MATLAB Simulation Verification for SPHERES

Aaron Huang
Supervisor: Zachary Funke

I. Purpose and Structure

The SPHERES program has seen many modifications over the course of two decades of operation by hundreds of undergraduate, graduate, and PhD students. As a result, testing and simulation procedures are very convoluted - to correctly make modifications to these procedures requires a detailed understanding of many parts of the testing systems. The opaqueness of these operating procedures is a considerable hindrance to further testing and development of the SPHERES program as it is a very high barrier to entry for new members of the SSL. Adding on to this barrier is a distinct lack of documentation, particularly in regards to the MATLAB simulation we use to test code before we move on to glass table testing and actual ISS test sessions.

The purpose of this log is to provide commentary and troubleshooting as I strive to achieve my current UROP objective. My role on the SPHERES team is as a UROP student working on MATLAB simulation verification for the VERTIGO project. We are trying to demonstrate autonomous SPHERES docking via SLAM algorithms using cameras on the Universal Docking Port (UDP) attachments.

II. Technical Background

During the “UDP_ISS_Science1c” test session, we observed SPHERES behavior that was inconsistent with what the MATLAB simulation told us to expect. There are two stages in the docking procedure. First, both SPHERES are told to hold position and change their attitude so that they are pointing at each other. Then, the SPHERES navigate towards each other until their UDPs can interlock and achieve docking. However, during the first step of the procedure, one of the SPHERES was observed to oscillate in a marginally stable manner and never converged on the correct attitude. This contradicts what we saw in the MATLAB simulation.

There are many possibilities for why this issue exists. The avenue that I am tasked with exploring is to evaluate how changes to the center of gravity, inertia matrix, and thruster forces of SPHERES with the UDP attachment might affect how the simulation runs. My end goal is to get quaternion plots in the simulation to match those observed on station and document how changes I made to the simulation affected its outputs. I will detail how I setup the simulation to graph quaternion data in response to changes in physical parameters and end with a short troubleshooting section of the technical issues I encountered.

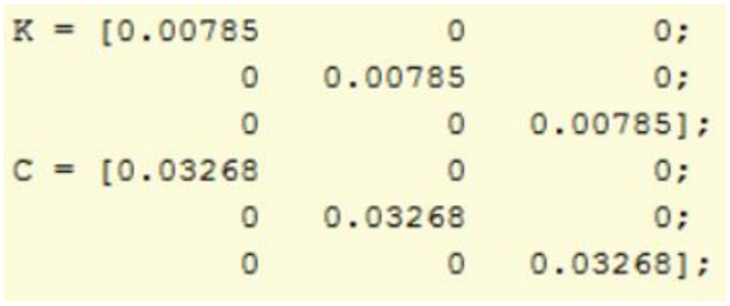
III. Simulation Modifications

To modify the desired parameters, parameters from four files had to be created or modified.

- 'generatecfg.m' located at '...\SPHERES\trunk\MatlabSim\CSPHERESCore\mfiles\@SatelliteCfg\'
- 'Docking Logic Function' under the 'Joint Dynamics Integration and Logic' block located in the Simulink model at '...\SPHERES\trunk\MatlabSim\CSPHERESCore\Simulink\DockingSimulation'
- 'configureSim.m' located at '...\SPHERES\trunk\TestProjects\Halo_Science1\MIT_P3913\Simulation'
- 'SPHERES_physical_ISS_UDP.c' located at '...\SPHERES\trunk\SpheresCore'

We decided to run the simulation under two configurations with different control gains. One set of gains was used in 'UDP Checkout' while the other set of gains was used in 'UDP Science 1c'. In 'configureSim.m', gains are included by adding the relevant file path to the 'cfg.srcs' list under 'BUILD CONFIGURATION' at the top.

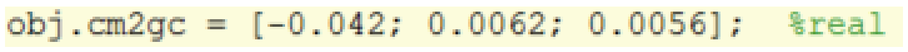
To differentiate between checkout and science gains, I created a copy of the 'SPHERES_physical_ISS_UDP.c' file and modified the appropriate gains as follows



```
K = [0.00785         0         0;
           0   0.00785         0;
           0         0   0.00785];
C = [0.03268         0         0;
           0   0.03268         0;
           0         0   0.03268];
```

This file was saved as 'SPHERES_physical_ISS_UDP_chk.c' and saved to the same file location as the original file. I then created a copy of 'configureSim.m' called 'configureSim2.m' and simply modified the 'cfg.srcs' to point to the new file with checkout gains.

The center of gravity for the SPHERES is represented as a vector in both 'generatecfg.m' and 'Docking Logic Function'. In 'generatecfg.m', we were interested in the 'obj.cm2gc' vector under the 'SatelliteCfg.MASS_UDP_ISS' case. This vector represents the distance in meters from the center of mass of SPHERES with a UDP attached to the geometric center of a SPHERES with no peripherals attached. The original cm2gc values are



```
obj.cm2gc = [-0.042; 0.0062; 0.0056];  %real
```

In ‘Docking Logic Function’, center of gravity is represented as ‘gc2cm’. This is simply distance in meters from the geometric center to the center of gravity - the inverse of ‘obj.cm2gc’ in ‘generatecfg.m’. The original gc2cm value is

```
gc2cm  = [0.0412; -0.0062; -0.0056];
```

gc2cm is represented in the ‘business cards’ of the three SPHERES that can be simulated by the simulation. To cut down on busywork, I replaced the hardcoded values of gc2cm in each SPHERES business cards with a variable gc2cm defined at the top. In this way, modifying the gc2cm variable also modifies the actual gc2cm value passed to all three SPHERES in one go.

We also changed the inertia matrix located in under the ‘SatelliteCfg.MASS_UDP_ISS’ case in ‘generatecfg.m’. Unmodified inertia is used as follows

```
cfg.dyn.inertia = [0.028856093583883000,-0.000271760426029421, 0.001061424114444470;
                  -0.000271760426029421, 0.055986602544449500, 0.000183704942897512;
                   0.001061424114444470, 0.000183704942897512, 0.058822298355749900];

cfg.dyn.inertia_multipliers = [1,1,1;
                               1,1,1;
                               1,1,1];
cfg.dyn.inertia = cfg.dyn.inertia.*cfg.dyn.inertia_multipliers;
```

We wanted to evaluate how changes to the inertia multipliers affected attitude convergence. To this effect, I was directed to change the diagonal inertia elements between 50% and 150% of their original values while changing cross-elements anywhere between up two orders of magnitude and down one order of magnitude. The row and column axes both correspond to ‘X’, ‘Y’, and ‘Z’ axes. For instance, the ‘XZ’ gain multiplier will refer to the first row, third column while the ‘YX’ gain multiplier refers to the second row, first column.

Lastly, we changed the thruster strengths located in ‘generatecfg.m’ directly under the ‘SatelliteCfg.MASS_UDP_ISS’ case. The original thruster strength is

```
%% Thruster Strength
cfg.dyn.thrusterForce = 0.098;
```

IV. Plots and Results

We first looked at the lance hole graphics to determine the effect of varying thruster forces on convergence.

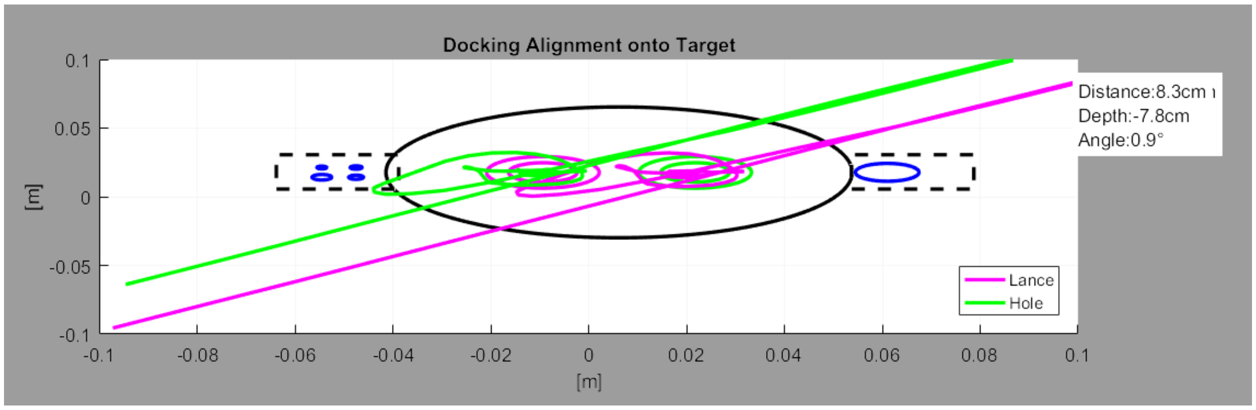


Figure 1: Lance-hole graphic at thruster strength = 0.07 N

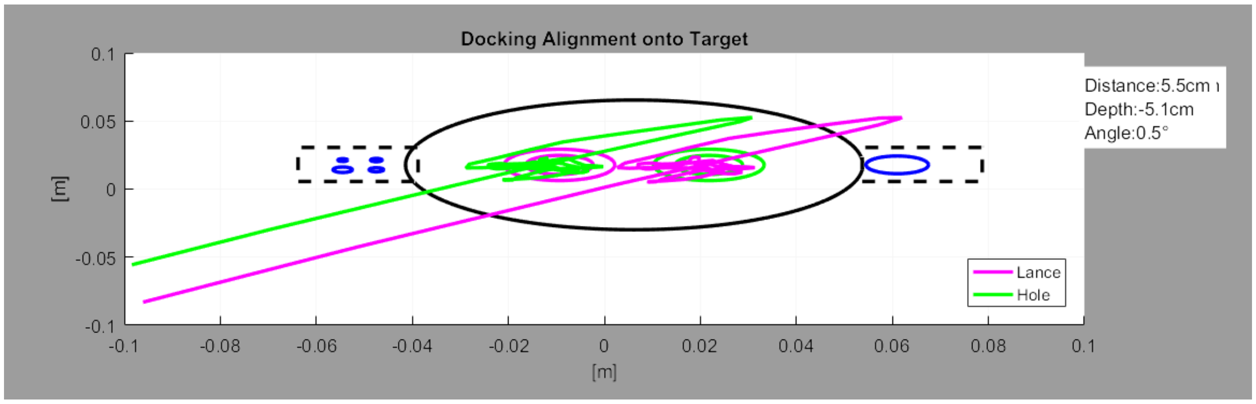


Figure 2: Lance-hole graphic at 'vanilla' thruster strength = 0.098 N

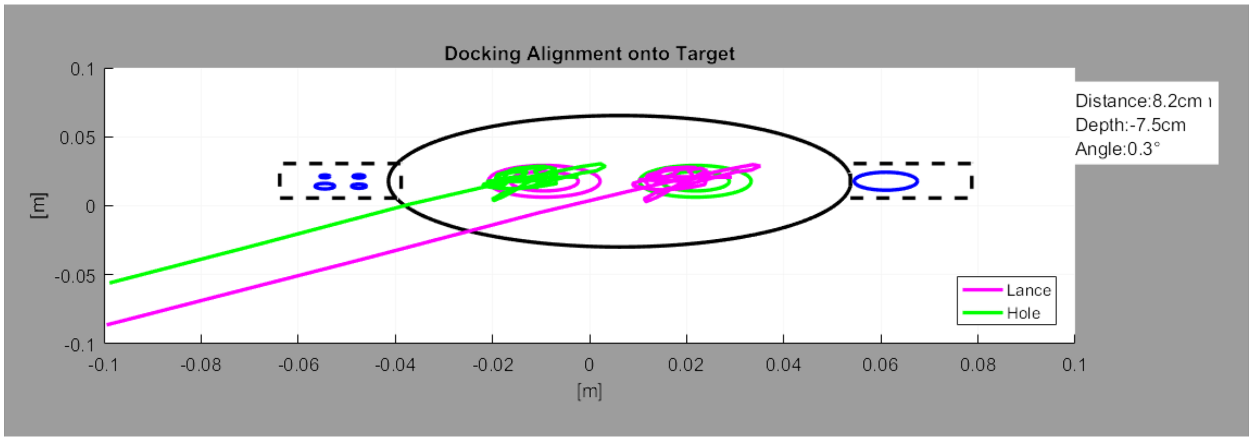


Figure 3: Lance-hold graphic at thruster strength = 0.13 N

These graphics indicate that lower thruster forces tend to increase the amount of time it takes for the SPHERES to converge on each other. Attitude convergence at the low thruster strength exhibits larger deviation, with a slight hint of oscillation to the left of both targets. On the other hand, attitude convergence at high thruster strength shows very quick attitude convergence with minimal deviation.

We then wanted to determine how changes to inertia multipliers affected quaternion data between UDP Checkout and UDP Science simulations. I wrote a simple script to automate the process of running and graphing the simulations. I created one instance of 'configureSim1.m' that used science gains and one instance of 'configureSim2.m' that used checkout gains and ran each simulation for 100,000 milliseconds. Then, each configuration's quaternion data was retrieved at 'cfg.data{1,1}.BackTel.StdState(8:10,:)'. Rows 8 to 10 represent x, y, and z quaternions respectively. An example plot with 'vanilla' values for all parameters used in both runs shown is below. **Science gains are on left, Checkout gains are on right. Vanilla values correspond to the original values depicted under 'Simulation Modifications'.**

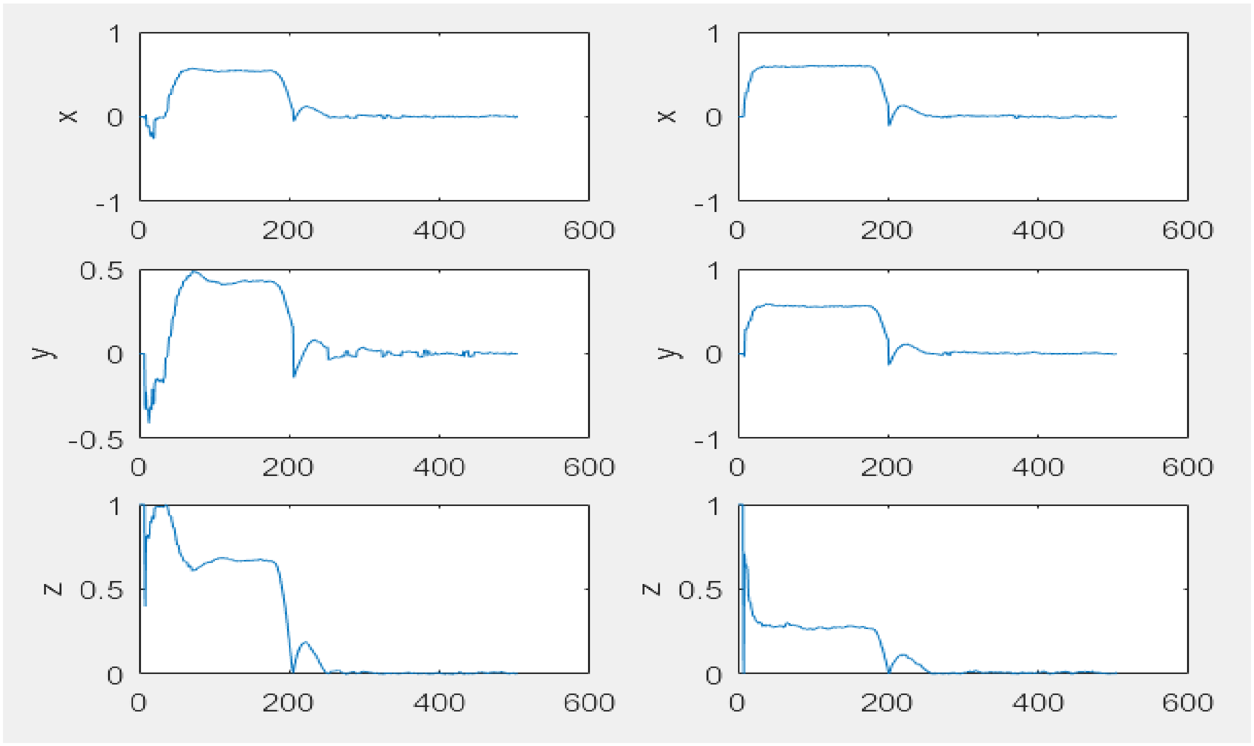


Figure 4: Quaternion data for checkout and science simulation runs using vanilla parameters

I decided to test changes to diagonal elements of the inertia matrix as there are far less possible combinations. **I also decided to use thruster strengths = 0.07 N for all runs as that seemed to be most conducive to marginal stability.**

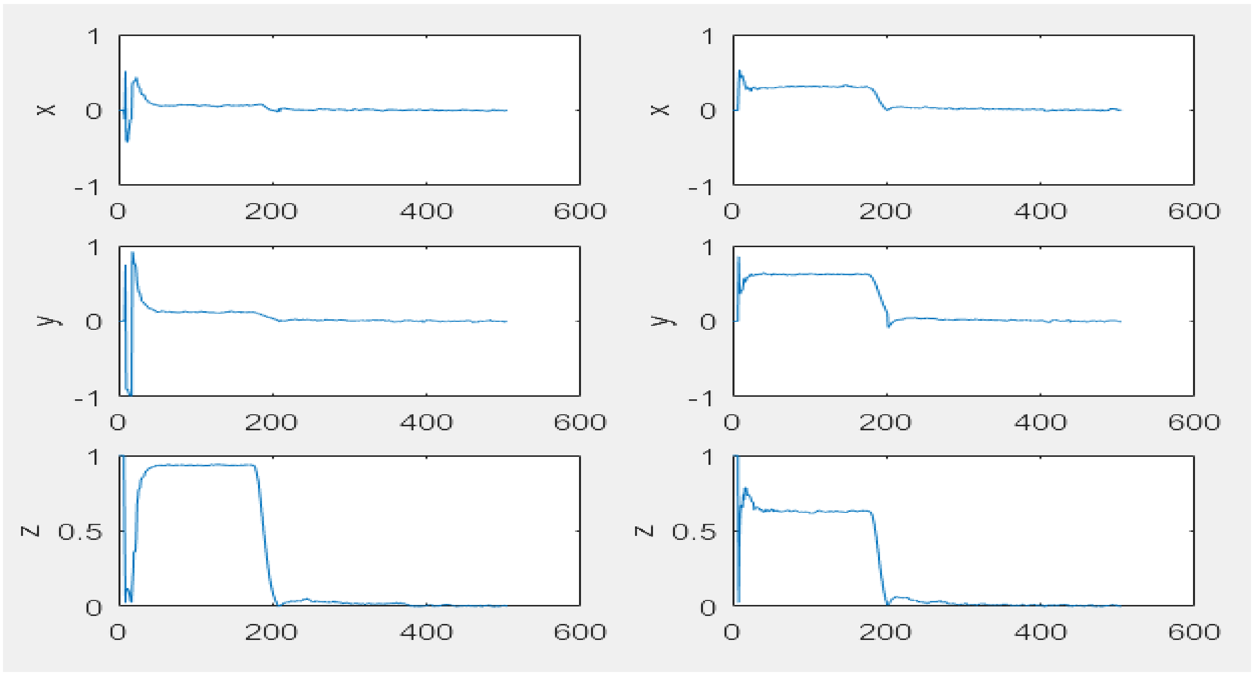


Figure 6: 0.5x diagonal inertial elements

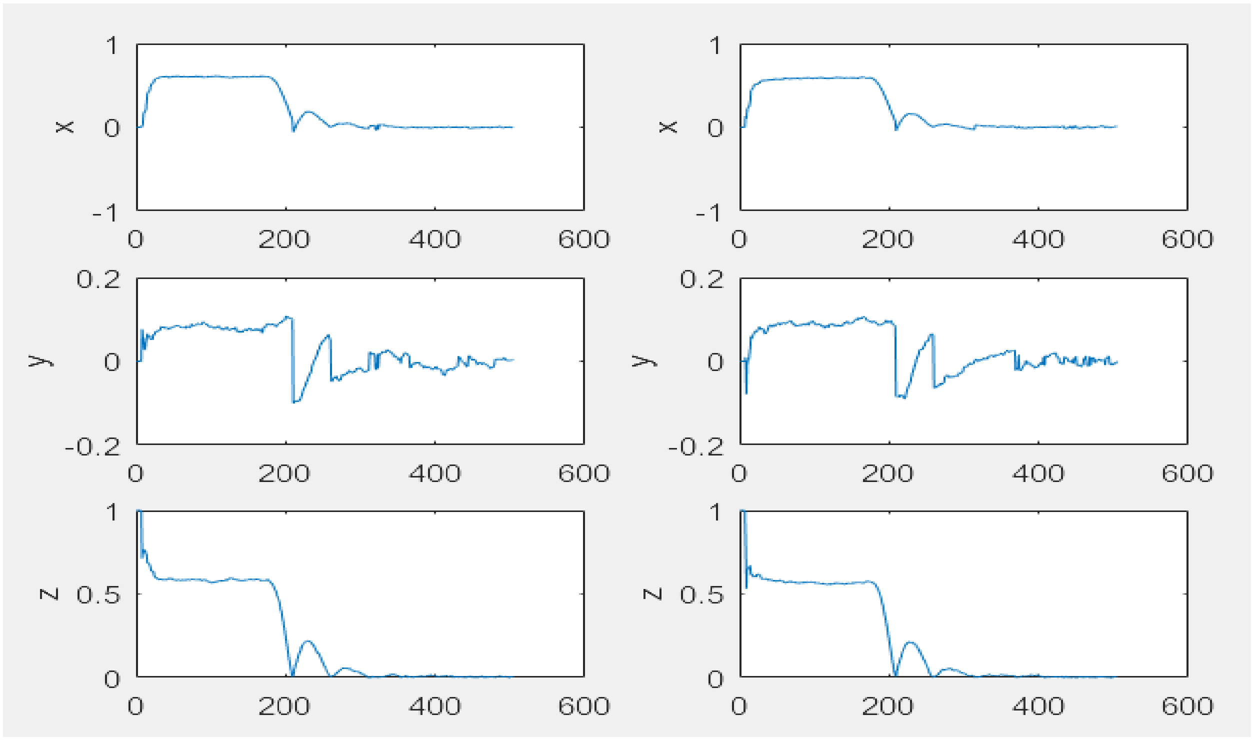
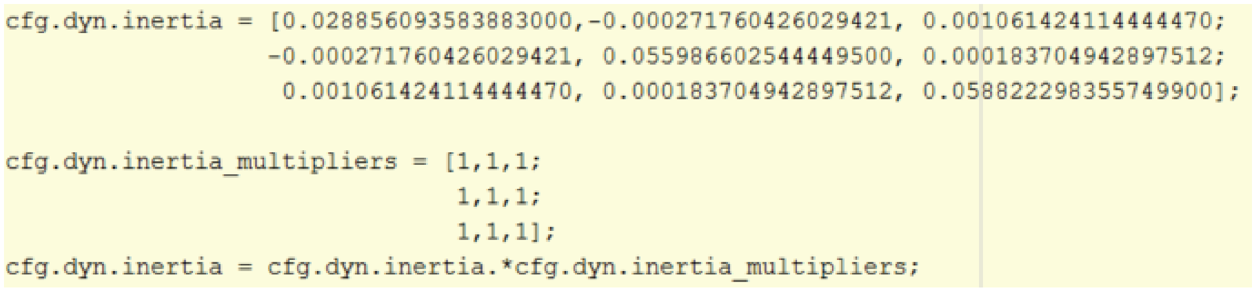


Figure 5: Quaternion data with 1.5x diagonal inertia elements

Diagonal inertia elements seemed to delay convergence when their values are increased.

Next, I tested how pairs of cross-element inertias affected convergence. As a quick refresher, each inertia is indexed ‘RowColumn’. For instance, ‘XZ’ refers to the first row and the third column. ‘ZY’ refers to the third row, second column.



```
cfg.dyn.inertia = [0.028856093583883000,-0.000271760426029421, 0.001061424114444470;
                  -0.000271760426029421, 0.055986602544449500, 0.000183704942897512;
                   0.001061424114444470, 0.000183704942897512, 0.058822298355749900];

cfg.dyn.inertia_multipliers = [1,1,1;
                               1,1,1;
                               1,1,1];
cfg.dyn.inertia = cfg.dyn.inertia.*cfg.dyn.inertia_multipliers;
```

Figure 6: Inertia setup in ‘generatecfg.m’ under case SatelliteCfg.MASS_UDP_ISS

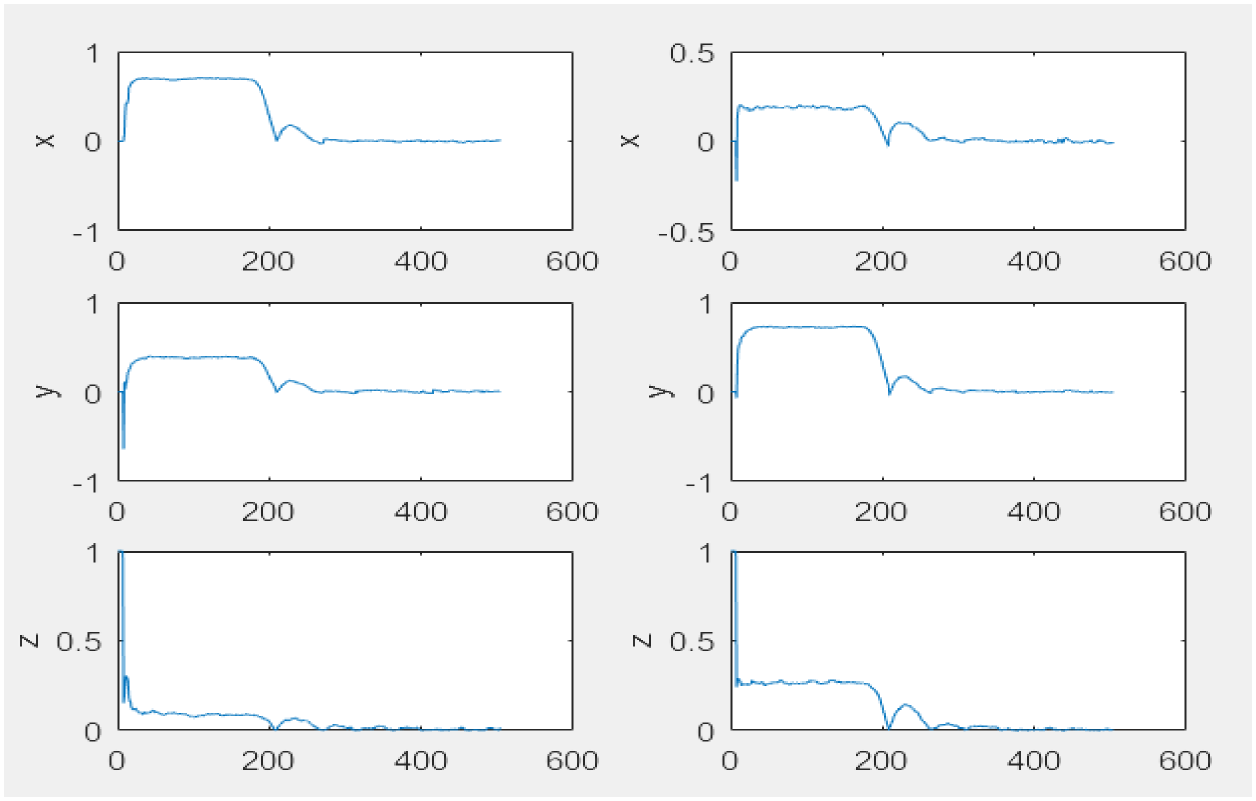


Figure 7: 10x XY and YX cross-element inertias

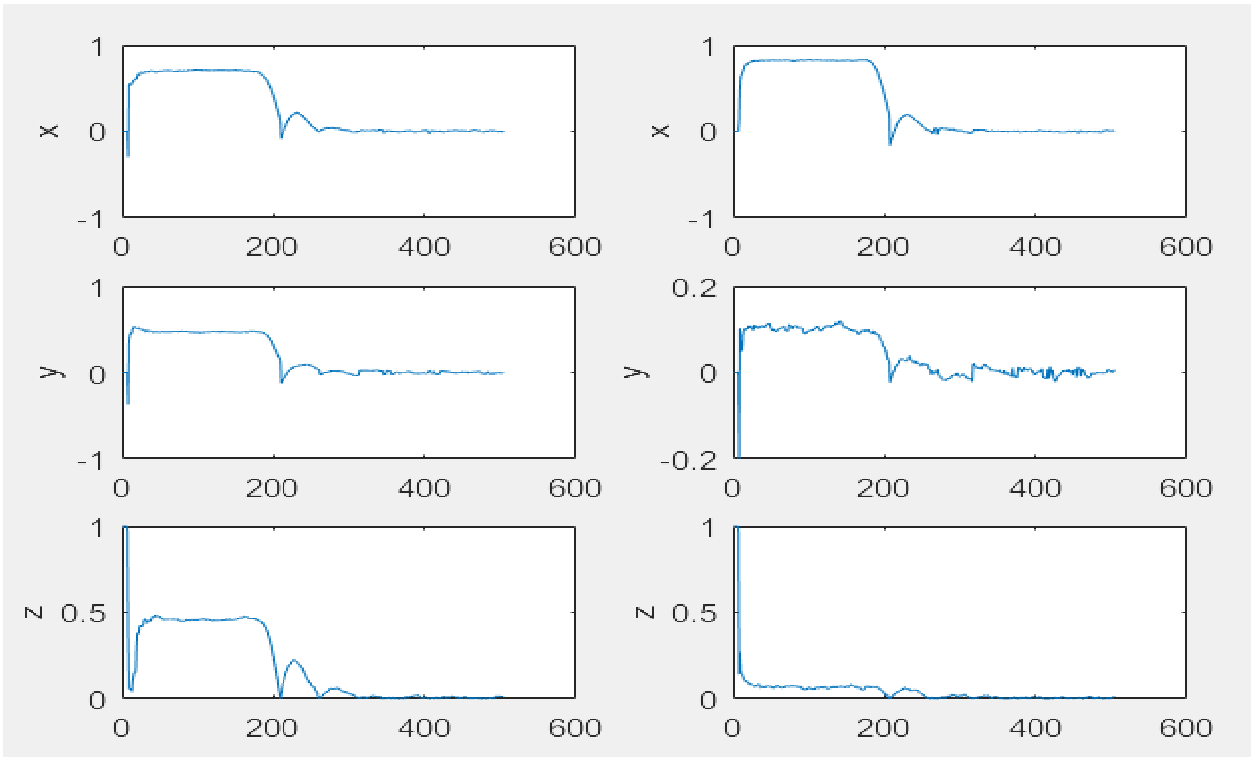


Figure 8: 10x YZ and ZY cross-element inertias

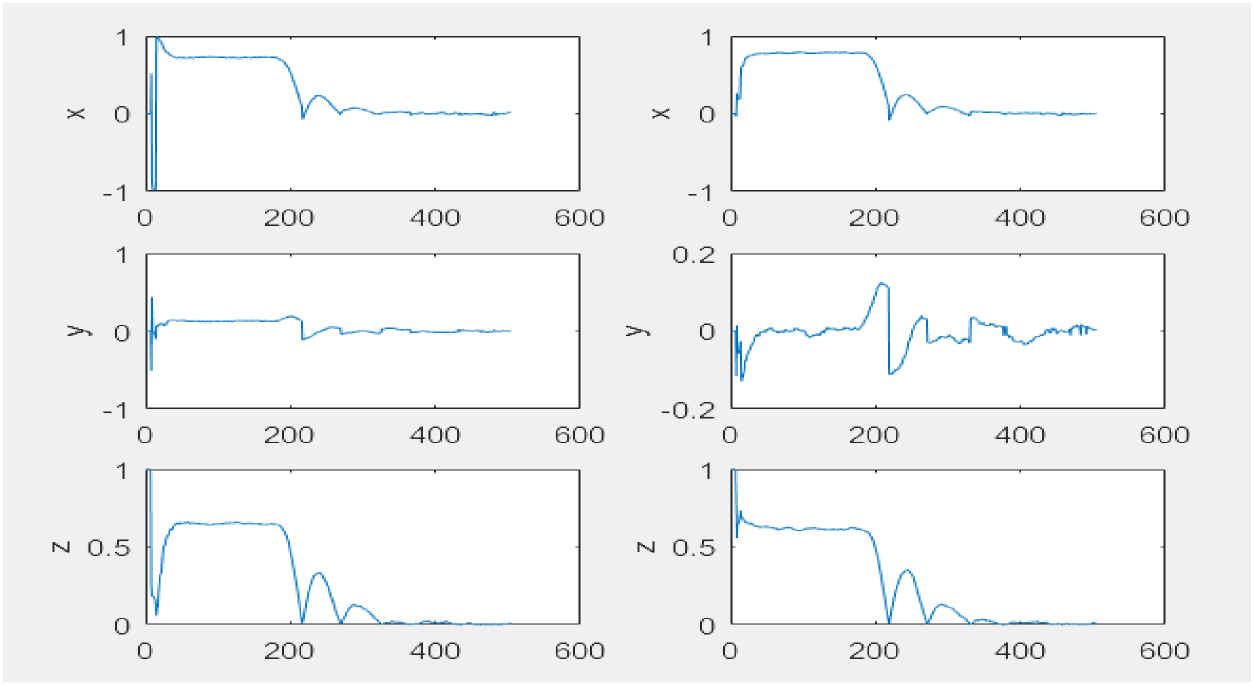


Figure 9: 10x XZ and ZX cross-element inertias

Increasing the XZ and ZX cross-element inertias by an order of magnitude seems to cause some oscillation in the checkout simulation.

I proceeded to increase all of the cross-element inertias by one and two orders of magnitude.

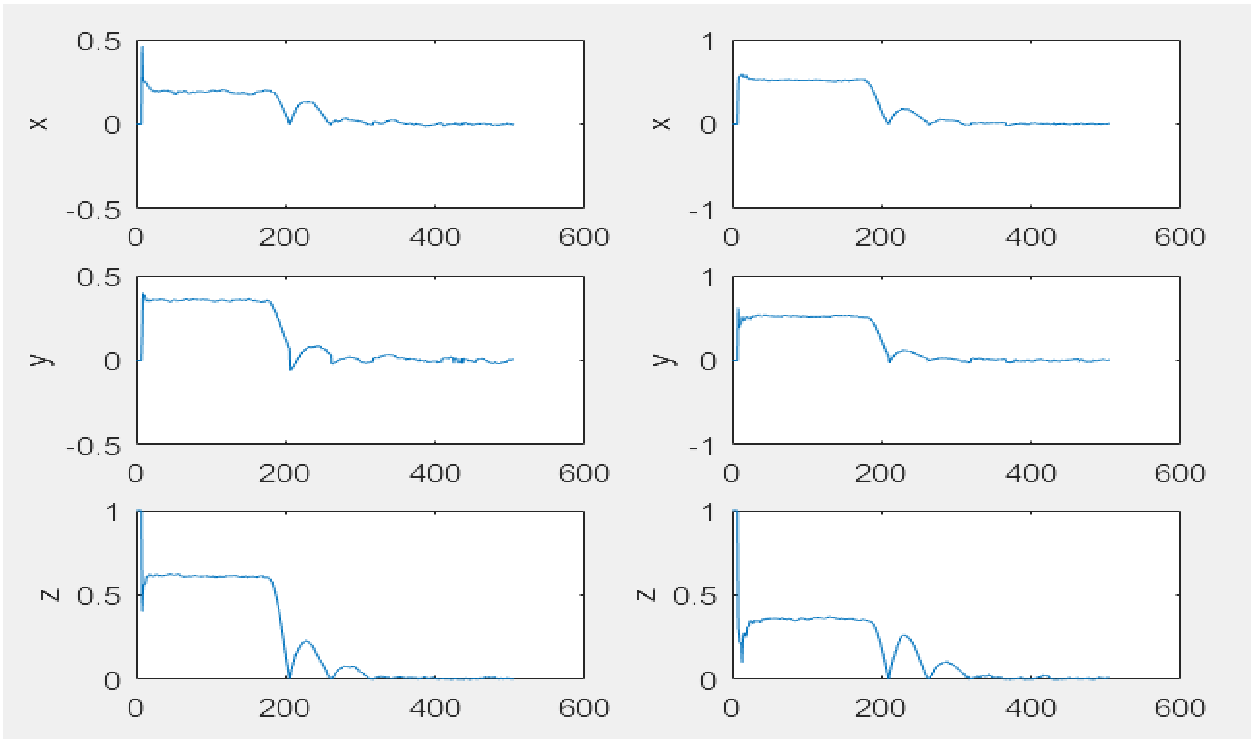


Figure 10: 10x all cross-element inertias

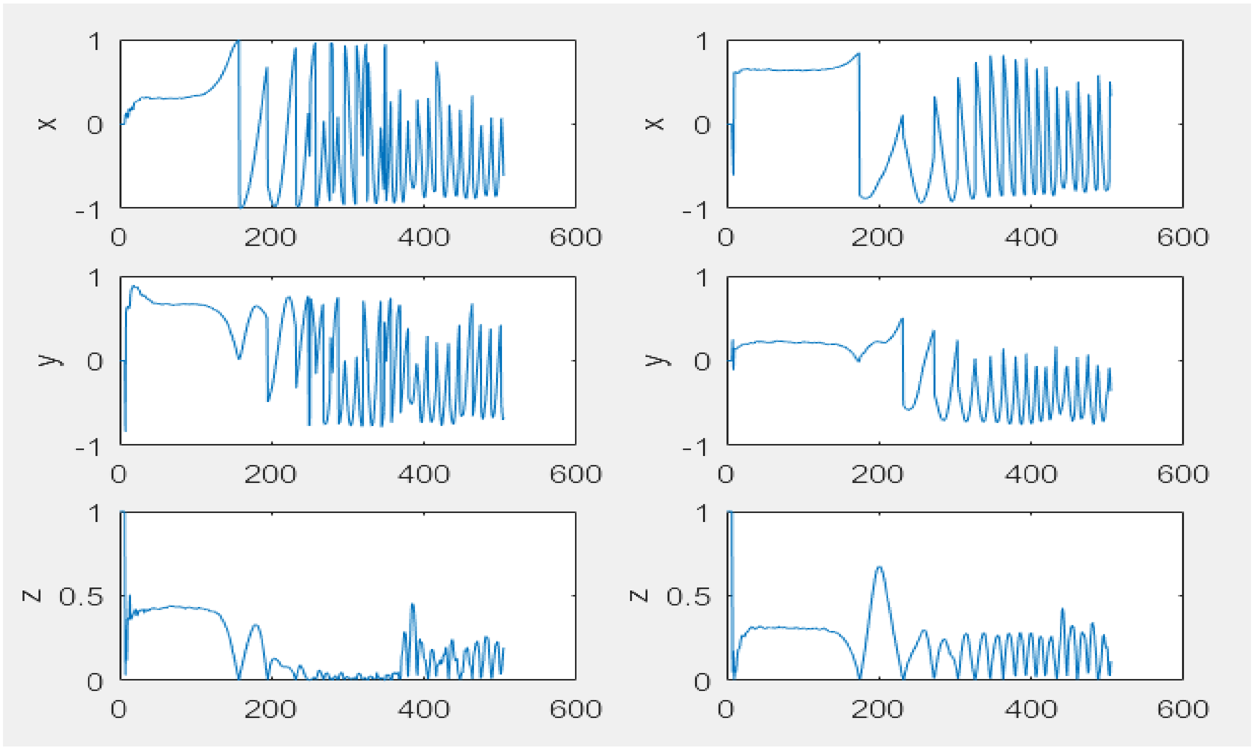


Figure 11: 100x all cross-element inertias

Figure 11 exhibits the type of oscillations we were looking for. Increasing every cross-element inertia by a factor of 100 puts the simulation in a marginally stable state. I then attempted to find the 'boundary conditions' at which the system transitions from stable to marginally stable. I focused on the XZ and ZX inertia multipliers as previous runs seem to suggest the simulation is most sensitive to those inertias. **I also returned thruster forces to their normal value of 0.098 N to see if marginal stability still existed.**

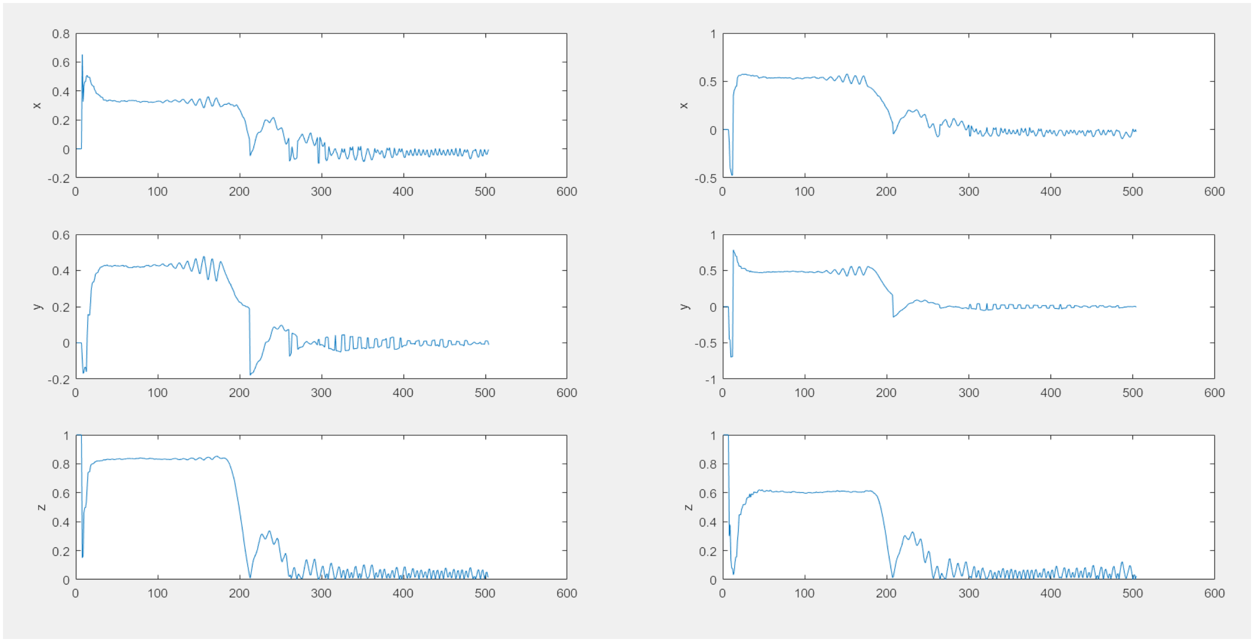


Figure 12: 30x XZ and ZX cross-element inertias, thruster strength = 0.098N

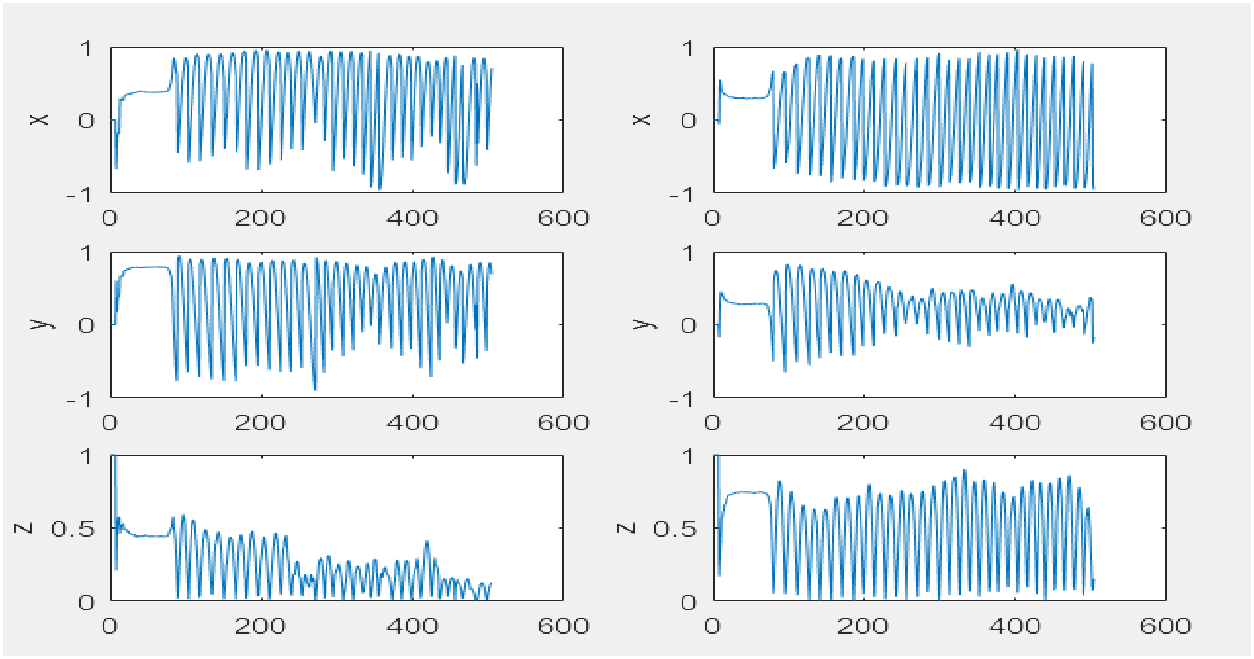


Figure 13: 40x XZ and ZX cross-element inertias

It appears that the system transitions from fully stable to marginally stable somewhere in the vicinity of 30-40x XZ and ZX cross-element inertias. Once the system fully transitions to marginal stability, the oscillations become quite large relative to any oscillations found in a fully stable system. I have also included a couple of interesting graphs below that I found while testing.

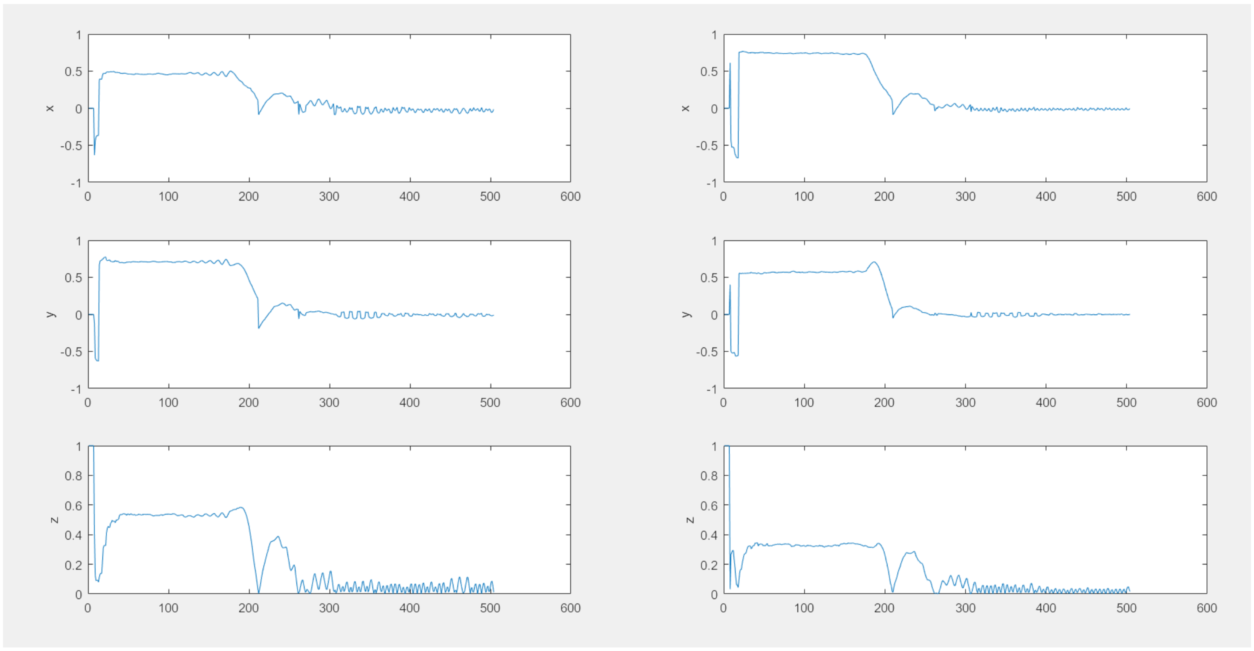


Figure 14: 29x XZ and ZX, XY and YX cross-element inertias

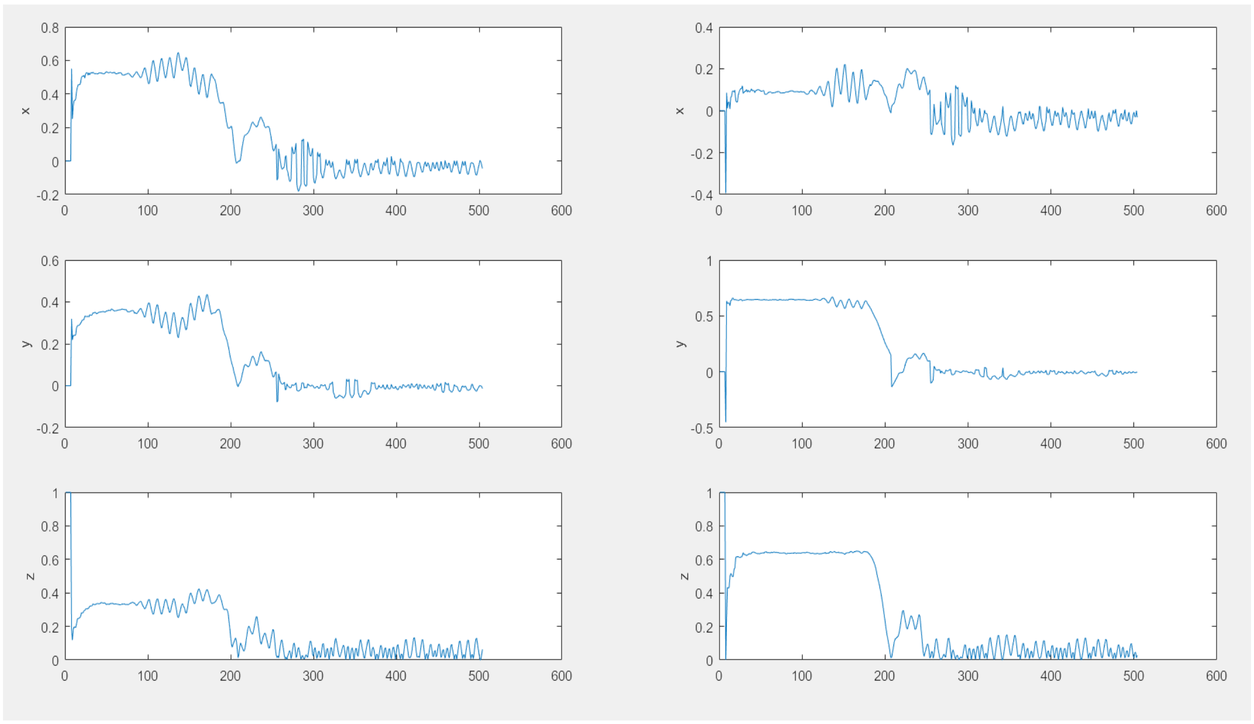


Figure 15: 30x XZ and ZX, XY and YX cross-element inertias

The above graphs show a relatively significant decrease in stability as XZ and ZX, XY and YX cross-element inertias are increased 30x from 29x. This seems to be a boundary condition.

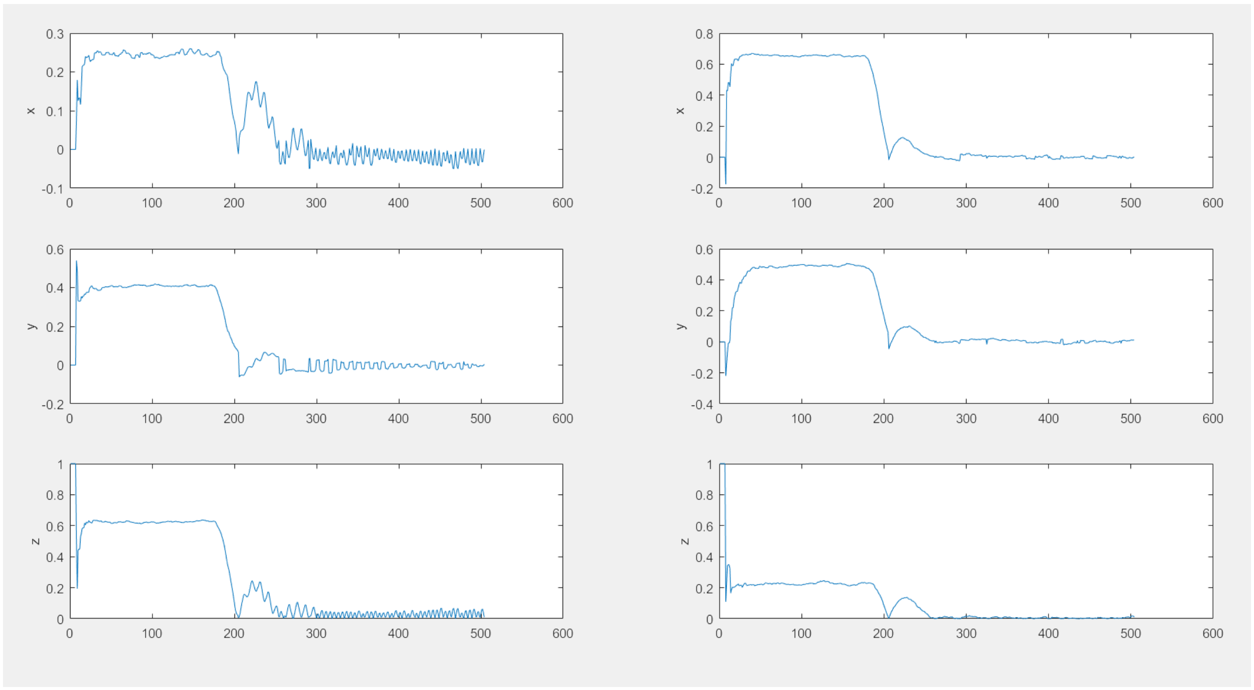


Figure 16: 20x XZ and ZX, YZ and ZY cross-element inertias

This plot is of particular interest since it seems to depict the gains used on UDP Science as less stable than those used on UDP Checkout. This is consistent with what we observed on station and is worth investigating further.

In conclusion, changing the inertia matrix as well as the thruster strength seemed to have the greatest impact on simulation convergence. The simulation seemed particularly sensitive to changes in the XZ and ZX cross-element inertias, with maximal effect seen when combined with changes in other cross-element inertias. **The boundary condition for transition into a marginally stable system occurs when cross-inertias are increase between 30-40x**. Further verification of the simulation would do well to look at combining increases in cross-element inertia values, decreasing thruster strength, and exploration of how center of gravity changes factor into simulation convergence.

V. Troubleshooting

<u>Specs</u>
MATLAB version R2016a
Microsoft Visual Studio 2010
Windows 10
Intel amdx86_64 processor

Issue: 'rebuildAll' is not a defined function or variable. 'configureSim()' also throws a similar error for 'globalvarSPHERES'

- The working theory is that the 'startup.m' file that runs upon MATLAB launch is somehow functioning incorrectly. This is where environment initialization and path indication is done.
- "...SPHERES\trunk\TestProjects\UDP_Science\UDP_ISS_Science1\Simulation' is the current MATLAB directory. This directory matches the general path "SPHERESCORE_DIR" takes. 'startup.m' was confirmed to be configured correctly.

Solution: 'startup.m' was actually named 'setup.m' - absolutely needs to be 'startup.m'

Observation: When I was making the above change, I ran the simulation while having an explicit error in the Docking Logic Function - gc2cm was never defined. The sim still ran but converged far more slowly than it usually does

Issue: While attempting to graph quaternions from both configureSim versions, simulation threw was unable to locate the data structure inside each configureSim object

- Science gains were included in a 'configureSim1.m' class while checkout gains were included in a 'configureSim2.m' class.
- configureSim1 and configureSim2 objects were named 'cfg1' and 'cfg2' respectively
- ans structures also did not include a data structure.

Solution: It turns out that the configureSim objects **must be named cfg**. Otherwise, the configureSim object will not contain data from the simulation.

Issue: Data structure was still not found in cfg objects even if they are named cfg
Solution: configureSim **must have animateOn = 1**. This has the unfortunate consequence of making test runs quite a bit longer as animating the simulation takes some time. However, this is necessary to plot the desired quaternion data.