic phase-plane controller operates. Included in this will be a single axis example that delves into some of the intricacies of this non-linear control method. Secondly, the paper will provide a detailed description of the TDAPPS method including the theory and mathematics that it is based on. The TDAPPS tool will also be compared to the non-linear stability analysis method that is most commonly used, known as the Describing Function method (5,6). The next step will be to show how the tool can be used to analyze an Apollo-like space capsule with 3-axis attitude control. The first example analyzed investigates a simplified 3-axis model in which the thrusters are fired only in the direction intended to correct the error in each axis. The second example removes this assumption and applies reaction control torque cross-coupling. The analysis of these cases provides an understanding of how the TDAPPS method can be used to analyze the stability of the controller design and determine phase and gain margins for a system. It also illustrates the flexibility of the TDAPPS tool, in that it can provide valuable feedback throughout a variety of scenarios, with nominal effort on the part of the analyst. Finally, the results of the TDAPPS analysis are verified using the original simulation to demonstrate the validity of the algorithm.
2. The Phase-plane Controller

2.1 Background

“Bang-bang” control systems are commonly used across a wide variety of applications from common household technology, such as thermostats, to satellites and other aerospace vehicles. In dual-mode thermostats, for example, the user simply sets a temperature minimum and/or maximum which act as the deadband values. If the temperature exceeds the upper deadband, the air conditioning will turn on to lower it. If the temperature were to drop below the lower deadband, the heat will activate until the room temperature is back within the given bounds. For spacecraft, a special type of “bang-bang” controller, known as a phase-plane controller can be used for precise attitude control during missions or phases that require quick and efficient attitude changes or holds with minimal fuel consumption. It works much like the thermostat example, except that there are two dependent variables, which are the vehicle attitude and attitude rate, as opposed to just temperature.

Phase-plane control has been used on board many spacecraft for attitude control [1,2,3]. Despite the vacuum of space, there can still be gravitational, mechanical, and electromagnetic disturbances that affect a vehicle’s attitude. Bang-bang control is very efficient for correcting these small disturbances after they have built up too much error. Notably, the International Space Station (ISS) and the Space Shuttle both employ(ed) phase-plane control for attitude maintenance during on-orbit attitude correction maneuvers and when the two vehicles were docked together [1,2,3]. Figure 2.1 is a diagram of the phase-plane space as defined for the Space Shuttle. Although the figure does not provide any values or explicit information, it is a great example of what the ‘phase-space’ looks like. Another attribute that makes the phase-plane controller so convenient is that this space will look very similar for different vehicles or even for different modes or configurations for
a system. Vehicles that dock to the ISS are a good example of this where, for example, the Space Shuttle docked to the ISS the coupling of the two vehicles creates an entirely different set of mass properties and responses to effectors [1]. For traditional control system design, this would require calculating a unique transfer function, but in this case the algorithm can simply switch to a different set of gains, which may be more computationally efficient. The phase-plane controllers used for these two vehicles are a great example of how flexible, efficient, and useful this type of control design can be.

![Diagram of Space Shuttle phase-plane space](image)

**Figure 2.1 – Space Shuttle phase-plane space. [3]**

### 2.2 Description

A phase-plane controller is a highly non-linear control method because it commands a binary reaction control system (RCS), meaning that the control effectors are either on or off [4]. In other words, there are no linear “ramps” connecting the two states. In some spacecraft, the control system actuators are small rocket engines, or jets, which cannot be throttled to provide variable forces. Each jet has only two states: on and off. When a vehicle’s RCS jets are off, the external torques acting about the center of mass are the primary contributors to changes to the vehicle’s attitude. If any of the RCS jets are on, then both external and RCS jet forces are acting
on the system, providing some control over the attitude. The job of the phase-plane controller is to decide when the state of each jet needs to be changed to achieve or maintain the desired attitude [4].

Determining the stability characteristics of such a non-linear system is not trivial. The industry-accepted method used to perform such analysis is known as the describing function method [5]. This method requires creating a linear estimator of the non-linear portion of the controller and determining the gain margins from the combined linear and “quasi-linear” portions of the plant [5,6]. However, upon studying the behavior of the phase-plane controller, a different and unique approach was hypothesized. The proposed method determines phase and gain margins in the time domain for each axis of a spacecraft and at every time step of a simulated trajectory. This is accomplished by predicting the behavior of the phase-plane controller in phase-plane space using the local inputs to the controller. This unique method is referred to as the time domain analysis of a phase-plane controller (TDAPPS). The theory behind the TDAPPS analysis tool will be discussed in detail starting with supporting information on the behavior of a simple, single axis phase-plane controller.

A phase-plane controller is a relatively simple concept that can have added complexity depending on the application. The simplest type of phase-plane controller requires accurate knowledge of the current actual attitude and rate of the vehicle, which it compares to the commanded attitude and rate to find two difference in each axis. All three axes of a vehicle can have a separate phase-plane controller. For each individual axis, the attitude and attitude rate differences are summed (a gain on rate error is included in some cases) to create a total error as shown in equation 2.1.

$$\overrightarrow{Err_{tot}} = \theta_{err} + \dot{\theta}_{err} \quad \text{Eqn. 2.1}$$
This total error is compared to pre-determined error limits that are parameters within the controller known as deadbands. If the total error does not exceed any of the deadband values, then the error is in a ‘drift’ region and the RCS jets are commanded to be off. When any of the deadbands are exceeded by the total error, an RCS jet is activated to correct the attitude and it will remain on until the total error is back within the drift region [4]. When drawn in the phase-plane space, these deadbands are represented by firing curves.

A convenient way to observe the behavior of the phase-plane controller is to plot the attitude error versus the attitude rate error in what can be called phase-plane space [4]. Figure 2.2 shows the trajectory of the total error in phase-plane space with two firing curves that represent the deadband parameters.

![Simple Phase Plane (400 sec)](image)

Figure 2.2. Phase-plane space plot for a simple phase-plane controller.
The hollow black circle marks the initial total error between the vehicle’s commanded and actual attitude and rate, while the solid black circle marks the target of zero error. The solid red dot is the location of the final data point. In this case, the initial point starts at the correct attitude but has a rate that leads to a gradually increasing attitude error. The solid red line is the trajectory of the total error and the black arrows show the direction that it is moving. The dashed lines are the firing curves, which are generated from the deadband values. In this case, the deadbands are set to a total error of ±1. Any time the total error crosses a firing curve, the RCS jet on-off state changes as does the trajectory of the total error. The phase-plane controller successfully drives the total error to within the tolerance even though the final point (red dot) does not match the target (black dot). This is acceptable because the attitude rate has been driven to zero and the attitude itself is within the deadbands, thus no further jet firings are required. It is also useful to show plots of the total error and RCS thrust versus time to illustrate the effectiveness of the controller. Figure 2.3 shows how the error is damped out within 40 seconds and how many jet firings it took to achieve the damping. This system does require some simplifying assumptions including no minimum on or off time for the RCS jets, which is unrealistic due to fuel tank pressure and command frequency. Another assumption is that the controller has perfect knowledge of the vehicles attitude and rates, as opposed to those derived from on-board sensors. Finally, the examples of firing curves shown throughout this thesis are all linear. It is more common to find firing curves are not infinitely linear, but that contain elbows (see Figure 2.1) or other non-linearities [2,3,4]. The assumption that they are linear is valid in this case because the total errors are always small enough that they exist within the linear portion of the phase-plane controller.
Figure 2.3. Top – Total Error vs. time; Bottom – Thrust versus time

It is notable in Figure 2.3 that between 38 and 40 seconds the negative jet is turned on and off many times, which is not an efficient use of RCS fuel. This is typically improved by introducing hysteresis to the system. Adding hysteresis complicates the logic required to implement the phase-plane controller, but it can significantly increase the performance. Hysteresis is introduced as a parameter to the system that shifts the firing curve in which the jet state would switch from on to off [4]. Figure 2.4 shows a plot of the phase-plane space under the same conditions as Figure 2.2, but with hysteresis firing curves included.
The hysteresis lines are the dashed red lines. They can be described as firing curves that have been shifted slightly closer to the target. Their primary function is to delay switching the jet state from on to off until after the total error crosses the hysteresis line. This prevents the rapid firing of an RCS jet, also known as a high duty cycle, seen in the Figure 2.3 between 38 and 45 seconds. However, the addition of the hysteresis parameter increases the time it takes for the total error to reach zero.

This type of controller behavior is ideal for a spacecraft in orbit that will be holding a specific attitude for more than a few seconds because it burns very little fuel and holds the attitude close to the commanded input [4]. It also helps that in orbit, the external forces on a spacecraft over a period of minutes are often negligible. Figure 2.5 supports this claim by showing the total error and thrust plots corresponding to the phase-plane trajectory in Figure 2.4. The advantage of
adding hysteresis is clear when comparing Figure 2.5 to Figure 2.3. Figure 2.5 shows that only five jet firings occur over the course of 400 seconds, whereas over 50 firings occur in 40 seconds in the previous example.

Both examples displayed in the previous figures are based on very simple single axis systems with no external forces. In the case of a spacecraft in orbit there will be tiny external forces from gravity gradients, solar pressure, payload operation, and cross-coupling forces from jet firings in other axes that affect each individual axis even if the total error is in the drift region. Even these small external forces will cause perturbations in the trajectory of the error [2]. Therefore, it is pertinent to present an example with an applied external force so that the error trajectory can be examined under conditions closer to what an actual orbiting vehicle would experience. Figure 2.6 shows the trajectory of the total error in phase-plane space with an arbitrary, constant external

Figure 2.5. Top – Total Error vs. time; Bottom – Thrust versus time
torque of 50 lb-ft applied to the system. The external torque causes an angular acceleration on the system that clearly affects the trajectory of the attitude and attitude rate. The error through the drift region is no longer a horizontal line as seen in the previous figures, but is now parabolic. Even the sections that are straight lines moving from one firing curve to the next can be described by a parabola, which means that the motion can be predicted. This is the basis of the proposed TDAPPS method.

![Diagram](image)

**Figure 2.6.** Single axis phase-plane controller with an applied external torque.

Defining the stability of a phase-plane controller from the phase-plane space is straightforward. Essentially, stability is evident if the attitude is closer to the desired target at the end of some analysis cycle in the phase-plane space than it was at the end of the previous cycle. An analysis cycle is defined a fixed number of times the RCS jet state has changed. For this study, an analysis cycle is defined as including six changes in the jet state, or crossing any firing curve
six times. To tell if the controller is stable over the span of one analysis cycle, the attitude rate error at the initial jet state change can be compared to the attitude rate error at the final jet state change. Figure 2.7 has numbers labeling each point where the state changes, which can also be described as points of intersection with firing curves, and how a single analysis cycle is defined. It is clear from this figure that the controller is stable at current conditions because the final absolute attitude rate error at point (6) is smaller than the rate error at point (1). Note: The attitude error is examined rather than the rate error, which cannot be used to determine stability in the same way. As the rate error starts to decrease, the attitude error values at firing-curve intersection points always approaches the deadband values. In other words, at low rates the attitude error values can be greater than they were at the initial high rate condition, but still fall within the acceptable limits even for a stable system.

Figure 2.7. Phase-plane space with numbered intersection points to show a full analysis cycle.
The same idea can be applied to any of the phase-plane examples shown so far to show that the results are stable. In fact, each of the phase-plane plots in the above figures can be shown to be consistently stable by utilizing this method. Figure 2.8 shows an example of an unstable phase-plane controller for reference. This instability was induced by subjecting the system shown in Figure 2.4 to a one-second pure delay between a commanded RCS jet state change and the actual jet’s response. The initial condition in Figure 2.8 is the hollow black circle, and the total error trajectory is steadily moving away from the target (at point [0,0]) instead of towards it. The solid black circle is the point at which a full analysis cycle is complete and, in this case, the rate error is greater than where it is at the starting point. In other words, the magnitude of the attitude rate error is increasing.

Figure 2.8. Unstable phase-plane controller.
Observations gleaned from studying the behavior of a single-axis phase-plane controller led to the proposal of finding instabilities in the time domain. The two observations that make the TDAPPS method viable are that the trajectory of the total error in the phase-plane space is parabolic for any steady non-zero external torque about the CG, and that the stability can be determined based on an initial and final value of the attitude rate error. TDAPPS is a predictive analysis tool that takes controller information at each time step and projects whether the response in the system will be stable or not. After determining stability, it can also apply gain and time delay margins to the system to determine the stability margins at each point. The information required to perform this analysis includes the deadband and hysteresis parameters, the target attitude, the current jet firing state, the current values of attitude and attitude rate error, and reasonably accurate knowledge of the torques in each region of the phase-plane space.
3. **TDAPPS Method**

3.1 **Description**

The TDAPPS method uses relatively simple algebra to predict the behavior of the controller. The equations for a straight line and a parabola are the main mathematical components of the algorithm. The difficulty lies in creating logic that determines which region the current error is, the amplitude of any external moments acting on the vehicle, and which deadband the total phase-plane trajectory will cross next.

The first step in developing the TDAPPS method is building a separate phase-plane space for each of the controlled variables, which, for this research project, are the spacecraft attitude Euler angles $\phi$, $\theta$, and $\psi$. The components of each phase-plane space are the firing curves, hysteresis curves, and the resulting three firing regions created in each Cartesian space. The three regions are referred to as the drift region, where no RCS jets are firing, and the positive and negative firing regions. Constructing this space requires knowledge of the deadbands ($DB_{nl}, DB_{lo}$) and hysteresis ($Hyst$) values for the controller being analyzed. For this study, it is assumed that the positive and negative deadband and hysteresis values for each axis will remain constant throughout each test case to make each of the firing and hysteresis curves linear. Because of this they can be expressed using the slope intercept formula (Eqn. 3.1):

$$y = mx + b \quad \text{Eqn. 3.1}$$

In the phase-plane space the attitude error ($\theta_{err}$) lies along the x-axis, while the attitude rate error ($\dot{\theta}_{err}$) is along the y-axis. Attitude error is defined as the difference between the vehicles current actual attitude and the commanded attitude. The attitude rate error is defined the same way. Applying this change to Eqn. 3.1 yields:
\[ \dot{\theta}_{err} = m\theta_{err} + b \quad \text{Eqn. 3.2} \]

Knowing that the deadband and hysteresis parameters are constant with respect to each axis of rotation (body roll, pitch, and yaw), and that the goal is to drive total attitude error to zero, then enough information is available to determine the slope, \( m \), and intercept, \( b \). Using the upper deadband as an example, the slope and intercept are expressed as follows:

\[
m = -|DB_{hi}|, \quad b = \theta_{des} + |DB_{hi}|\]

where \( \theta_{des} \) is the target attitude angle for that axis. Combining the above equations with Eqn. 3.2 results in the equation for the negative firing curve:

\[
\dot{\theta}_{err} = -|DB_{hi}| \ast \dot{\theta}_{err} + (|DB_{hi}| - \theta_{des}) \quad \text{Eqn. 3.3}
\]

Eqn. 3.3 yields the negative firing curve even though the upper deadband was used to formulate it because if the total error exceeds the upper deadband then the negative RCS jet needs to be activated to correct the error. The equations for the remaining firing curve and the two hysteresis curves can be generated using the same process, but with opposite signs. The equations for each of these curves are central to the TDAPPS algorithm as they will be used to calculate the projected points of intersection. Table 3.1 displays these four firing-curve equations in terms of the deadband, hysteresis, and commanded attitude parameters.

**Table 3.1. Equations for each of the firing curves in terms of controller parameters.**

| Negative Firing Curve | \[ \dot{\theta}_{err} = -|DB_{hi}| \ast \dot{\theta}_{err} + (|DB_{hi}| - \theta_{des}) \] | Eqn. 3.3 |
|-----------------------|-----------------------------------------------------------------|---------|
| Negative Hysteresis Curve | \[ \dot{\theta}_{err} = -|DB_{hi}| \ast \dot{\theta}_{err} + (|DB_{hi}| - \theta_{des} - Hyst) \] | Eqn. 3.4 |
| Positive Firing Curve | \[ \dot{\theta}_{err} = -|DB_{lo}| \ast \dot{\theta}_{err} - (\theta_{des} + |DB_{lo}|) \] | Eqn. 3.5 |
| Positive Hysteresis Curve | \[ \dot{\theta}_{err} = -|DB_{lo}| \ast \dot{\theta}_{err} + (Hyst - \theta_{des} - |DB_{lo}|) \] | Eqn. 3.6 |
Figure 3.1 is a simple representation of the phase-plane space created using the above equations. The hysteresis curves are represented by the dashed red lines and the firing curves by the dashed blue lines. The negative firing region is the area in the upper right hand side of the figure and is where the RCS jet firing to provide negative torque about the current axis is active. The positive firing region is in the lower left area of the figure and is where the opposite jet will be fired. The drift region, in the center of the figure, is the location at which no RCS jets are active. Each of the firing curves are labeled along with the three regions that they divide the space into. In this example, the controller parameters are set as follows:

\[ DB_{hi} = 1, \quad DB_{lo} = -1, \quad Hyst = 0.05 \]

Figure 3.1. Example of phase-plane space with key terminology labels.

Upon describing the phase-plane space for a vehicle, the basis of the TDAPPS algorithm has been formalized. The next step is the development of the TDAPPS logic. It should be noted
that the deadband and hysteresis values are not necessarily constant for any given vehicle, but for proof of concept, they will be assumed constant throughout this study. Adding non-linearities to the firing curves would complicate the logic required to run the TDAPPS algorithm, but certainly does not make it impossible.

The logic involved in analyzing a controller’s stability will be specific to the complexity of the phase-plane space as described above. However, the TDAPPS process that represents the basics of the algorithm will not change. Figure 3.2 is a flow chart that defines the main structure of this stability analysis method. Each step listed in the chart will be described in detail as it pertains to the spacecraft being analyzed in this study, but again, the overall method would change only slightly from one application to the next.
The inputs to the algorithm at each time step include the attitude error vector ($\hat{\theta}_{err}$), attitude rate error vector ($\hat{\dot{\theta}}_{err}$), jet state flag vector, angular acceleration in the body frame ($\ddot{a}_b$), the
deadband parameter \((DB_{hi}, DB_{lo})\), hysteresis parameter \((Hyst)\), and the commanded attitude vector \((\hat{\theta}_{cmd})\). The attitude error and attitude rate error vectors are defined by the difference between the vehicle’s current actual state and its commanded state. Equations 3.7 and 3.8 describe how they are calculated from simulation or measurement data to be provided to TDAPPSS.

\[
\hat{\theta}_{err} = \hat{\theta}_{act} - \hat{\theta}_{cmd} \quad \text{Eqn. 3.7}
\]

\[
\dot{\theta}_{err} = \dot{\theta}_{act} - \dot{\theta}_{cmd} \quad \text{Eqn. 3.8}
\]

The deadband and hysteresis parameters are known quantities since they are parameters defined in the design of the phase-plane controller itself. The commanded attitude vector is also a function of the controller and is likely based on mission requirements. These quantities need to be passed into the TDAPPSS function so that the phase-plane space can be constructed and projections can be made. The final input is the jet state flag vector, which is the value of the binary command passed from the guidance to the RCS models that commands whether a specific jet is active or inactive.

Upon receiving the appropriate inputs, the stability analysis algorithm begins with step one from the flow chart (Figure 3.2), ‘Determine Current Region.’ This involves using logic to decipher the value of the jet state flag. For the spacecraft that is studied in this paper, each rotational degree of freedom has a positive and a negative RCS jet. Table 3.2 shows how values are assigned to the jet state flag and what region that corresponds to in phase-plane space. If no jet is active, the flag has a value of 0 and the total error is in the drift region. If the positive RCS jet is commanded to be active, the jet has a value of 1 and the total error is in the positive firing region. Finally, if the negative jet is commanded to be active the value is \(-1\), and the total error is in the negative firing region.
Table 3.2 Jet state flag value assignment based on the command from vehicle guidance

<table>
<thead>
<tr>
<th>COMMANDED ON BY GNC</th>
<th>POSITIVE JET</th>
<th>NO JET</th>
<th>NEGATIVE JET</th>
</tr>
</thead>
<tbody>
<tr>
<td>JET STATE FLAG VALUE</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>PHASE SPACE REGION</td>
<td>+ Firing Region</td>
<td>Drift Region</td>
<td>- Firing Region</td>
</tr>
</tbody>
</table>

After determining the region in which the current total error resides, the algorithm can move on to predicting the path that the error will take within that region. This corresponds to step two in the flow chart, ‘Determine Shape Trajectory.’ This step is slightly more complicated as there are multiple branches of logic to follow, but it is the same procedure for any given application. There are two different possible paths that the total error will trace through the phase-plane space: linear and parabolic. This is based on whether the vehicle is being subjected to any external torques, thus providing an angular acceleration on the body during the (no RCS jets firing) drift. If there are no external torques acting on the system (i.e. from aerodynamics, gravity gradient, etc.) then the trajectory will be linear. This is only the case for an orbital spacecraft whose attitude currently rests within the drift region of the phase-plane space. If the total error is in either of the other regions or if the vehicle is subject to other external torques, then the trajectory in the phase-plane space will be parabolic.

Steps three and four require the most complicated logic of the process. Now that the algorithm has determined the shape of the trajectory, an equation for that curve can be realized. If the trajectory is linear, the sign on $\dot{\theta}_{err}$ can be used to determine which was the last firing curve crossed and which is the next to be crossed. Referring to Figure 3.3, the current Cartesian position of the total error is at the black circle showing that the rate error is greater than zero. Due to the nature of the phase-plane plot, this means that the total error trajectory is going to cross the negative firing curve at point (2) and previously crossed the positive hysteresis curve at point (1). Selecting
the corresponding firing curves for a parabolic trajectory is slightly more difficult. Fortunately, selection of the previous firing curve only needs to be completed on the first iteration of each cycle, because data can be saved from one projection to the next.

![Simple Phase Plane w/ Hyst (t = 400 s)](image)

**Figure 3.3 – Visual aid for describing curve projection.**

If the total error is in one of the firing regions, the selection of previous and next firing curve encounter is obvious. A total error in the negative firing region (i.e. between points (2) and (3) in Figure 3.3) means that the last firing curve crossed was the negative firing curve and the next one to be crossed will be the negative hysteresis curve. The difficulty in this step crops up if there is an external torque on the vehicle while in the drift region. Although this case will not be specifically examined in this paper, a method for determining these conditions was developed and can be reviewed in Appendix B in the `traj_project.m` Matlab code.
Step five involves solving for the next point of intersection with a firing curve based on the information gathered in the previous step. This is trivial in the case of a linear trajectory since the rate error will be constant. The only thing that needs to be solved for is the attitude error, $\dot{\theta}_{err}$, which can be done by solving one of Eqns. 3.3 – 3.5 depending on which curve is currently of interest. Once that next point of intersection is calculated, it is saved as the starting point for the next iteration of the TDAPPS cycle. This step is also more difficult when a parabolic trajectory is involved.

The only extra piece of information required to calculate the intersection of the error trajectory with a firing curve is the angular acceleration that the RCS jet is supplying. This study assumes that there are no external torques acting on the vehicle, which greatly simplifies the calculation and projection of angular acceleration values during a TDAPPS analysis cycle. If this assumption was not made, it would be very difficult to obtain the actual angular acceleration at a given time step, which would also complicate projecting the vehicle’s trajectory through the analysis cycle. However, thanks to this assumption, the angular acceleration ($\ddot{\alpha}$) values associated with each section of the phase-plane space can be initialized in the algorithm using the RCS torque ($\vec{\tau}_{RCS}$) and vehicle inertia data ($[I]$) as seen in Equation 3.9.

$$\ddot{\alpha} = \frac{\vec{\tau}_{RCS}}{[I]} \quad \text{Eqn. 3.9}$$

The next step is to develop the equation for the parabolic trajectory by observing a relationship between the vehicle’s angular acceleration and the phase-plane space. Equation 3.10 shows this relationship and comes from taking the partial derivatives of the rate of change through the phase-plane space.
By rearranging this equation and taking the partial derivative with respect to $\partial \theta_{err}$, the attitude error can be projected through the phase-plane space using the following:

$$\frac{\partial \theta_{err}}{\partial \theta_{err}} = \frac{\theta_{err}}{\alpha}$$  \hspace{1cm} Eqn. 3.10

where $B$ is a constant equal to the value of $\theta$ when $\omega$ is zero. This equation is valid for any parabolic trajectory of the total error through the phase-plane space.

With these Eqns. 3.3 – 3.5, 3.10, and 3.11, a full cycle through the phase-plane space can be projected. TDAPPS uses a logic tree to determine which of the four firing curves will be crossed next and then, since the parabolic equation and the appropriate firing curve equation are in terms of $\theta$, they can be solved simultaneously for $\theta$ using the quadratic formula. This provides two values of $\theta$ at intersection points with the given firing curve and these examined in another logic tree to determine the correct one. This process is repeated, projecting backwards in time, to find the value of $\theta$ at the previous point of intersection with whichever firing curve has been most recently crossed. With these two values of omega, the parabolic equation can be used to calculate the $\theta$ values at the intersection points and the trajectory throughout the current region has been projected. This process is repeated a total of six times for each set of attitude and attitude rate errors to project a cycle all the way around the phase-plane space. Finally, the initial attitude rate error and the final attitude rate error values from the projection are compared and if magnitude of the final value is less than the initial, the controller is declared stable for this point in time. A tolerance is applied to account for minor increases in rate error that would be common when the total error is very small. Equation 3.12 is the mathematical expression for determining stability:
\[
\dot{\theta}_{err,f} - \dot{\theta}_{err_i} \leq 1e^{-3} \quad \text{Eqn. 3.12}
\]

If the conditions of Equation 3.12 are satisfied, the stability flag output is assigned a value of 1 and considered stable. Otherwise, the flag is given a value of 0 and the system is considered unstable. Also, if the rate error itself is less than 0.1°/s, the stability flag is automatically set to one (stable). This helps to eliminate the condition in which the error is just barely greater than zero such that when a jet is fired, it increases more than the tolerance.

After this algorithm is executed and the initial stability is determined, the same set of points is passed through two more algorithms that do almost the same thing. Instead, the first algorithm adds in an artificial time delay margin which simulates what would happen if the command was not followed until sometime after it was given. The delay margin is initialized at 500 ms and the stability of this new system is determined in the exact same way as before. Then the delay value is updated using the Newton-Raphson optimization method to locate the point at which the controller just barely becomes unstable. This value of additional time delay is the smallest amount that can be added to the system to cause it to go unstable. This same method is applied to find the positive and negative gain margins. If the open-loop system is initially stable, a negative gain margin does not exist.

The outputs of the TDAPPS function are time histories of the stability flag, time delay margin, and positive and negative gain margins. Each iteration of TDAPPS works from a ‘snapshot’ of the plant and control system, projecting ahead based on current parameters and dynamics. This provides valuable insight at each point of the vehicles trajectory as to whether the control system was stable or not and how much gain or phase delay would need to be induced to make it unstable. With problem areas such as these identified, investigations could then be
launched to determine the causes of such a result and assess the effectiveness of the controller in unique situations.

3.2 **Comparison with Describing Function Method**

A popular method used for analyzing the stability of non-linear controllers, including phase-plane controllers, is known as the describing function method [5]. There are several approaches to this problem including the absolute stability approach, which employs a combination of Popov’s criterion and Siljak’s loop transformation, but requires fairly limiting assumptions and has mainly been used to analyze the roll channel of vehicle launch systems [6,7]. However, based on a recent literature review, the describing function method seems to be the most widely used and accepted. Thus, it seems pertinent to briefly describe how a describing function analysis is accomplished and compare the process and results to the TDAPPS method.

The describing function method is often billed as a frequency domain analysis for non-linear control [5]. Therefore, the first step in this method is to fit the system dynamics into a classic control loop as seen in Figure 3.4. This traditional loop consists of the controller, in this case non-linear and the plant representing the dynamic response, which closes the loop on the system. The reason for fitting the desired system into this loop is to use well-established methods to perform frequency domain stability analysis on the system. The catch is, these methods require a linear controller model, thus, a linear representation of the non-linear controller is required. Creating a linear approximation of the phase-plane controller is the next step in the describing function method [5].
Coming up with a reasonable linear approximation of a non-linear system is one of the biggest challenges of this method because the analyst needs to create a linear transfer function in the frequency domain that is specific to each non-linearity and is dependent on the amplitude of the error signal. When there are more than one non-linear elements, this process needs to be repeated for each. However, once the transfer function has been identified, several traditional and well-known analysis techniques are available for use in determining margins [5]. Examples of these include Bode and Nyquist plots which display the frequency response of the system and allow for graphical determination of gains. The ability to evaluate the controller using classical and modern controls techniques is a distinct advantage of the describing function method.

The TDAPPS and describing function methods use an entirely different process to estimate stability margins and both have distinct advantages over the other. The TDAPPS method uses straightforward math and logic to predict controller performance and then provides the margins along with graphical feedback in the time domain. The describing function method requires linearized approximations for any non-linear component of the controller, but then allows for traditional analysis in the frequency domain [5]. This is a distinct advantage over the TDAPPS method. The use of these established methods provides a lot more confidence in results and will be the expectation of any control experts. However, the flexibility, ease of use, and unique graphical methods that TDAPPS provides makes it an attractive option. It may also provide an alternate means of checking gains calculated in another manner and describing these margins to anyone who is not a controls analyst.
TDAPPS is also comparable to using brute force to determine margins. The brute force method is applying gains to the controller in a simulation environment and changing them until instabilities occur. This can be time consuming and could make it easy to miss conditions in which the gains may be different. TDAPPS is a sort of faster, formalized brute force method. Using its definition of stability, it calculates the margins for each axis and at each time step of a simulation. This ensures that the minimum tolerable margins are defined for each simulation. The results can then be confirmed by applying the specific gain to the simulation instead of estimating values and running it multiple times.

Upon conclusion of the literature review and weighing the advantages and disadvantages of TDAPPS and the existing techniques, it was determined that there was value in continuing to develop TDAPPS as an alternate form of phase-plane controller analysis.

### 3.3 Simulink® Simulation Description

A MATLAB® Simulink® simulation was created to aid in the formulation and testing of the Time Domain Analysis of Phase-plane Systems algorithm. Simulink® is a block diagram based simulation tool that is often used to develop and analyze a wide range of systems and applications. TDAPPS was developed using MATLAB® version R2015b, but has since been updated to R2017b. The Simulink® model, *three_axis_phase_plane_exp.slx*, and its supporting suite of MATLAB® scripts will be described in this section. Most of these files will also be included in Appendix 6.3.

Figure 3.5 shows the top level of the simulation of a spacecraft attitude control example containing a three-axis degree-of-rotational-freedom rigid body and an associated three-axis phase
plane controller. This has the traditional feedback loop in which the states are sent into the controller to determine the changes needed to track the desired rotational states. The subsystem labeled Phase-plane Cont. contains both the discrete phase-plane controller and a single reaction control system. The inputs to the controller are the current attitude and attitude rate errors. The outputs are the total commanded RCS torque and the variable called Jet_Flag, which denotes what region of the phase-plane space the error in each axis currently resides. The commanded torque is combined with any external torques (if desired) to create a total torque acting on the vehicle, which is then sent into the Plant model. This continuous subsystem calculates the current angular acceleration of the vehicle based on the total torques and the current body rates using the following equation:

\[
\begin{bmatrix}
\alpha_x \\
\alpha_y \\
\alpha_z
\end{bmatrix} = \begin{bmatrix} l_{xx} & l_{xy} & l_{xz} \\
l_{yx} & l_{yy} & l_{yz} \\
l_{zx} & l_{zy} & l_{zz}
\end{bmatrix}^{-1} \begin{bmatrix}
t_x \\
t_y \\
t_z
\end{bmatrix} - \begin{bmatrix}
\omega_x \\
\omega_y \\
\omega_z
\end{bmatrix} \times \begin{bmatrix} l_{xx} & l_{xy} & l_{xz} \\
l_{yx} & l_{yy} & l_{yz} \\
l_{zx} & l_{zy} & l_{zz}
\end{bmatrix} \begin{bmatrix}
\omega_x \\
\omega_y \\
\omega_z
\end{bmatrix}
\]

Eqn. 3.13

In Equation 3.13, \( \ddot{\alpha} \) is the angular acceleration, \( \tau \) is the torque on the system, \( \omega \) is the angular rate, and \([I]\) is the vehicle's inertia matrix. The angular acceleration is then integrated to determine the attitude rate and attitude. These states are fed back into the controller for the next time step. Figure 3.6 displays the Plant subsystem.

**Figure 3.5** – Top level of *three_axis_phase_plane_exp.slx*.
Figure 3.6 – Plant model for the three-axis phase-plane controller.

The more complicated part of this simulation model was the development of the phase-plane controller itself. It was developed using Kubiak as a reference [4]. Figure 3.7 shows the first level of the Phase-plane Cont. subsystem.

Figure 3.7 – Top level of the phase-plane controller subsystem.
This level splits the input attitude error and attitude rate error into their respective axes and feeds them into a separate phase-plane controller for each axis. The resulting commanded torques are output as a vector into the plant model. The individual jet flags are combined into a vector and saved as an output because they are essential for the TDAPPS algorithm. Digging down further into the phase-plane controller itself, Figures 3.8 and 3.9 help describe the finer details involved. Figure 3.8 is the level in which the controller inputs are split between the upper and lower deadband logic models. In this simulation, the phase-plane space is symmetric about the zero-error location, or the center of the Cartesian plot. This means that the deadband and hysteresis magnitudes are the same for each firing curve, but the signs are opposite. The other inputs to each logic model are the current total error in the desired axis and the values of RCS torque if the jet is on and off. The outputs of both models are combined to form the commanded RCS torque and jet flag value for the specified axis.

![Diagram](image-url)

Figure 3.8 – Splitting phase-plane controller between upper and lower bounds.

The logic for both the upper and lower phase-plane boundaries are very similar, so just the upper deadband logic is displayed in Figure 3.9. At this level, the total error is compared to the deadband and hysteresis values. The logic dictates that if the error exceeds the deadband, then the
jet is active and the jet flag value becomes negative one (or negative one in the lower bound logic) to activate the negative RCS jet. This jet is referred to as the negative RCS jet because it reduces the error from a positive value back down to zero. The hysteresis adds some complexity to this logic because it only comes into play when the jet flag value is going from on to off. This is handled by also checking the sign on the error value, which denotes what quadrant the error is in. If the error is in the lower right quadrant, that means the hysteresis curve comes into play and it delays the termination of the jet firing.

![Phase-plane controller upper level deadband logic model](image.png)

**Figure 3.9 – Phase-plane controller upper level deadband logic model.**

This simulation is initiated and run using a MATLAB script called `run_3x_phaseplane.m`. This script creates a structure called `SimIn`, which is how all the initial conditions and parameters are passed into the Simulink workspace. From this interface, the user can change these values of these parameters for analysis purposes. After all the values of this structure are populated, the simulation is executed for the user-specified amount of time. Finally, the outputs are saved to the MATLAB workspace so that they can then be passed into the TDAPPS algorithm. The simulation is run as a fixed-step model at 10 frames per second. This is the test bed that was utilized to conduct all the testing and analysis for this proof-of-concept study.
4. **TDAPPS Application**

4.1 **Apply to generic re-entry capsule**

The time domain analysis of a phase-plane controller stability algorithm was originally developed to aide in the evaluation of a capsule-like spacecraft’s control system. Space capsules often require precise attitude adjustments or maintenance so that they can dock with other spacecraft, point sensors at a target, or enter the atmosphere of some celestial body [2,3] at the proper angle. For example, it is imperative that a capsule docking with the International Space Station damp out any rates and hold a precise attitude so that the docking mechanisms will operate correctly [1,2]. Since it is common to find a phase-plane controller in a capsule’s guidance, navigation, and control system (GNC), the TDAPPS algorithm was tested using specifications for an Apollo-like space capsule.

The simulation used to test TDAPPS is only concerned with the attitude of the capsule, so the only information required are the mass properties and reaction control system design. Several documents are online that contain the necessary information [8,9]. The nominal set of vehicle mass properties in this simulation are defined using the Apollo 14 mission report when the vehicle is at entry interface, or the point at which the capsule is about to enter the Earth’s atmosphere [10]. This means that the Service Module (SM) is not included in this simulation. Table 4.1 shows the nominal Apollo 14 mass properties used in the simulation.
The only required information for the RCS model was the total magnitude of thrust that the jets were capable of supplying. According to the Apollo Experience Report [10] the command module RCS jets produced an average of 93 lbf of thrust. Unfortunately, data describing the orientation and location of each jet on the CM could not be found. Instead, reasonable values were assumed and the simulation was tested across a range of these values to try to incorporate the different possibilities. Figure 4.1 is a drawing of the Apollo command and service modules. The command module is on the left side of the image and is the resource used to estimate the locations of the positive and negative RCS jets. The image also shows the orientation of the command module body frame, which is important for defining the Euler angles and rotation rates of the CM. This also helps with assigning the thrust alignment unit vectors to each RCS jet. Originally, the jets were treated as if they were perfectly aligned with its respective axis; however, this simplification was made only to test the simulation and TDAPPS method. Upon confirming that the results were within the expected ranges, more realistic estimated unit vectors were applied such
that there would be cross-coupling of moments in other axes. Both the simple and cross-coupling cases will be discussed in this section.

**Figure 4.1** – Depiction of CM body reference frame and general RCS jet locations [9].

### 4.1.1 3-axis attitude control

Preliminary investigation of the usefulness of the TDAPPS algorithm was conducted with the assumption that the RCS jets were perfectly aligned parallel to one axis. In other words, no cross-coupling of RCS torques were generated. The initial attitude state error conditions of the capsule can be seen in Table 4.2 and were defined in such a way that the controller would be forced to damp out initial rates in each axis.

**Table 4.2** – Initial Euler angle attitude and angular rates.

<table>
<thead>
<tr>
<th>$\phi_{\text{err}}$ (°)</th>
<th>$\theta_{\text{err}}$ (°)</th>
<th>$\psi_{\text{err}}$ (°)</th>
<th>$\dot{\phi}_{\text{err}}$ (°/s)</th>
<th>$\dot{\theta}_{\text{err}}$ (°/s)</th>
<th>$\dot{\psi}_{\text{err}}$ (°/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
<td>0.2</td>
<td>0.5</td>
</tr>
</tbody>
</table>
These values were subsequently varied to provide a sweep through a range of slow rates to a relatively high rate of rotation for a spacecraft. Also, due to differences in thruster location and inertia values about each axis, the RCS torques and angular accelerations acting on the system are different depending on the axis. Table 4.3 shows the torques and accelerations that each axis experiences when its respective RCS jet is fired. There are a few assumptions regarding the RCS model that were made to simplify the simulation. The first is that there are no minimum or maximum on-off times for any of the RCS jets. This means that no matter when a command is issued the system can respond, which is not always true of an actual physical system. Another assumption is that the thrust is modeled as a perfect step-input to the system, which excludes the typical dynamics and rise-time seen in actual engine performance.

**Table 4.3 – RCS torque and corresponding angular acceleration magnitude in each axis.**

<table>
<thead>
<tr>
<th>Axis</th>
<th>RCS Torque (lb-ft)</th>
<th>Ang. Accel. (°/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roll (x)</td>
<td>±325.5</td>
<td>±0.0552</td>
</tr>
<tr>
<td>Pitch (y)</td>
<td>±558</td>
<td>±0.1057</td>
</tr>
<tr>
<td>Yaw (z)</td>
<td>±558</td>
<td>±0.1172</td>
</tr>
</tbody>
</table>

The simulation was run for 200 seconds with these initial conditions and the subsequent simulation data was analyzed with the TDAPPS algorithm. The results are broken down by axis and discussed in this section.

The vehicle began with a small 0.05°/s rotation rate about the roll-axis and Figures 4.2 and 4.3 show the time history of the capsules response to this condition. The black, hollow circle represents the starting point of the total error and it is initially moving in the positive x-direction. Since the vehicle is drifting slowly along the x-axis, the control system only needs to fire three times to keep the error within the tolerable region. It should be noted that each time the jet fires, the magnitude of the drift rate decreases. In an actual spacecraft application, there may be a
secondary control law aiming to damp out this type of drift to avoid a limit cycle, but this feature was excluded from the simulation. The reason for this is to confirm that the TDAPPS algorithm provides valuable feedback when a short, burst jet firing occurs providing a rapid transition in the region that the total error occupies.

Figure 4.2 – Roll axis phase-plane plot with no RCS cross-coupling.
This data was fed into the TDAPPS algorithm and the resulting stability parameter time histories can be seen in the plots of Figure 4.4. The first plot in Figure 4.4 is of the roll-axis stability flag, which shows that the controller was considered stable (stability flag value of one) throughout the entire simulation. As a reminder, the current definition of stability (defined in section 3.1) is that the magnitude of the vehicle’s angular rate error is less than or equal to the rate error at the previous jet-state change. Looking closely at the phase-plane plot of the total error in Figure 4.2, the correction thrust is stronger than necessary for the total error and the controller enters a low-frequency duty cycle. During the period that TDAPPS shows that the vehicle is stable, but that this type of control method may not be the most efficient for this regime of the attitude error. This analysis is backed up by the gain margins that TDAPPS calculated for this axis.
Figure 4.4 – TDAPPS results for the roll axis.

The second, third, and fourth plots in Figure 4.4 show the time delay and system gain margins (positive and negative), respectively. The range of allowable delay values is between 0 and 0.75 seconds. This means that the roll axis can tolerate a maximum delay of 0.75 seconds between the jet command and its response. However, that amount of delay is only tolerable briefly at the beginning of the simulation. Otherwise, after the initial corrective firing at 20 seconds, the system cannot tolerate much, if any, time delay in the roll axis. This make sense since the vehicle is in a duty cycle where the total error is close to the target, so any abnormal conditions could easily cause an increase in the rate error. This observation is further supported by the predicted gain margins.

The positive gain margin has a range of 0 to 12 dB, or up to 4 times the nominal thrust magnitude. A value of 0 dB means that no gain margin can be tolerated and this is the result for when the controller initially enters the duty cycle. Note that the zero-gain margin condition is
constant between the first and second jet command. This means that if an increase in effector torque occurred, it would be enough to increase the magnitude of the total error instead of decreasing it. This will be explored further when these results are tested in the simulation. Similarly, the negative gain margin has a range of -12 to 0 dB. The margin of -12 dB correlates to a 75% reduction of the nominal thrust. This value remains at 0 dB for most of the simulation, except for the beginning and the times where the delay margin increases. The consistency of these results, as well as the observations that can be made to support them, help to validate these initial TDAPPS results.

The pitch and yaw axes were examined in a similar way and the results can be seen in Figures 4.5 through 4.8. Note from Table 4.2, that the initial rate errors are different from the roll channel. This provides two different examples on the usefulness of the TDAPPS method. For the pitch axis, the error does not enter a limit cycle until after the second firing. The results for this axis are reasonably straightforward. The results show that this axis is also stable throughout and show consistent delay and gain margins. The gain margin set is again between -12 and 12 dB while outside of the limit cycle, and each margin goes to zero when the total error is too small. The amount of tolerable delay starts at 0.9 seconds, but it drops to 0 for the same periods that the gain margins do. After entering the limit cycle, the same analysis from the roll axis can be applied; if the total error is too small, it cannot afford any changes to the system. This does show that TDAPPS is sensitive to these limit cycle and that either the stability condition needs to be changed, or a change to the physical or control system needs to be made. In this case, the limit cycle is not significantly inefficient since the rate error is low and the jets only briefly fire twice once it begins. This may be a location where more analysis would be useful and a convenient part about the
TDAPPS algorithm is that experimentation in changing parameters can be performed in a simulation setting and then re-examined with relative simplicity.

Figure 4.5 – Pitch axis phase-plane plot with no RCS cross-coupling.
The results for the yaw axis are the most interesting of this initial test. With the largest initial rate error, the response in this axis does not enter a limit cycle before the simulation ends. Figure 4.7 shows the trajectory of the total yaw axis error in the phase-plane space. Figure 4.8 shows the results of the TDAPPS analysis for this axis and it again shows stable behavior throughout. This example is the easiest one to visually confirm (using Figure 4.7) since the total error is clearly getting closer to zero with each jet firing. Looking at the margins calculated for this axis, all three start out relatively large with a maximum delay of 1.1 seconds, positive margin of 12 dB, and a negative margin of -12 dB. However, these all drop to zero after the 40-seconds mark. This is again because the total error has been driven so close to zero that if a jet activates at all, it would drive it further from the target. Again, this shows that the strength of the RCS jets gets the error in check very quickly, but does not drive it to zero. Since the jets are firing so infrequently and in such short bursts, it is hard to claim that the controller is behaving inefficiently.
Figure 4.7 – Yaw axis phase-plane plot with no RCS cross-coupling.

Figure 4.8 – TDAPPS results for the yaw axis.
The analysis performed in this section was conducted on a simplified system. The assumption that the reaction control system will fire perfectly along a single axis is not quite realistic. It did, however, prove that the TDAPPS algorithm provides reasonable results that offer a unique look at the control system. Each set of results can be compared to the phase-plane plot for the corresponding axis and checked for accuracy. Along with showing when and if instabilities occur in the controller, it also provides a discrete set of gain and phase margins for each axis. The flexibility of the algorithm can also be utilized to vary stability conditions and observe the robustness of the system under different conditions. The next section will remove the assumption of a perfect jet firing and examine results for a system that contains RCS torque cross-coupling.

4.1.2 RCS Jet cross-coupling

Upon conclusion of the single-axis torque testing, it became clear that some minor changes could be made to the simulation to make the results more interesting and useful. The first of the changes made were to the initial conditions. Specifically, the magnitude of the initial rate errors was increased to view a wider envelope of RCS performance in each axis. Table 4.4 displays the initial attitude and rate errors used in this section.

Table 4.4 – Initial Euler angle attitude and angular rates for the cross-coupling simulation.

<table>
<thead>
<tr>
<th>$\phi_{err}$ (°)</th>
<th>$\theta_{err}$ (°)</th>
<th>$\psi_{err}$ (°)</th>
<th>$\dot{\phi}_{err}$ (°/s)</th>
<th>$\dot{\theta}_{err}$ (°/s)</th>
<th>$\dot{\psi}_{err}$ (°/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>-0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Another internal change was to add cross-coupling of torques. This provided a much more realistic model than the previous assumption of uncoupled responses because most vehicles do not generate perfect torque about a single axis during RCS actuation. A reference with the exact
position and direction of thrust for each RCS jet on an Apollo-like capsule could not be acquired. Instead, reasonable values were assumed using Figure 4.1 as a guide. Table 4.5 displays the thruster positions and corresponding unit vectors for each jet. The center of gravity and inertia of the vehicle remained unchanged from the values given in Table 4.1.

Table 4.5 – RCS jet thrust alignment unit vectors.

<table>
<thead>
<tr>
<th>RCS Jet</th>
<th>X (ft)</th>
<th>Y (ft)</th>
<th>Z (ft)</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos. Roll</td>
<td>86.5</td>
<td>0.0167</td>
<td>3.1</td>
<td>0.1625</td>
<td>-0.95</td>
<td>0.3</td>
</tr>
<tr>
<td>Neg. Roll</td>
<td>86.5</td>
<td>0.0167</td>
<td>3.1</td>
<td>-0.1625</td>
<td>0.95</td>
<td>-0.3</td>
</tr>
<tr>
<td>Pos. Pitch</td>
<td>80.5</td>
<td>0.0167</td>
<td>0.475</td>
<td>0.3</td>
<td>0.1625</td>
<td>0.95</td>
</tr>
<tr>
<td>Neg. Pitch</td>
<td>80.5</td>
<td>0.0167</td>
<td>0.475</td>
<td>-0.3</td>
<td>-0.1625</td>
<td>-0.95</td>
</tr>
<tr>
<td>Pos. Yaw</td>
<td>86.5</td>
<td>6</td>
<td>0.475</td>
<td>-0.95</td>
<td>0.3</td>
<td>0.1625</td>
</tr>
<tr>
<td>Neg. Yaw</td>
<td>86.5</td>
<td>6</td>
<td>0.475</td>
<td>0.95</td>
<td>-0.3</td>
<td>-0.1625</td>
</tr>
</tbody>
</table>

The torque supplied by each jet is calculated by multiplying the total thrust magnitude by the above unit vectors. This vector is then crossed with the difference between the thruster location and the vehicle CG. The resulting torque values are shown in Table 4.6.
Table 4.6 – RCS jet torque vectors.

<table>
<thead>
<tr>
<th>RCS Jet</th>
<th>( Q_x ) (lb-ft)</th>
<th>( Q_y ) (lb-ft)</th>
<th>( Q_z ) (lb-ft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos. Roll</td>
<td>232.8</td>
<td>39.7</td>
<td>-0.5</td>
</tr>
<tr>
<td>Neg. Roll</td>
<td>-232.8</td>
<td>-39.7</td>
<td>0.5</td>
</tr>
<tr>
<td>Pos. Pitch</td>
<td>2.9</td>
<td>530.1</td>
<td>-91.6</td>
</tr>
<tr>
<td>Neg. Pitch</td>
<td>-2.9</td>
<td>-530.1</td>
<td>91.6</td>
</tr>
<tr>
<td>Pos. Yaw</td>
<td>90.9</td>
<td>0</td>
<td>531.6</td>
</tr>
<tr>
<td>Neg. Yaw</td>
<td>-90.9</td>
<td>0</td>
<td>-531.6</td>
</tr>
</tbody>
</table>

The increased initial rate error in each axis provides a much different response as can be seen in Figures 4.9 and 4.10, which show the roll axis phase-plane, total error, and torque plots.

Figure 4.9 – Roll-axis phase-plane response of vehicle with cross-coupling.
In this simulation, the total error in the roll axis was high enough to require five jet firings, each greater than three seconds to reduce it. It is also easy to see from both plots how the trajectory is affected by the jets firing to correct error in other axes. By the end of the 200 second run, the rate error has been reduced to well below 0.1°/s, but this does not occur until about 125 seconds (time of last roll jet firing). Two minutes may be a long time to arrest the total error into a tolerable range; however, this axis is seeing large torques from the firing of jets in other axes. This is clear from the roll axis torque plot in Figure 4.10 where several variations in torque are noticeable; especially in the first 30 seconds. It should also be noted that the moment arm for the torques in this axis is only about three feet. This, combined with the fact that the vehicle’s roll inertia is the largest inertia component, means that the RCS torque will be lowest about the roll axis.

Figure 4.10 – Roll-axis total error and torque time histories.
Figure 4.11 shows the TDAPPS stability flag, gain and delay margin results for the roll axis. The stability flag shows a stable response throughout the entire 200 second simulation. The time delay has a maximum value of 3.5 seconds, but this is only valid for a brief period at the 4-second mark. It should be noted that the tolerable time delay margin is getting smaller as the total error is driven closer to zero. This follows the observations made in the previous section showing that the closer the total error is to zero, the smaller the tolerable margins are. The same thing occurs for both the positive and negative gain margins, as well. The positive gain margin, prior to 100 seconds, jumps back and forth between 6 and 12 dBs. This means that the system could tolerate a maximum of double the nominal thrust and remain stable during this period. The negative gain margins hold fairly constant at -12 dB. Therefore, the system can tolerate a range of 0.25 and 4 times the nominal amount of thrust from the RCS jets and a maximum of 3.5 second delay in command response.
The pitch axis response can be seen in Figures 4.12 and 4.13. The phase-plane space in this case looks much different from the previous ones. The interesting portion of the error trajectory occurs between 45 and 60 seconds. During this period, there is an almost 10-second long firing of the negative roll jet, which produces a small negative pitching torque. It happens that this occurs just as the total error is about to cross the positive (lower) firing curve in the pitch-axis phase-plane space. This is the cause of the double pulse firings (seen in Figure 4.13) of the positive pitch RCS jets as the error rate changed sign without crossing the opposite firing curve. This also matches up to a decrease in gain and phase margins to zero. The controller is still stable at this point, but the unique timing of the multiple firings is a sensitive area that has been appropriately highlighted by the TDAPPS results, shown in Figure 4.14. Examining a situation like this is a unique feature provided by this analysis method.

![Pitch Channel Phase Plane Space](image)

**Figure 4.12 – Pitch-axis phase-plane response of vehicle with cross-coupling.**
Figure 4.13 – Pitch-axis total error and torque time histories.

The last 75 seconds of the simulation show the pitch axis with a small, negative rate error that is below -0.1°/s. It is so small that any corrective firings would unavoidably increase the magnitude of the rate error, which is why the TDAPPS results show it cannot tolerate any gain changes during this time. While this is technically true, it is not necessarily of any concern because of how slow the drift is. Through the first 100 seconds, when jets in all axes are firing, the delay margin is consistently greater than 0.45 seconds except for the two lengths previously discussed. After 100 seconds, the time delay margin drops close to zero for the rest of the trajectory. The range of positive gain margin values during this time is between 5.9 and 12 dB, or between two and four times the maximum thrust available. The negative gain margin oscillates between the maximum allowable reduction of 0.25 times the thrust and zero tolerable reduction. It does this more frequently than the positive gain margin and the time delay suggesting that the vehicle is more sensitive to reduction in thrust than increases or delays between commands. This would
follow general logic as a loss in thrust obviously provides less control authority than an increase or delay.

![Pitch axis TDAPPS results](image1)

![Delay (ms)](image2)

![Pos. gain (dB)](image3)

![Neg. gain (dB)](image4)

**Figure 4.14 – TDAPPS analysis for pitch-axis cross-coupling simulation.**

The yaw axis has the most uniform results of this set of analyses. The cause of this is the miniscule component of torque cross-coupling with the roll jet firings and the infrequent pitch jet firings. Thus, the effects of cross-coupling torque are almost insignificant. There is an interesting and unique occurrence, however. Figures 4.15 and 4.16 show the yaw axis phase-plane plot, z-axis torques, and total error. The initial attitude correction portion is dominated by the pitch axis’ own control system with only the occasional ‘wiggle’ from another axis’ firing. The error is driven below a magnitude of 0.1°/s within the first 100 seconds of the simulation at which point the cross-coupling torques become more noticeable. During the limit cycle phase of the trajectory, Figure 4.15 allows the observation of several deviations and zero-crossings that would normally not be present. There are only three brief yaw jet pulses in the last 160 seconds, but there are a lot of rate
error changes throughout this period, which provides another unique situation to observe how the TDAPPS algorithm will behave.

Figure 4.15 – Yaw-axis phase-plane response of vehicle with cross-coupling.
Figure 4.16 – Yaw-axis total error and torque time histories.

Figure 4.17 shows the TDAPPS results for the yaw axis. Again, it is stable throughout the entire 200 second simulation and nothing from the phase-plane plots would suggest that this is incorrect. The time delay and positive gain margins only drop to zero for a brief time span between 41 and 50 seconds. This is the only time when the rate error reaches a low enough value that any change in system parameters affecting these two margins would de-stabilize the control system. The maximum allowable time delay is 1.1 seconds and this number generally decreases as the simulation progresses, but again, it does not reach zero indicating that the system can still tolerate off-nominal conditions. The positive gain margin ranges between 6 and 12 dB save for the 44 to 55 second period. The negative gain margin oscillates more frequently between zero and -12 dB, but even that stabilizes at -12 dB beyond 100 seconds. The reason that this axis has a more consistent period of stability is because the cross-coupled torques prevent the rate error from hovering too closely to zero. This is not necessarily a good thing because it means the RCS jets
will have to pulse more often, but it does once again prove that the TDAPPS results can be trusted in another atypical situation.

![Figure 4.17 – TDAPPS analysis for yaw-axis cross-coupling simulation.](image)

### 4.2 Discussion of TDAPPS results

The testing of the Time Domain Analysis of Phase-plane Stability algorithm conducted in the previous section provided examples of how it can be used for analyzing unique systems with a non-linear portion of control. Aside from providing a range of gain and phase margins for each axis, the algorithm highlighted where the controller was inefficiently entering a limit cycle, how the system can be sensitive to thruster torque cross-coupling, and how a reduction in total RCS thrust (increase in negative gain margin) is a more significant issue than a time delay as seen in Figure 4.17. This proof-of-concept experiment also revealed some shortcomings that need to be solved or supplemented. The following discussion will prove the accuracy of the TDAPPS-
predicted gain and phase margins, further explore how this tool can be utilized, and discuss the outstanding issues that the method contains.

Prior to confirming the phase and gain margin results, it should be noted that the bounds on the gain margins were set between -12 and +12 dB. The purpose for this was to speed up the optimization portion of the TDAPPS algorithm, by stopping the Newton-Raphson based method after it exceeded non-realistic values. For example, it would not be likely that any of the jets would provide more than four times the nominal thrust (+12 dB). In order to confirm the TDAPPS results, gain and delay blocks were applied to each axis’ commanded torques. The simulation was then run three different times using the same initial conditions as the cross-coupling section, but with one set of gains or the delays applied. The timing between these figures and the nominal versions will be different, thanks to the inclusion of the margins, but the instabilities can still be verified.

To illustrate that the maximum allowable delay in the system was correctly predicted by the TDAPPS algorithm, unit delay blocks were added to the original simulation. The simulation was then run in the cross-coupling configuration for 200 seconds to examine the vehicles response with a 1.1 second delay between the controller’s command and the RCS jet’s response. This was the maximum delay found for each axis from the TDAPPS results although they occurred at different times throughout the simulation. Figures 4.18 and 4.19 show the pitch axis response with added delay. The trajectory in Figure 4.18 starts at the hollow black circle and the total error is clearly increasing as the trajectory is moving further and further away from the target error of zero.
Figure 4.18 – Pitch axis phase-plane plot with 1.1 seconds of delay.

The proof that the controller is close to a stability limit can be seen in Figure 4.19, which is again the pitch-axis phase-plane space but with only 0.9 seconds of delay in the system. In this plot, the trajectory is moving closer to the target, but barely. This indicates that the appropriate maximum phase margin would be somewhere between 0.9 and 1.1 seconds, meaning that TDAPPS may have overshot the exact margin. The observations made from the test show that there is room for improvement in the TDAPPS optimization method and/or parameters. The results may also be improved by increasing the frame rate of the simulation. Doing so would be costly in terms of computation time (on the order of 10 minutes per 100 Hz), however this is less of a concern due to available technology.
To prove that the positive gain margins were being predicted accurately, gain blocks were applied to the torque signals to double the RCS torque on the system. The minimum allowable gain margin in any of the three axes was 6 dB, which translates to a doubling in torque. While doubling a jet’s torque output would be a rare actual occurrence, it is easily simulated. The resulting plots are less interesting than some of the previous ones so most of them have been included in Appendix A for reference, but Figures 4.20 and 4.21 show that the gain in torque will increase the number of occurrences and the frequency of inefficient limit cycles. Figure 4.20 shows the yaw channel phase-plane plots for this simulation, in which the limit cycles are reached quickly and are indeed unstable. Figure 4.21 is proof that it only takes one brief negative jet burn to reduce the rate error by over 80%, but it also shows how a higher frequency limit cycle occurs. This statement can be proven by comparing these plots to those in Figure 4.3, which show the roll axis essentially starting out in a limit cycle. In Figure 4.3, the controller fires three jets in a span of 180
seconds while riding along the edge of being in a limit cycle. In this example, the same number of correction firings occur in about 140 seconds and it often crosses the stability condition boundary, becoming unstable. This seems to be solid evidence that doubling the torque will cause unstable reactions.

Figure 4.20 – Yaw axis phase-plane space from the doubled RCS torque simulation.
This portion of the study also helped to reveal another shortcoming of this initial version of the TDAPPS method. Since the cross-coupled torque components also have an increased magnitude their effects can be seen more easily in the form of the small vertical jumps that the error trajectory makes in the drift region. These jumps within any region represent a change in the angular acceleration error, which throws the previous TDAPPS projection off. In other words, prior to these jumps, the algorithm does not currently look ahead to see if another axis’ jet will be fired as the error moves from region to region and, therefore, will have been making projections that are slightly off from what actually happens. One way to improve this feature would be to add a cross-axis predictive ability that would provide an accurate value of angular acceleration as opposed to using a constant angular acceleration within each section. Adding this capability would likely require extensive effort, but a suggestion on accomplishing this would be to add a logic structure capable of reading the jet commands, angular rate errors, and total error in all three axes.

Figure 4.21 – Yaw axis total error and torque plots from the doubled RCS torque simulation.
Finally, the predicted negative gain margins will be studied. The negative gain margin correlates to a reduction in RCS torque, which is a far more realistic situation than accruing a half second of delay or doubling the RCS torque. It also happens to be the case where the most obvious and egregious instabilities occur. In Figures 4.22 and 4.23, the yaw axis results are shown with a gain of 0.25 applied to the RCS torque. It is evident in this case that the vehicle’s response is unstable. In the plot of the phase-plane space (Figure 4.22), the hollow black circle represents the starting point of the error and the red trajectories are clearly spiraling away from the target. Figure 4.23 offers further proof of this because the amplitude of the total error is steadily increasing.

![Figure 4.22 – Yaw axis phase-plane space from the quartered RCS torque simulation.](image-url)
The most valuable observation from these two figures, however, are how close together the error is from the stability boundary. The error trajectory loops from the phase-plane plot are growing in diameter only slightly. The same can be said for the magnitude of the amplitude of total error in that plot. To further prove that TDAPPS has reached a reasonable approximation of the stability boundary in this case, the same simulation was run, but with an RCS torque gain of 0.3 instead of 0.25. Figures 4.24 again shows the yaw axis, but the results of this simulation are just on the stable side of the stability boundary, since the trajectory of the error in the phase-plane space is slowly making its way toward the center of the graph. This is not to say that any gain margin just below the minimum allowable is desirable. Obviously, a system like the one portrayed in Figure 4.24 is undesirable, but it clearly supports the results of the TDAPPS algorithm for this experiment.
Figure 4.24 – Yaw axis phase-plane plot with gain of 0.3 applied to RCS torque.

One question that these tests raises about the TDAPPS process is whether the stability condition needs to be tweaked to discover more instabilities and inefficiencies. Specifically, does the look-ahead time set by the length of a single analysis cycle (section 3.1) need to be tweaked to catch all the small instabilities that occur? This would be one area where further testing of the algorithm may be useful. However, it may also be unique to any given system. For instance, an object trying to re-orient itself for re-entry or orbit insertion may have more time to damp out its oscillations than a capsule trying to dock with the space station or an orbital imager trying to turn to face an object while in orbit. Either way, continued evaluation of the definition of stability is needed.

The results of each of these three tests all provide support that the results and observations from the TDAPPS method are reasonable and could be useful to someone analyzing a phase-plane controller. Studying these results also provided some insight on how to improve the current
algorithm. The first and likely more difficult improvement suggested is to incorporate a method for predicting future angular acceleration values prior to predicting the next segment of a trajectory. The current algorithm assumes that the torque acting on the system is constant depending on what region of the phase-plane space the error is in. Removing this simplification would allow for better projections of the angular rate following the completion of a cycle, thus providing a higher level of confidence in the results. Another improvement would be to improve the granularity of the provided gain and phase margins. This would require improving the optimization algorithms that calculate these gains, or switching to a different optimization method altogether. The final improvement suggestion may be somewhat subjective as it involves re-defining the definition of stability. While comparing error rates is a straightforward method to define stability, this experiment has also shown that this will label sub-optimal limit cycles as stable. Another way to approach this would be to separate the phase-plane space into two different regimes where the attitude rate error is the separating factor. This would separate normal phase-plane performance from the limit cycles that were commonly seen in these tests. The definition of stability and the tolerable gains could be different in each regime. It is planned to continue investigating these and other options for improving this algorithm; the overall results of this proof-of-concept study are encouraging for TDAPPS. In its current form, the TDAPPS method may not be ready to replace any existing and proven method for calculating the margins of a non-linear system, but the author believes that it has shown its worth as an easy to understand analysis tool to aid in the investigation into such systems.
5. Conclusions

The purpose of this paper was to introduce and validate a novel method of phase-plane controller stability analysis using a unique method called the Time Domain Analysis of Phase-plane Stability (TDAPPS). The intent for the initial development of this method was to create a stability analysis tool that could be simple to use and understand, while still providing the information required to quantify a controller’s stability margins. A brief history of phase-plane control in industry was provided along with a high-level description of how a phase-plane controller works. This was followed by a detailed description of the TDAPPS algorithm, including the math and logic involved in its development. Next, the TDAPPS method was compared to the current, most commonly used method for this analysis, known as the Describing Function Method. Here, it was determined that both approaches had distinct advantages and disadvantages and that the TDAPPS method was worthy of further development. After describing the simulation environment, a set of basic and, subsequently, more advanced tests were conducted to obtain initial TDAPPS results. The resulting phase and gain margins were tested in the same simulation environment and their merit was discussed along with several ways that the tool could be improved.

The conclusions that can be drawn from this proof-of-concept study are mostly positive for the TDAPPS algorithm. First and foremost, it provided justifiable values for the system delay and gain margins in the time domain. Combine the relatively simple set up and use with unique visualization methods, and the value of TDAPPS for system analysis and design is apparent. However, several areas for improving the current algorithm were identified throughout this study, including (but not limited to) finding better ways to measure or predict vehicle angular acceleration, improving the embedded optimization algorithms, and fine-tuning the definition of what constitutes an analysis.
cycle. Resolving these outstanding issues is left as future work for this project, but it is believed that the current body of work stands as proof that the TDAPPS method is a worthwhile addition to the repertoire of non-linear analysis tools.
6. References


function [Stab_flag, mdt, T_gain, a_vec] = TDAPPS(accel, rate, att, DB, hyst, jet_flag, dt, a_t, axis_id)

% TDAPPS - Predicts whether a phase plane controller is stable and
% finds its stability margins (thrust gain/time delay).
% [Stab_flag, mdt, T_gain, a_vec] = TDAPPS(accel, rate, att, DB,
% hyst, jet_flag, dt, a_t)
% This function takes phase plane controller simulation data and
determines whether the controller is stable under the given conditions and then
determines what time delay would cause an unstable response.
% Inputs:
% accel = angular acceleration (deg/s^2) (accel = torque/inertia)
% rate = vehicle ang. velocity error, integrated from alpha (deg/s)
% att = vehicle ang. attitude error, integrated from omega (deg)
% DB = Deadband location (degrees)
% hyst = Hysteresis value (degrees)
% jet_flag = string that describes whether an RCS jet is active/
inactive
active (1= pos. jet active; -1= neg. jet active; 0= no jets)
% dt = length of known time delay in ms (time b/w command
issued and
response)
% a_t = target attitude value to set deadbands about
% axis_id = A parameter defined by the user to select
which
pitch
axis, 3-yaw axis
% Output:
% Stab_flag = Denotes whether controller is stable/unstable (1 =
stable;
0 = unstable) at this time step
% mdt = time delay margin at which controller becomes unstable
(delay
(b/w command and response) at this time step
% T_gain = [2,1] array of [neg, pos] gain values for the entire
system
% a_vec = vector of alpha values for each region of the phase-
plane
controller; [a_drift, a_pos, a_pos2, a_neg, a_neg2]
% Local Variables:
% w_prev = (Persistent var); value of omega at 1st projected
intersection
w/ a firing curve from the previous iteration
% mdt_prev = (Persistent var); max. time delay margin from prev. iteration
% g_prev   = (Persistent var); T_gain values from prev. iteration
% alpha    = vector of alpha values from update_alpha
% rateP    = proj. omega values at points of intersection w/ firing curves
% attP     = proj. theta values at points of intersection w/ firing curves
% tol      = tolerance value for comparing current omega_i w/ w_prev

% Required Functions:
%   - fit_curve.m
%   - traj_project.m
%   - get_intersect.m
%   - get_roots.m
%   - max_delay.m
%   - thrust_gain.m

% 2017-01-03 Jacob Reedy, Adaptive Aerospace Group, Inc.
%            <jreedy@adaptiveaero.com>
% © 2017 Adaptive Aerospace Group, Inc.

addpath('./Scripts')

% Initialize function outputs or Simulink
rateP = NaN;
attP = NaN;

% Set tolerance for detecting change in previous intersection point
tol = 1e-3;

% Create persistent variable to mark the first pass through the function
persistent w_prev mdt_prev g_prev

% Predefine persistent vars
if isempty(w_prev)
    w_prev = NaN;
    mdt_prev = NaN;
    g_prev = [NaN, NaN];
end

% Retrieve/Update angular acceleration for each region of the given axis
SimIn = evalin('base', 'SimIn'); % Retrieve SimIn structure from workspace
alpha = get_alpha(SimIn, accel, jet_flag, axis_id);

% Project (forward and back to most recent) the value of angular velocity error
[rateP, attP] = traj_project(alpha, rate, att, DB, hyst, jet_flag, dt, a_t);
Determine the stability of the controller based on diff in omega

Find $d_{\omega}$ (change in magnitude b/w final and initial omega value)

\[
\begin{align*}
d_{\omega} &= \text{abs}(\text{rateP}(\text{end})) - \text{abs}(\text{rateP}(1)); \\
\% \text{ If the original rate is less that 0.1 deg/sec, consider the vehicle} \\
\% \text{ stable until it exceeds this value} \; \% \text{ Still calculate gain and delay} \\
\% \text{ margins} \\
\text{if rate < 0.1 \% deg/sec} \\
\quad \text{Stab_flag = 1}; \; \% \text{ considered stable if rate is slow enough} \\
\text{else} \\
\quad \begin{align*}
\text{if} \; d_{\omega} <= \text{tol} \quad \% \text{ Stable} \\
\quad \text{Stab_flag = 1;} \\
\text{else} \\
\quad \text{Stab_flag = 0}; \; \% \text{ Unstable}
\end{align*}
\end{align*}
\]

Find maximum allowable time delay and pos/neg gain margins

\[
\begin{align*}
\text{if isnan(w_prev)} & \quad \% \text{ lst pass - run max_delay } \\
\text{max_gain} & \\
\quad \text{mdt = max_delay(alpha, rateP(1), attP(1), DB, hyst, jet_flag, dt, rateP(end), a_t);} \\
\quad \text{T_gain = thrust_gain(alpha, rateP(1), attP(1), DB, hyst, jet_flag, dt, rateP(end), a_t);} \\
\text{else} & \quad \% \text{ Check if init intersect has} \\
\text{changed} & \quad \% \text{ Check if final projected omega = 0} \\
\quad \text{if rateP(end) == 0} \quad \% \text{ If not, check if it has changed from the last value} \\
\quad \quad \text{if abs(rateP(1) - w_prev) <= tol} \quad \% \text{ Dont run max_delay (gets} \\
\quad \quad \text{same result)} \\
\quad \quad \text{mdt = mdt_prev;} \\
\quad \quad \text{T_gain = g_prev;} \\
\quad \text{else} & \quad \% \text{ Run max_delay with new conditions} \\
\quad \quad \text{mdt = max_delay(alpha, rateP(1), attP(1), DB, hyst, jet_flag, dt, rateP(end), a_t);} \\
\quad \quad \text{T_gain = thrust_gain(alpha, rateP(1), attP(1), DB, hyst, jet_flag, dt, rateP(end), a_t);} \\
\text{end} & \quad \% \text{ If omega_final is 0, force to previous value of max time delay} \\
\text{else} & \quad \% \text{ If final projected w = 0, mdt cannot} \\
\quad \text{mdt = mdt_prev;} \quad \% \text{ currently be updated} \\
\quad \text{T_gain = g_prev;} \\
\text{end}
\end{align*}
\]
% Set previous values of persistent vars for the next pass through
% TDAPP5
w_prev  = rateP(1);
mdt_prev = mdt;
g_prev  = T_gain;

a_vec = alpha;

Published with MATLAB® R2017a
function [alpha, theta0] = fit_curve(omega, theta)

% FIT_CURVE Fit phase plane trajectory through samples
% [alpha, theta0] = FIT_CURVE(omega, theta) will find parameters
% describing a phase plane trajectory assuming constant
% torque-to-inertia ratio (alpha). Other identified parameter is
% attitude angle at zero rate (theta0) where
% theta = (1/(2*alpha))*omega + theta0
% for a given set of samples of omega and theta.
% Algorithm from "Numerical Recipes in C," 2nd ed, p 665
% for y = a + b*x
% where x = omega^2
% y = theta
% a = theta0
% b = 1/(2*alpha)
%
% Note: this can be solved in Matlab more simply by using
% P = theta \ [ones(size(omega)) omega.^2];
% if theta and omega are column vectors; resulting column vector P can
% be decomposed as [theta0, 1/(2*alpha)]. However, the algorithm
% contained in this function lends itself to hand-coding or
% real-time operation.

% 2017-01-03 Bruce Jackson, Adaptive Aero Group, Inc.
% <mailto:bjackson@adaptiveaero.com> (c) 2017 AAG, Inc.

Check for proper number of arguments

if nargin ~= 2
   error('Must provide two arguments');
end

if length(omega) ~= length(theta)
   error('Omega and theta must have the same length');
end

Matrix method

p = [omega.*omega ones(size(omega))] \ theta;
if abs(p(1)) <= 1e-6
   alpha = 0;
else
   alpha = 1/(2*p(1));
end
end

theta0 = p(2);

Published with MATLAB® R2017a
function [omegaP, thetaP] = traj_project(alpha, omega, theta, DB, hyst, jet_flag, dt, a_t)
% TRAJ_PROJECT - Uses the current error and phase plane data to project the
% trajectory of the error through n number of commands
% [omegaP, thetaP] = TRAJ_PROJECT(alpha, omega, theta, DB, hyst,
% jet_flag, dt, a_t)
% This function takes phase plane controller simulation data and projects
% the omega and theta error trajectory through n number of commands, or
% changes in jet state. The output is the projected omega and theta
% values.
%
% Inputs:
% alpha = vector of alpha values for each of the phase plane
% regions
% omega = vehicle ang. velocity error, integrated from alpha (deg/s)
% theta = vehicle ang. attitude error, integrated from omega (deg)
% DB = Deadband location (degrees)
% hyst = Hysteresis value (degrees)
% jet_flag = string that describes whether an RCS jet is active/
% inactive (1- pos. jet active; -1- neg. jet active; 0- no jets
% active)
% dt = length of known time delay in ms
% a_t = target attitude value to set deadbands about
%
% Output:
% omegaP = projected omega vector after n intersection points
% thetaP = projected theta vector after n intersection points
%
% Local Variables:
% n = number of projected pnts to be calc. (# of changes in
% region)
% alpha1 = initial value of alpha; corresponding to current region
% a_local = vector of alpha values of length n, pre-defined to 0 for
% projected values
% tnn = projected theta vector from get_intersect
% vn = projected omega vector from get_intersect
% jfn = projected jet_flag value of next command from
% get_intersect
%
% Required Functions:
% - get_intersect.m
% - get_root.m

% 2017-01-09 Jacob Reedy, Adaptive Aerospace Group, Inc.
% <jreedy@adaptiveaero.com>
% © 2017 Adaptive Aerospace Group, Inc.

% Initialize function outputs or Simulink
tn = NaN;
wn = NaN;
jfn = NaN;

% Define the number of intersection points in a cycle
n = 6;

switch jet_flag
    case 0
        alphal = alpha(1); % Choose drift region alpha
    case 1
        alphal = alpha(2); % Choose pos. jet firing region
    case -1
        alphal = alpha(3); % Choose neg jet firing region
    otherwise
        error('Invalid jet_flag value.')
end

% Initiate state variables
a_local = [alphal, zeros(1, n-1)];
omega = [omega, zeros(1,n-1)];
theta = [theta, zeros(1,n-1)];
jet_flag = [jet_flag, zeros(1,n-1)];

% Add 'for' loop to streamline things
for i = 2:n
    % Find the next point of intersection in the trajectory
    [tn, wn, jfn] = get_intersect(a_local(i-1), omega(i-1),
                                  theta(i-1), DB, hyst, jet_flag(i-1), dt, a_t);

    % Check if result is valid
    if isnan(wn(end)) % Does not intersect firing curve
        wn(end) = abs(omega(1)) + 1; % Ensure it is unstable
        break
    end

    % Set new state values
    omega(i)  = wn(end);
    theta(i)  = tn(end);
    jet_flag(i) = jfn;

    % Place the starting point at the previous point of intersection to
    % compare for stability purposes (if at point of intersection the
    % point will remain the same)
    if i == 2
        omega(1) = wn(1);
        theta(1) = tn(1);
    end
% Determine new value of alpha based on new region
switch jet_flag(i)
  case 0
    a_local(i) = alpha(1);  % Choose drift region alpha
  case 1
    a_local(i) = alpha(2);  % Choose pos. jet firing region
  case -1
    a_local(i) = alpha(3);  % Choose neg jet firing region
  otherwise
    error('Invalid jet_flag value.')
  end
end

omegaP = omega;
thetaP = theta;

Published with MATLAB® R2017a
function \([\text{theta}_P, \text{omega}_P, \text{jet\_flag\_new}] = \text{get\_intersect}(\text{alpha}, w_l, \text{the1}, DB, \text{hyst}, \text{jet\_flag}, dt, a_t)\)

% GET_INTERSECT - Finds the intersection b/w a trajectory and firing curve.
% \([\text{theta}_P, \text{omega}_P, \text{jet\_flag\_new}] = \text{GET\_INTERSECT}(\text{alpha}, w_l, \text{the1}, DB, \text{hyst}, \text{jet\_flag}, dt, a_t)\)
% \% Find the intersection between the attitude error trajectory and the local
% \% firing curve. For a stable system there will either be 1, or 2
% \% points of
% \% intersection. These will be the points at which the firing command
% \% changes and this function provides the curve projection and
% \% intersecting
% \% point that is not equal to the input point.
% \% \% INPUT Description:
% \% alpha = Ang. acc. of attitude in the region (assumes const)
% \% w_l = omega value at the initial crossing
% \% the1 = theta value at the initial crossing
% \% DB = Deadband location (degrees)
% \% hyst = Hysteresis value (degrees)
% \% jet\_flag = flag that denotes which jet is active
% \% \[-1 = \text{Neg Jet}; 1 = \text{Pos Jet}; 0 = \text{No Jet}\]
% \% dt = length of time delay in ms
% \% (time b/w command issued and response)
% \% a_t = target attitude value to set deadbands about
% \% \% OUTPUT Description:
% \% theta_P = projected values of theta throughout the current
% \% region
% \% omega_P = projected values of omega throughout the current
% \% region
% \% jet\_flag\_new = designates which region the traj is in after the
% \% intersection occurs
% \% \% Local Variables:
% \% A = constant for parabolic equation
% \% B = theta value on parabola when omega equals 0
% \% m = slope of firing curve and hyst lines (for slope int. eqn.)
% \% b_pos = theta intercept value of positive firing curve
% \% b_neg = theta intercept value of negative firing curve
% \% b_hpos = theta intercept value of positive hysteresis curve
% \% b_hneg = theta intercept value of negative hysteresis curve
% \% b_n = y-intercept value of the NEXT point of intersection
% \% b_l = y-intercept value of the LAST point of intersection
% \% B\_crit motion = value of theta when omega = 0 for which the parabolic
% \% will be tangent to the corresponding firing curve
Calculate coefficients for governing equations

Eqn for parabola is \[ \theta = A\cdot w^2 + B \] (B is theta at \( w=0 \))

```matlab
if alpha == 0
    A = 1/(2*alpha);
    B = theta1 - (A*wl^2);
else
    A = 0;
    B = 0;
end
```

Eqn for linear firing curve can be obtained from \[ y = mx + b \] by solving
for \( x \rightarrow \theta = (1/m) \cdot (w - b) \)

Get firing curve information (\( w = m(\theta) + b \))

\( m = -1 \); \% Firing curve slope (Assuming theta & omega limits are equal)

\% Firing curve intersects \( b \) along the theta axis (will include hysteresis
\% if in one of the jet firing regions (jet firings do not stop until past
\% the hysteresis line but they do not start until past the FCs)

\( b_{pos} = a_t - DB; \quad b_{neg} = a_t + DB; \)
\( b_{hpos} = b_{pos} + hyst; \quad b_{hneg} = b_{neg} - hyst; \)
\[
\begin{align*}
\text{if } \text{jet\_flag} \equiv 0 \\
\text{bn} &= b\_neg \\
\text{else} \\
\text{bn} &= b\_hneg \\
\end{align*}
\]

% Calculate B\_crit - value of theta0 where the parabola inside the drift region is tangent to the firing curve.
% Calculation comes from setting the portion of the quadratic eqn under the radical equal to 0 in order to find the point at which the parabola only intersects the firing curve at 1 point. It is forced to be positive in order to ensure that the value will be inside of the drift zone \% \{B\_crit < maxE\}. This is valid assuming the phase plane controller is symmetrical as it will be the same for either the pos or neg FCs. \nB\_crit = abs(bn/m) - (abs(alpha)/(2*m^2));

% Determine which region the error is currently in and which firing curve \% will be crossed.
\[
\begin{align*}
\text{if } \text{jet\_flag} \equiv -1 & \quad \% \text{Error is in Negative Jet Firing Region} \\
\text{traj\_shape} &= 1; & \quad \% \text{Trajectory is parabolic} \\
\text{bn} &= b\_hneg; & \quad \% \text{Next int. pnt is on Neg Hyst Curve (HC)} \\
\text{bl} &= b\_neg; & \quad \% \text{Last int. pnt is on Neg Firing Curve (FC)} \\
\text{elseif } \text{jet\_flag} \equiv 1 & \quad \% \text{Error is in Positive Jet Firing Region} \\
\text{traj\_shape} &= 1; & \quad \% \text{Trajectory is parabolic} \\
\text{bn} &= b\_hpos; & \quad \% \text{Next int. pnt is on Pos Hyst Curve (HC)} \\
\text{bl} &= b\_pos; & \quad \% \text{Last int. pnt is on Pos FC} \\
\text{elseif } \text{jet\_flag} \equiv 0 & \quad \% \text{Error is in Drift Region} \\
\text{if } \text{alpha} \equiv 0 & \quad \% \text{No torques acting on system} \\
\text{traj\_shape} &= 0; & \quad \% \text{Trajectory is a straight line} \\
\text{if } w1 \equiv 0 & \quad \% \text{Error heading towards Neg FC} \\
\text{bn} &= b\_neg; & \quad \% \text{Next int. pnt is on Neg FC} \\
\text{bl} &= b\_hpos; & \quad \% \text{Last int. pnt is on Pos HC} \\
\text{else } w1 < 0 & \quad \% \text{Error heading towards Pos FC} \\
\text{bn} &= b\_pos; & \quad \% \text{Next int. pnt is on Pos FC} \\
\text{bl} &= b\_hneg; & \quad \% \text{Last int. pnt is on Neg HC} \\
\text{end} \\
\text{elseif } \text{alpha} \equiv 0 & \quad \% \text{Positive torque acting on system} \\
\text{if } w1 \equiv 0 & \quad \% \text{Error heading towards Neg FC} \\
\text{if } B > b\_pos & \quad \% \text{Upper section of parabola in drift reg} \\
\end{align*}
\]
\begin{verbatim}
traj_shape = 1;
bn = b_neg; \% Next int. pnt is on Neg FC
bl = b_hneg; \% Last int. pnt is on Neg HC
else \% Pos FC -> Neg FC (no apex)
  traj_shape = 2; \% Not Parabolic in drift region
  bn = b_neg; \% Next int. pnt is on Neg FC
  bl = b_hpos; \% Last int. pnt is on Pos HC
end

else \% w1 < 0 \% Upper section of parabola in drift
  reg
    if abs(B) <= abs(B_crit - a_t)
      traj_shape = 1; \% Trajectory is parabolic
      bn = b_neg; \% Next int. pnt is on Neg FC
      bl = b_hneg; \% Last int. pnt is on Neg HC
    else \% abs(B) > abs(B_crit - a_t)
      acc > 0
        bn = b_pos; \% Next int. pnt is on Pos FC
        bl = b_hneg; \% Last int. pnt is on Neg HC
      end

else \% alpha < 0 \% Negative torque acting on system
  if w1 <= 0 \% Error heading towards Pos FC
    reg
      if B < b_neg \% Lower part of parabola in drift
        traj_shape = 1;
        bn = b_pos; \% Next int. pnt is on Pos FC
        bl = b_hpos; \% Last int. pnt is on Pos HC
      else
        traj_shape = 3; \% Neg FC -> Pos FC (no apex); acc < 0
        bn = b_pos; \% Next int. pnt is on Pos FC
        bl = b_hneg; \% Last int. pnt is on Neg HC
      end

else \% w1 > 0 \% Drift Reg
  if abs(B) <= abs(B_crit + a_t) \% Parabolic inside
    traj_shape = 1; \% Trajectory is parabolic
    bn = b_pos; \% Next int. pnt is on Pos FC
    bl = b_hpos; \% Last int. pnt is on Pos HC
  else \% abs(B) > (B_crit + a_t)
    \% Error crosses straight through Drift Region
    traj_shape = 2; \% Pos FC -> Neg FC (no apex)
    bn = b_neg; \% Next int. pnt is on Pos FC
\end{verbatim}
find points along the desired curve if angular acc. is zero or non-zero

if traj_shape == 0  % Trajectory of interest is a straight line
   % (w-Const)
   % When alpha is 0, omega is constant and equal to w1 so there will be one
   % point of intersection along the given firing curve and it can be found by
   % solving for theta at the provided omega.
   w_int = w1; the_int = (1/m)*(w_int-bn) + (w_int * dt);
   the_prev = (1/m)*(w_int-bl) + (w_int * dt);
   omega_P = [w1, w_int];
   theta_P = [the_prev, the_int];
elseif traj_shape == 1  % alpha ~= 0 (gives a parabolic curve)
   % Find intersecting points of theta (parabolic response) and thetaf (firing
   % curve) by setting theta and thetaf equal to each other and using the
   % quadratic formula -> 0 = Aw^2 + (-w/m) + (B+(bf/m))
   wPn = get_roots(A,B,m,bn);  % bn is for NEXT int. point
   wPl = get_roots(A,B,m,bl);  % bl is for LAST int. point
   % Determine the past point of intersection vs the upcoming point
   if jet_flag == -1  % In Neg Firing Region
      w_int1 = max(wPl) + (alpha * dt);  % w1 is > 2 values on FC
      w_int2 = min(wPn) + (alpha * dt);  % w2 is < of 2 values on
      hyst   % plus the time delay
      term
   elseif jet_flag == 1  % In Pos Firing Region
      w_int1 = min(wPn) + (alpha * dt);  % w1 is < of 2 values on
      w_int2 = max(wPl) + (alpha * dt);  % Opposite of Neg Firing
      Reg.
   else  % jet_flag == 0  % In drift region and b lies on FC (not
      hyst)
      if alpha > 0  % Parabola concave towards Neg FC
         w_int1 = min(wPl) + (alpha * dt);  % w is < of 2 values on
      hyst
   end
end
w_int2 = max(wPn) + (alpha * dt); % w2 is > of 2 values on DB
else % alpha < 0 % Parabola concave towards Neg FC
    w_int1 = max(wP1) + (alpha * dt); % w1 is > of 2 values on hyst
    w_int2 = min(wPn) + (alpha * dt); % w2 is < of 2 values on DB
end

omega_P = linspace(w_int1, w_int2); % Project whole curve in current reg.
theta_P = A.*(omega_P.^2) + B;

elseif traj_shape == 2 % Pos FC -> Neg FC (no apex); acc > 0
    wPn = get_roots(A, B, m, bn); % Get roots for NEG FC
    wP1 = get_roots(A, B, m, bl); % Get roots for POS HC
    w_int1 = max(wP1) + (alpha * dt); % w1 is > of 2 values on HC
    w_int2 = max(wPn) + (alpha * dt); % w2 is > of 2 values on FC
    omega_P = linspace(w_int1, w_int2); % projected omega values
    theta_P = A.*(omega_P.^2) + B; % projected theta values

elseif traj_shape == 3 % Neg FC -> Pos FC (no apex); acc > 0
    wPn = get_roots(A, B, m, bn); % Get roots for POS FC
    wP1 = get_roots(A, B, m, bl); % Get roots for NEG HC
    w_int1 = min(wP1) + (alpha * dt); % w1 is < of 2 values on HC
    w_int2 = min(wPn) + (alpha * dt); % w2 is < of 2 values on FC
    omega_P = linspace(w_int1, w_int2); % projected omega values
    theta_P = A.*(omega_P.^2) + B; % projected theta values

else % traj_shape == 4 (error is hovering along the FC at low rates)
    omega_P = -w1; theta_P = -theta1; % Error is in a stable but barely % changing cycle about the FC
end

if omega_P(1) == 0 && omega_P(end) == 0 % Not intersecting curve
    omega_P = NaN;
end

% Determine new jet flag based on which region the new coordinates are
in switch jet_flag
    case -1
        jet_flag_new = 0; % From Neg Firing Region to Drift Region
case 1
    jet_flag_new = 0;  % From Pos Firing Region to Drift Region
end

case 0  % Need to determine which FC it will intersect
    if bn >= b_hneq  % Traj will cross Neg FC
        jet_flag_new = -1;
    elseif bn < b_hpos  % Traj will cross Pos FC
        jet_flag_new = 1;
    else  % Traj will stay in the current region
        jet_flag_new = jet_flag;
end

Published with MATLAB® R2017a
function r = get_roots(A, B, m, b)
% GET_ROOTS - Finds intersections b/w linear firing curve & phase plane err
% [omega_intersect] = GET_ROOTS(A, B, m, b)
% Find intersecting points of theta (parabolic eqn -> theta = Aw^2 + the_0)
% and thetaf (linear firing curve -> thetaf = mw + b) by setting theta and
% thetaf equal to each other and using the quadratic formula.
% => 0 = Aw^2 + (-w/m) + (B+(bf/m))
% % INPUTS:
% A = 1/(2*alpha) where alpha is applied vehicle angular acc (from torque)
% B = theta when omega = 0, aka. theta intercept value
% m = -1 for most linear phase plane firing curve boundaries
% b = Firing Curve intercept on the attitude axis; depends on which FC or
%     HC is being crossed (pos/neg FC/HC)
% % OUTPUTS:
% r = two roots (w1_P and w2_P) which are omega values at points of
%     intersection b/w the firing curve and error trajectory. Which
%     root(s) are
%     needed is then decided by logic outside of the function.

% 2017-01-03 Jacob Reedy, Adaptive Aerospace Group, Inc.
%    <jreedy@adaptiveaero.com>
% % © 2017 Adaptive Aerospace Group, Inc.

% Quadratic Formula -> (-Bq +/- sqrt(Bq^2 - 4*Aq*Cq))/(2*Aq)
Aq = A;
Bq = -1/m;
Cq = B+(bf/m);

% If the portion of the quadratic function under the radical is real then
% the parabola crosses the FC at least once, if it is unreal it does not
% cross at all
term = Bq^2 - (4 * Aq * Cq);
if term > 0
    root = sqrt(term);
    % Calculate points of intersection between the lines
    w1_P = (-Bq + root)/(2*Aq); w2_P = (-Bq - root)/(2*Aq);
else
    % No intersection points
    w1_P = 0; w2_P = 0;
end
end

% Output Roots
r = [w1_P, w2_P];

Published with MATLAB® R2017a
function mdt = max_delay(alpha, omega, theta, DB, hyst, jet_flag, dt, w_f, a_t)

% MAX_DELAY - Determines the maximum allowable time delay for the
% provided
% phase plane controller conditions
%
% [mdt] = MAX_DELAY(alpha, omega, theta, DB, hyst, jet_flag, dt, w_f, a_t)
%
% This function determines the maximum allowable time delay between the
% phase plane controller command and vehicle response before the system
% becomes unstable. It takes a 'snapshot' of the controllers state and
% projects the error values, using the traj_project function. After each
% iteration it determines a new delay value to try (Newtons opt. method)
% and re-runs. The function stops when the delta_omega value is
% <=0.01. The delay value that achieves this is the max allowable time delay.
%
% Inputs:
% alpha = vector of alpha values for each of the phase plane
% regions
% omega = vehicle ang. velocity error, integrated from alpha (deg/s)
% theta = vehicle ang. attitude error, integrated from omega (deg)
% DB = Deadband location (degrees)
% hyst = Hysteresis value (degrees)
% jet_flag = string that describes whether an RCS jet is active/inactive
%   (1- pos. jet active; -1- neg. jet active; 0- no jets active)
% dt = length of known time delay in controller (in ms)
% w_f = final omega value from the traj_project function using all
% the above conditions
% a_t = target attitude value to set deadbands about
%
% Output:
% mdt = maximum allowable time delay (in sec) that can be added to dt
% b/w the issued command and the systems response before controller
% becomes unstable
%
% Local Variables:
% tol = convergence tolerance for the optimization algorithm
% target = target value for the cost function
% x = gain for altering the calc delta value
% dB_max = Maximum gain (in dB) beyond which it is not useful to calculate
% d_omega = change (mag.) in initial omega and final projected omega values
% delta = change in time delay to be added to the delay
% d_max = upper bound for delay; initially set to 10 sec, updated in-loop
% d_min = lower bound for delay; initially set to 0 sec, updated in-loop
% delay = amount of delay to be used in the trajectory projection
% w_new = the updated projected omega values after the delay has changed
%
% Required Functions:
% - get_intersect.m
% - get_roots.m

% 2017-01-09 Jacob Reedy, Adaptive Aerospace Group, Inc.
% <jreedy@adaptiveaero.com>
% © 2017 Adaptive Aerospace Group, Inc.

% Set optimization parameters
tol = 1e-5; % algorithm convergence tolerance
target = 0.0; % target value for d_omega
k = 0.95; % gain for altering calculated delta value
dB_max = 12; % Maximum time delay margin (in dB)

% Calculate the initial difference in final and initial omega
omega = abs(w_f) - abs(omega);

% Initialize the delta value for updating the delay
if d_omega <= 0 % Controller is stable at current time delay
    delta = 500; % need to add delay -> +delta (in ms)
else % Controller is unstable at current time delay
    delta = -500; % need to subtract delay -> -delta (in ms)
end

% Initialize the bounds for the delay term (will be changed in loop)
d_max = 10000; % Max. delay value in ms (set to 10 sec to start)
d_min = 0; % Min. delay value (cannot have negative delay)

% Initialize delay - delta (in ms)
delay = dt + delta;

if delay < d_min % delay cant be neg so set to 0 if necessary
    delay = 0;
end

% Set max # of iterations
n = 51; % max # of iterations is 50 (starts @ 2)

% Initialize the variables in the for_loop
w_f = [w_f, zeros(1, n-1)];
next = [next, zeros(1, n-1)];
delay = [delay, zeros(1, n-2)];

% Create iteration counter
count = 0;

for i = 2:n
    count = count + 1; % Update iteration counter
    % Calculate new delta_omega - only changing the value of the time delay
    w_new = traj_project(alpha, omega, theta, DB, hyst, jet_flag, delay(i), a_t);
    w_f(i) = w_new(end);
    d_omega(i) = abs(w_f(i)) - abs(omega);

    % Check for convergence (where w_f is slightly greater than w_i i.e.
    % where d_omega > 0.
    if abs(d_omega(i) - target) <= tol % > 0 but <= to 0.01
        break % Considered converged; report current delay value
    elseif d_omega(i) > target % Set max delay bound
        d_max = delay(i);
    else
        d_min = delay(i);
    end

    % Update delta using Newton's Optimization method
    x1 = d_omega(i-1); x2 = d_omega(i);
    dx = x2 - x1; % Change in d_omega (delta = change in delay)
    dxd = target - x2; % Desired change to reach the target d_omega
    if abs(dx) >= tol % Protect against dividing by 0
        delta = (delta/dx)*dxd; % Update delta value
        delta = k * delta; % Apply opt. gain to delta
    else
        delta = 0.01; % Set to small but non-zero value
    end

    % Ensure loop doesn't reach for an index outside the provided bounds
    if i <= (n-1)
        delay(i+1) = delay(i) + delta; % Update delay for next iteration
    else
        delay(i+1) = k * d_max; % Set to lower than d_max bound
    end
    if delay(i+1) > d_max % Delay is higher than max bound
        delay(i+1) = k * d_max; % Set to lower than d_max
    elseif delay(i+1) < d_min % Delay is lower than min bound
        delay(i+1) = (2-k) * d_min; % Set to higher than
    else
        break % Considered converged; report current delay value
    end
% delay(i+1) falls w/in bounds (use line 118 value)
    end
else
    break
end
% Determine current gain in decibels and check if greater than max
dt_g = 10*log10(delay(i)); % Pos gain value (in dB) for controller
if dt_g >= dB_max
    break
end

mdt = (delay(i)-dt)/1000; % Maximum time delay in ms
Table of Contents

Find maximum thrust gain - only run if stable ................................................................. 1
Find minimum thrust gain - whether stable or not ......................................................... 2
Find Gain Value in decibels (subfunction 'get_T') ....................................................... 4

function T_gain = thrust_gain(alpha, omega, theta, DB, hyst, jet_flag, dt, w_f, a_t)

% THRUST_GAIN - Determines the maximum and minimum allowable gain in thrust
% for the provided phase plane controller conditions
% [T_gain] = THRUST_GAIN(alpha, omega, theta, DB, hyst, jet_flag, dt, w_f, a_t)
% This function determines the gain margins at which the thrust can be increased (T_gain > 1) and decreased (T_gain < 1) such that the phase plane controller becomes unstable. It applies a gain to the input 'alpha'
% vector until the instability criteria has been met. The final gain value
% is the output thrust gain value since angular acceleration and RCS thrust
% are proportional.
% Inputs:
% alpha = vector of alpha values for each of the phase plane regions
% omega = vehicle ang. velocity error, integrated from alpha (deg/s)
% theta = vehicle ang. attitude error, integrated from omega (deg)
% DB = Deadband location (degrees)
% hyst = Hysteresis value (degrees)
% jet_flag = string that describes whether an RCS jet is active/inactive
% dt = length of known time delay in controller (in ms)
% w_f = time b/w command issued and response
% a_t = target attitude value to set deadbands about
% Output:
% T_gain = [1,2] array where the T_gain(1) is the minimum allowable
% thrust gain & T_gain(2) is the maximum allowable thrust
gains are expressed in decibels (db)

Local Variables:
% tol = convergence tolerance for the optimization algorithm
% target = target value for the cost function
% k = gain for altering the calc delta value
% db_max = Max/min gain (in dB) beyond which it is not useful to calculate
% d_omega = change (mag.) in initial omega and final projected omega values
% G = gain value for the alpha vector
% delta = change in G to be added for the next iteration
% g_max = upper bound for G; initially set to 1 db, updated in-loop
% g_min = lower bound for G; initially set to 0 db, updated in-loop
% w_new = the updated projected omega values after the delay has changed
% T_pos = Positive gain margin in decibels
% T_neg = Negative gain margin in decibels

Required Functions:
% - get_intersect.m
% - get_roots.m

2017-01-09 Jacob Reedy, Adaptive Aerospace Group, Inc.
<jeeidy@adaptiveaero.com>
© 2017 Adaptive Aerospace Group, Inc.

Set optimization parameters
tol = 1e-5; % algorithm convergence tolerance
target = 0; % target value for d_omega
k = 0.95; % gain for altering calculated delta value
db_max = 12; % Max gain margin above/below which is out of scope

Calculate the initial difference in final and initial omega
d_omega = abs(w_f) - abs(omega);

Initial alpha and its gain (G = 1, no gain has been applied yet)
a1 = alpha;
G1 = 1;

Find maximum thrust gain - only run if stable

if (d_omega - target) <= tol % If stable - run pos gain calc.

if stable - negative gain margin is not applicable (at least w/o taking % other conditions into account i.e. fuel efficiency, max G, etc.)
T_neg = NaN;

Initiate the delta value for updating the alpha gain
delta = 0.5; % need to add gain -> +delta (in decibels)
% Initialize the bounds for the alpha gain (will be changed in loop)
g_max = 100;  % Max. delay value in db
g_min = 1;    % Min. delay value (cannot be < 1 for max gain)

% Initialize gain w/ first delta (in db)
G = Gi + delta;

% Set max # of iterations
n = 51;  % max # of iterations is 50 (starts @ 2)

% Initialize the variables in the for_loop
w_f = [w_f, zeros(1, n-1)];
d_omega = [d_omega, zeros(1, n-1)];
G = [G, G, zeros(1, n-2)];

% Create iteration counter
count = 0;

for i = 2:n
    count = count + 1;  % Update iteration counter

    % Update the original alpha with the new gain value
    alpha = G(i) * ai;

    % Calculate new delta_omega - alpha is the only change
    w_new = traj_project(alpha, omega, theta, DB, hyst, jet_flag, dt, a_t);
    w_f(i) = w_new(end);
    d_omega(i) = abs(w_f(i)) - abs(omega);

    % Check for convergence (where w_f is slightly greater than w_i
    % i.e. % where d_omega > 0.
    if abs(d_omega(i) - target) <= tol % d_omega is just barely > 0
        % Determine current gain in decibels and check if greater than
        max
        T_pos = get_T(G(i), dB_lim);
        break  % Considered converged; report current delay value
    elseif d_omega(i) > target  % Set max delay bound
        g_max = G(i);
    else
        g_min = 1;
    end

    % Update delta using Newtons Optimization method
    x1 = d_omega(i-1); x2 = d_omega(i);
    dx = x2 - x1;  % Change in d_omega (delta = change in gain
    - G)
    dxd = target - x2;  % Desired change to reach the target d_omega
    if abs(dx) >= tol  % Protect against dividing by 0
        delta = (dx/abs(dx)) * dxd;  % Update delta value
        delta = k * delta; % Apply opt. gain to delta
    else
        delta = 0.01;  % Set to small but non-zero value
end

% Ensure loop doesnt reach for an index outside the provided bounds
if i <= (n-1)
  G(i+1) = G(i) + delta;
  if G(i+1) > g_max
    G(i+1) = k * g_max;
    delta = G(i+1) - G(i);
    % Re-set delta
  elseif G(i+1) < g_min
    G(i+1) = 2 * g_min;
    delta = G(i+1) - G(i);
    % Re-set delta
  else
    % G(i+1) falls w/in bounds (use line 118 value)
  end
else
  % Determine current gain in decibels and check if greater than max
  T_pos = get_T(G(i), dB_lim);
  break
end
T_pos = get_T(G(i), dB_lim);
end
else
  T_pos = 0; % Controller is unstable already so there is no allowable gain margin
end

Find minimum thrust gain - whether stable or not

if (d_omega - target) <= tol % If stable - run NEG gain calc.
% Initiate the delta value for updating the alpha gain
delta = -0.5; % need to subtract gain -> -delta (in decibels)
% Initialize the bounds for the alpha gain (will be changed in loop)
g_max = 1; % Max. delay value in ms (set to 1 db to start)
g_min = 1e-4; % Min. delay value (cannot have negative delay)
% Initiate gain w/ first delta (in db)
G = Gi + delta;
if G < g_min
  G = 0.1; % G cant be 0 or neg so set to 0.1 (at least initially)
end
% Set max # of iterations
n = 51; % max # of iterations is 50 (starts @ 2)
% Initialize the variables in the for_loop
w_f = [w_f, zeros(1, n-1)];
d_omega = [d_omega, zeros(1, n-1)];
G = [G, G, zeros(1, n-2)];

% Create iteration counter
count = 0;

for i = 2:n
    count = count + 1; % Update iteration counter

    % Update the original alpha with the new gain value
    alpha = G(i) * a_i;

    % Calculate new delta_omega - alpha is the only change
    w_new = traj_project(alpha, omega, theta, DB, hyst, jet_flag, dt, a_t);
    w_f(i) = w_new(end);
    d_omega(i) = abs(w_f(i)) - abs(omega);

    % Check for convergence (where w_f is slightly greater than w_i
    % i.e.
    % where d_omega > 0.
    if abs(d_omega(i) - target) <= tol  % > 0 but <= to 0.01
        % Determine current gain in decibels and check if less than
        % T_neg = get_T(G(i), -dB_lim);
        break  % Considered converged; report current delay value %
    elseif d_omega(i) > target  % Set max delay bound
        g_max = G(i);
    else
        g_min = G(i);
    end

    % Update delta using Newtons Optimization method
    x1 = d_omega(i-1); x2 = d_omega(i);
    dx = x2 - x1;  % Change in d_omega (delta = change in gain
    - G)
    dxd = target - x2;  % Desired change to reach the target d_omega
    if abs(dx) >= tol  % Protect against dividing by 0
        delta = (delta/dx)*dxd;  % Update delta value
        delta = k * delta;  % Apply opt. gain to delta
    else
        delta = 0.01;  % Set to small but non-zero value
    end

    % Ensure loop doesnt reach for an index outside the provided
    % bounds
    if i <= (n-1)
        G(i+1) = G(i) + delta;  % Update Gain for next iteration
        if G(i+1) > g_max  % Gain is higher than max bound
            G(i+1) = k * g_max;  % Set to lower than g_max
        elseif G(i+1) < g_min  % Gain is lower than min bound
            G(i+1) = (2-k) * g_min;  % Set to higher than g_min
```
else  
  % G(i+1) falls w/in bounds (use line 118 value) 
  end 
else  
  % Determine current gain in decibels and check if less than 
  min 
  
  T_neg = get_T(G(i), -dB_lim); 
  break 
  
  T_neg = get_T(G(i), -dB_lim); 
  end 
else  
  T_neg = 0;  % Controller is unstable already so there is no 
  allowable gain margin 
  end 

T_gain = [T_neg, T_pos];
end % end of 'thrust_gain' main function

Find Gain Value in decibels (subfunction  'get_T')

function T_db = get_T(g, dB_lim)
  % Determine current gain in decibels and check if outside provided 
  limit 
  
  % Inputs:
  %  g = Current value of gain (non-dimensional) 
  %  dB_lim = Max/Min gain limit in dB 
  %
  % Output:
  %  T_db = System gain in dB 
  T = 20*log10(g);  % Pos gain value (in dB) for controller 
  
  if abs(T) >= abs(dB_lim)  % Outside provided limit 
    T_db = dB_lim;  % Set to maximum dB value if greater 
  else  % Within provided limit 
    T_db = T;  
  end
end % end of 'get_T' sub-function

Published with MATLAB® R2017a