NOTE: This Read Me is meant to orient the user as to which files are connected and necessary to run the various CSACS simulations. I tried to comment the code as well as possible, but please don't hesitate to contact me if you need more assistance. All relevant equations and derivations are provided in the thesis, "Advancements in the Design and Development of CubeSat Attitude Determination and Control Testing at the Virginia Tech Space Systems Simulation Laboratory.'' All m-files, functions, and Simulink models developed with MATLAB R2017a.

Anthony Wolosik

awolosik@vt.edu

(412) 225-9076

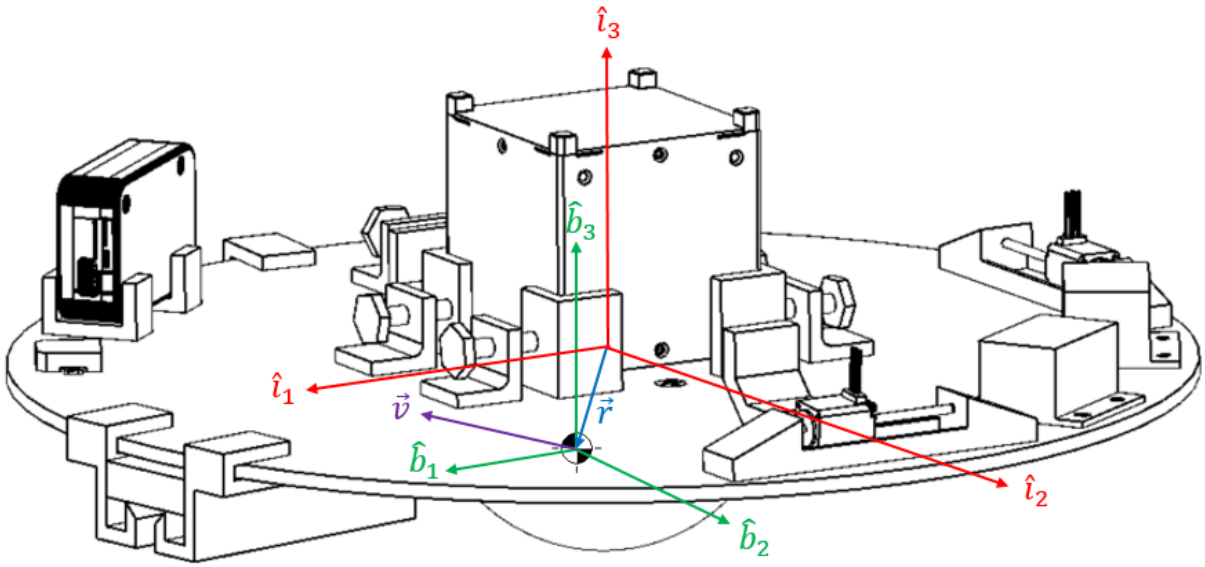

Figure 1: The CSACS System Model Reference Frames. The assembled CSACS with a 1U CubeSat as the demonstration payload is shown. The origins of the inertial frame (red) and the body-fixed frame (green) are located at the air bearing's CR and the CSACS CM, respectively. The r vector (blue) is the distance from the CR to the CM, and the v vector (purple) is the velocity of the CM. Note that the linear stepper motors (MMUs) are individually aligned with the $\hat{b}_1$- and $\hat{b}_2$-axes, which correspond to the body-fixed frame *x*- and *y*-axes, respectively.

Software Contents

1. Automatic Balancing Algorithm

`CSACS_Sim.m` – This is the main program that simulates the nonlinear dynamics of the CSACS, given initial conditions for the state vector. All other functions in the "Balancing Algorithm" folder are used by this m-file. Going through the code, you will see that the program first calls `user_inp.m`, which is an initialization function that will be addressed later. The user can then specify the number of test iterations (shouldn't need to set the variable `runs` to an integer larger than 4, as long as the CSACS is adequately balanced as defined by the theory, *i.e.*, the magnitude of the *x*- and *y*- components of the CM vector must be less than 0.305 mm). Next, the user can define the initial time `t0` along with the data acquisition time step `dt`. Obviously a smaller the time step yields a higher resolution solution, but at higher computational cost. The CSACS physical properties are determined from the most recent CAD model. Note that the current model needs to be developed at a much higher quality and detail from which it currently stands. Initial MMU displacements should be set to zero in simulation. Physically, they should be centered on their respective threaded rods, such that maximum distance in the positive and negative direction can be achieved. The variable `rm_max` defines the maximum linear distance each MMU can move, and establishes the maximum correctable CM offset error check. Lastly, the user can define which type of ode solver he or she would like to use. The variable `caseNum` lets you choose between Matlab's ode45, Euler's explicit method, two-stage Runge-Kutta (Heun's method), or four-stage Runge-Kutta. Matlab's ode45 will yield the highest resolution results; however, a separate analysis showed that the RK4 algorithm generates almost identical results at a fraction of the computational cost. When converting this code to an on-board solution, RK4 should be the implemented ode solver. For this reason, `caseNum = 'Case 4'` is the default solver, where `'Case 4' = Four-Stage Runge-Kutta (RK4)`. And that's it, run the code and produce some results!

`csacseom.m` – This function is called by `CSACS_Sim.m` and calculates the time derivatives of the state vector, using the full CSACS equations of motion. Just be sure to verify the CSACS physical properties match those defined in `CSACS_Sim.m`. The aerodynamic damping coefficient vector is currently defined from the literature (Greenwood, 1988). For anyone continuing this work, an interesting research problem would be working to develop a higher fidelity aerodynamic torque model, specifically for the CSACS.

`r_est.m` – This function is called by `CSACS_Sim.m` and estimates the vector to the CM of the CSACS, from the CR of the air bearing, using the dynamic data. Again, be sure to verify the CSACS physical properties match those defined elsewhere.

`rm_est.m` – This function is called by `CSACS_Sim.m` and calculates the distance to move the MMUs in order to remove the CM offset.

`user_inp.m` – As previously mentioned, this function is called by `CSACS_Sim.m` and initializes certain variables that are a part of the state vector. Total simulation time is defined by `tf` [seconds] (default simulation time `tf = 60`). The user can define initial angular positions [radians], but remember that the CSACS is physically limited to $\pm 15^{\circ}$ in pitch and roll. Initial angular velocities are input in [rad/sec]. Lastly, the most important initial conditions are the *x*-, *y*-, and *z*- components of the CM vector. These values are obtained directly from the CAD model. Remember, the CSACS must be manually balanced to within the maximum correctable CM offset of 0.305 mm. Thus, if $\sqrt{r_x^2 + r_y^2} > 0.305$ mm, an error message will be triggered in `CSACS_Sim.m`.

- `animate.m` – After `CSACS_Sim.m` generates the simulation results, `animate.m` can be called to generate a 3D animation display of the CSACS dynamics. The script is self-contained, so all the user needs to do is run the code. The only requirement is that `CSACS_Sim.m` must be successfully executed in order for `animate.m` to extract the trajectory data. The code parses the CSACS state vector time-history data and generates "timeseries" data-types for attitude and simulation length. Since this script is used for both auto-balance visualization and active control visualization, reaction wheel speeds need to be generated for order for the code to run properly. In the case of the auto-balancing procedure, reaction wheel speeds are null. Note that the state vector that `animate.m` extracts data from represents the last iteration executed by `CSACS_Sim.m`. Thus, if the user wishes to animate the dynamics of an earlier iteration of the algorithm, he or she should specify the desired iteration accordingly, which is defined by the `runs` variable in `CSACS_Sim.m`. A screenshot of the animation is shown in Fig. 2, where the dynamics of the CSACS are simulated after the balancing algorithm has relocated the MMUs one time, *i.e.*, `runs = 1`. Lastly, note that the functions `R1.m`, `R2.m`, and `R3.m` are used by `animate.m` to rotate the various geometries about the 1, 2, and 3 axes, respectively, thus producing dynamic, 3D spatial rotation animations.

- `animate.m` – Sub-script called by `animate.m` that generates the CSACS geometry for animation.

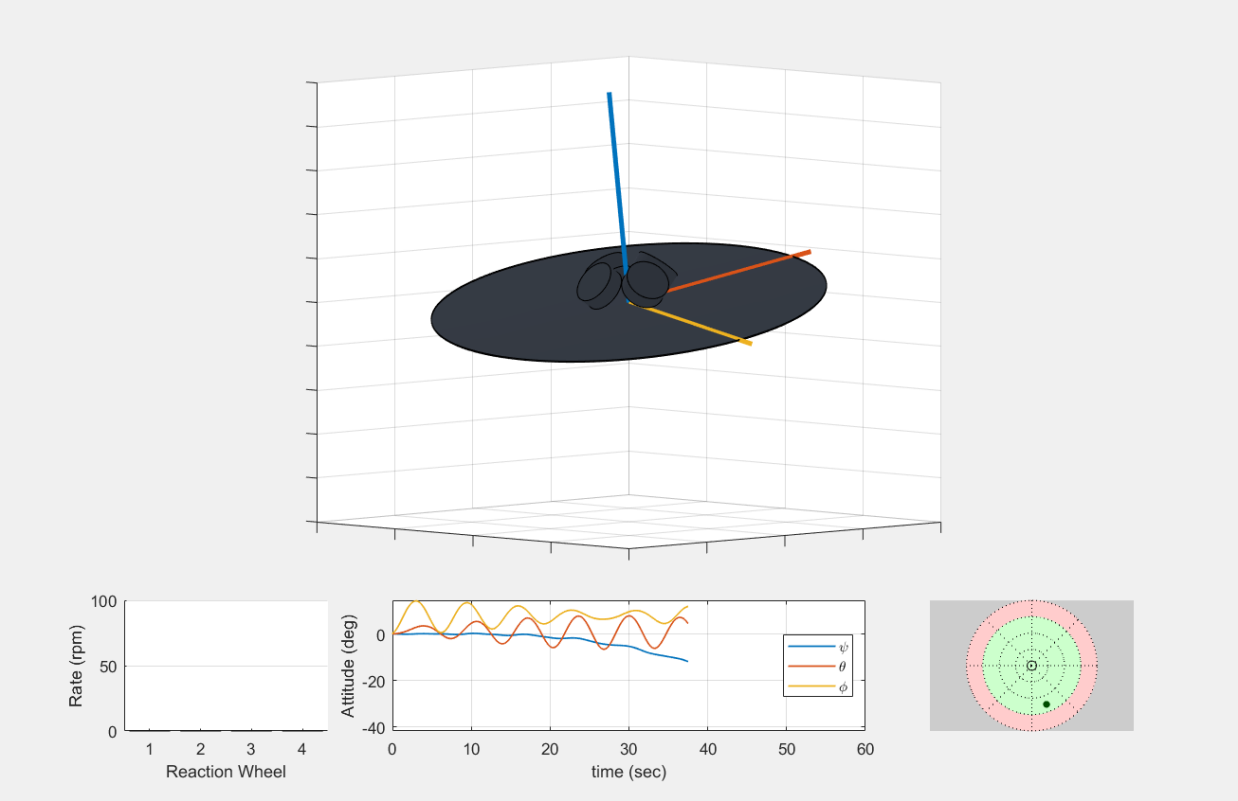

Figure 2: 3D Animated Display of the CSACS Auto-Balance Routine. The animation window shown is a result of successfully executing `CSACS_Sim.m`, thus generating the CSACS trajectory time-histories, and then running the 3D animation script `animate.m`. This screenshot was taken at approximately 38 seconds into the animation, where only 1 iteration of the balancing algorithm, *i.e.*, one MMU relocation, was performed.

A description of each component of the animation window follows. The 3D motion of the CSACS is displayed in the top figure, where the body-fixed frame *x*-, *y*-, and *z*-axes (*i.e.*, $\hat{b}_1$-, $\hat{b}_2$-, and $\hat{b}_3$-axes) are colored yellow, orange, and blue, respectively. The reaction wheel rates in the bottom-left figure are clearly not active during the auto-balance procedure. Attitude time-histories are displayed in the bottom-center where ϕ, θ, and ψ correspond to roll (about the $\hat{b}_1$-axis), pitch (about the $\hat{b}_2$-axis), and yaw (about the $\hat{b}_3$-axis), respectively. Lastly, a tilt limit plot is displayed in the bottom-right figure. In this plot, the circular radius is proportional to the angle between the $\hat{b}_3$-axis and the inertial frame *z*-axis (*i.e.*, $\hat{\imath}_3$-axis), which corresponds to tilt. The location of the dot-indicator (measured in degrees) is simply the tilt angle of the $\hat{b}_1$-$\hat{b}_2$ plane. In other words, think of this plot as a top-down view of the CSACS platform, where the dot-indicator shows the current tilt angle of the CSACS platform during simulation. The dashed-circles increment by 5° starting at the origin, where any tilt angle greater than the platform's $\pm 15°$ maximum tilt limit becomes indicated in red, signifying that the platform made contact with the air bearing's pedestal and re-balancing must ensue.

2. Pyramidal Reaction Wheel Array, Quaternion Feedback (PD) Controller

`RWA_Controller.slx` – Simulink model of the full quaternion feedback (PD) controller for the pyramidal RWA. Excluding the post-processing figure generation script `figures.m` and the 3D animation script `animate.m` (both will be addressed later), all of the remaining functions in the "Quaternion Feedback Controller" folder are embedded and automatically executed upon running the Simulink simulation. A "block-by-block" description of the model follows.

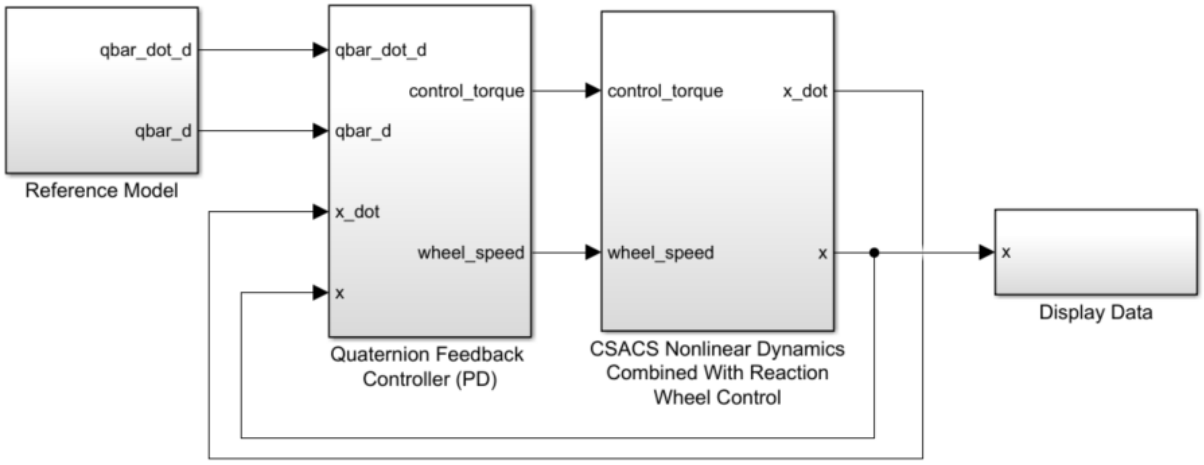

Figure 3: Top Level Simulink Model

When viewing the top level of the controller model, note that the user can define the simulation length (currently defined as `ts+60`, which is the second-order system settling time plus 60 seconds). Also, by selecting *View* > *Property Inspector*, or pressing Ctrl+Shift+I, reveals that `RWA_Controller.slx` uses the Model Callback InitFcn* to execute the initialization script, `Init_Sim.m`.

`Init_Sim.m` – This file initializes various simulation parameters and sets up the Simulink reaction wheel model for nonlinear, quaternion feedback control. After the auto-balance routine has successfully determined where to positon the MMUs, the CSACS physical properties are determined from the updated CAD model. Reaction wheel array properties are defined from literature and the Maxon motor datasheet (Dannemeyer and Tibbs). The pyramidal distribution matrix is defined from the theory, where each reaction wheel inclination angle is identically $\beta = 45^{\circ}$. All wheels are

weighted equally, thus $K \in \mathbb{R}^{4\times4}$ is defined by $K = \text{diag}\{1,\ 1,\ 1,\ 1\}$ and $W \in \mathbb{R}^{4\times4}$ is defined by $W = \text{diag}\{1,\ 1,\ 1,\ 1\}$. The user has the option to disable any one reaction wheel for simulation; however, disabling more than one wheel will create an under-actuated system (not good). As explained in my thesis, the user defines the second-order system elements, which influence both the reference trajectory and controller gains. The reference trajectory is set to be critically damped, *i.e.*, $\zeta = 1$, with an arbitrary settling time $t_s = 60$ and maximum overshoot criteria $M_p = 2\%$. Now, the natural frequency ω_n can be established. Controller gains are defined according to the thesis, but they can be tuned differently if the user wishes to do so. Initial attitude [degrees] and body rates [rad/sec] are user-defined. The final user input is the desired attitude [degrees]. Remember, gravitational torques that develop when the CM is not aligned with the CR cause the reaction wheels to immediately saturate. Thus, only adjust the variable `psi_d`. For anyone continuing this work, implementing three-axis control would be a great research project. Many ways to approach this, but definitely not an easy problem. Some ideas include: active feedback control of *x*-, *y*-, and *z*-axis MMUs which consistently align the CSACS CM with the air bearing's CR, increasing reaction wheel size (but at what cost to overall system mass?), or separating the RWA and individually relocating each reaction wheel further away from the CR to generate a larger moment-arm.

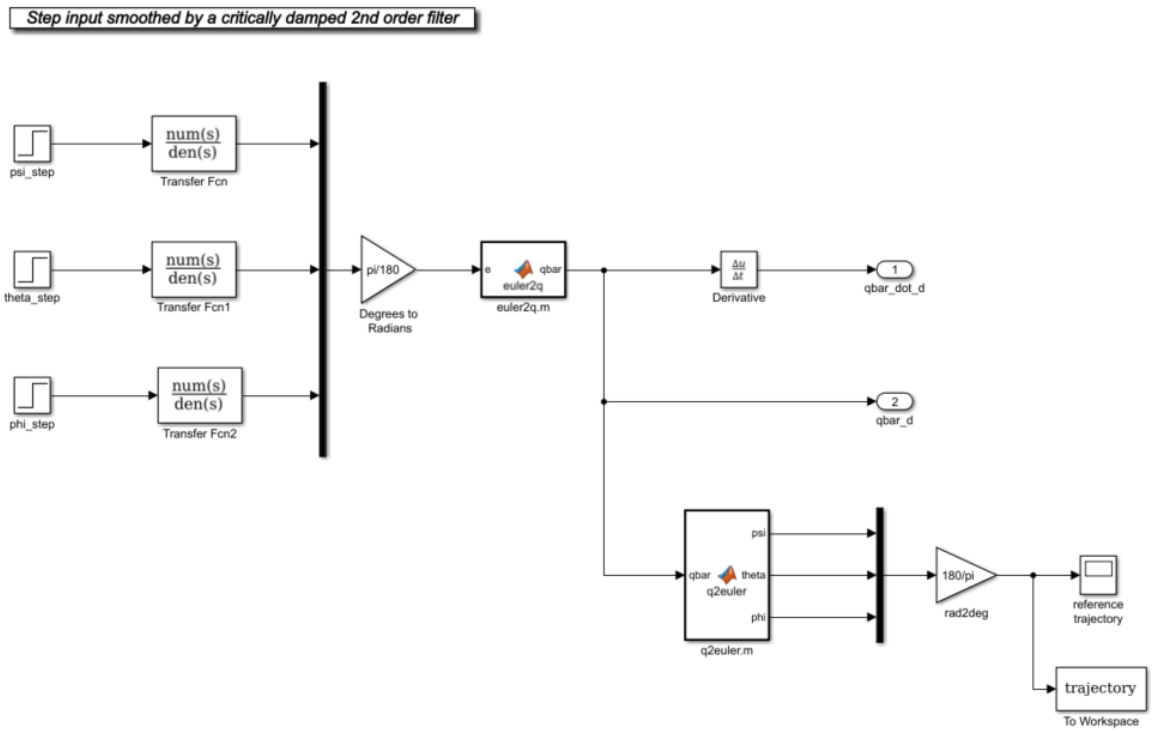

Figure 4: Reference Model Block. The reference model takes unit-step inputs for each attitude orientation and filters them to provide a smooth signal for the controller to follow. Unit-step inputs are defined by initial and desired attitudes. Filters are all second-order systems specifically defined in my thesis.

`euler2q.m` – Function that transforms Euler angles to quaternion parameters via a 3-2-1 sequence, where ψ is the first rotation, θ is the second rotation, and ϕ is the third rotation.

`q2euler.m` – Function that computes the Euler angles from the unit quaternion $\bar{\boldsymbol{q}} = [q_0 \ q_1 \ q_2 \ q_3]^T$ via a 3-2-1 sequence, where ψ is the first rotation, θ is the second rotation, and ϕ is the third rotation.

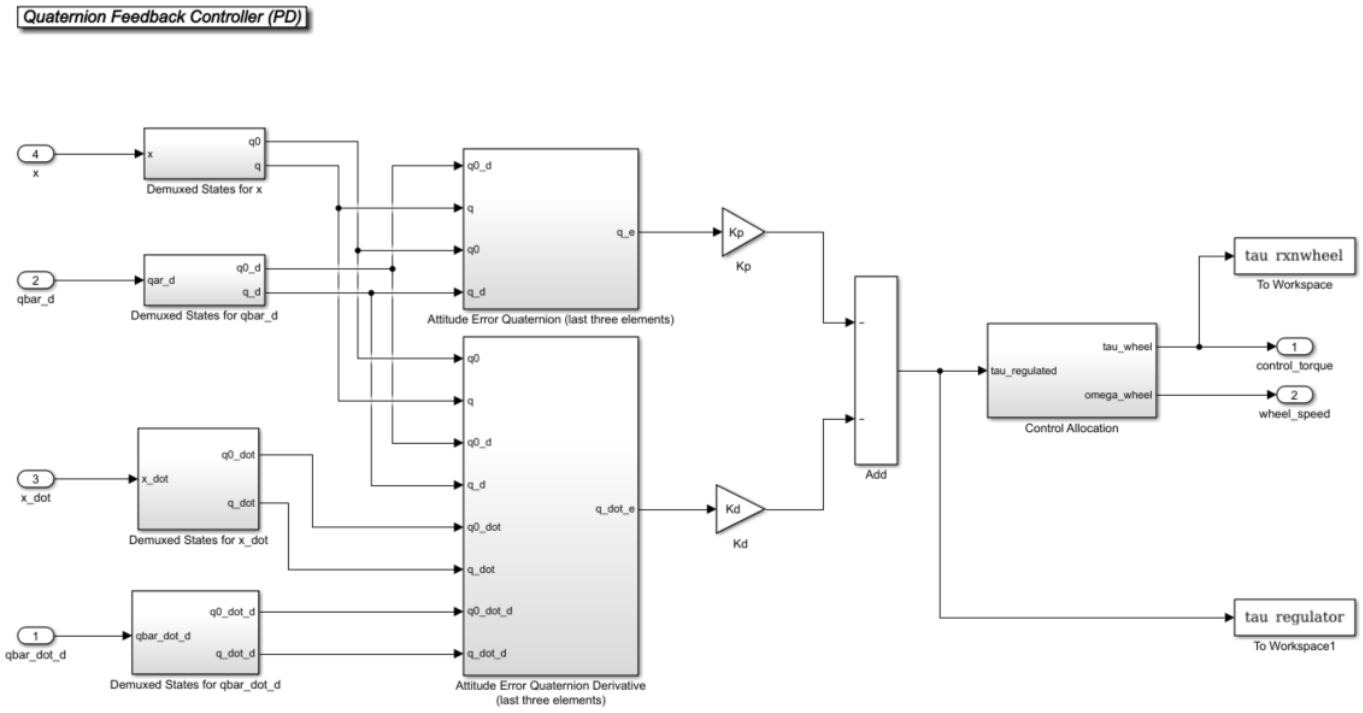

Figure 5: Quaternion Feedback Controller (PD) Block. The attitude error quaternion and it's derivative are formed, then multiplied by the proportional gain K_p and derivative gain K_d, respectively, and lastly, negated and summed. This forms the control input $u = -K_p\boldsymbol{q}_e - K_d\dot{\boldsymbol{q}}_e$, which is subsequently fed into the Control Allocation block.

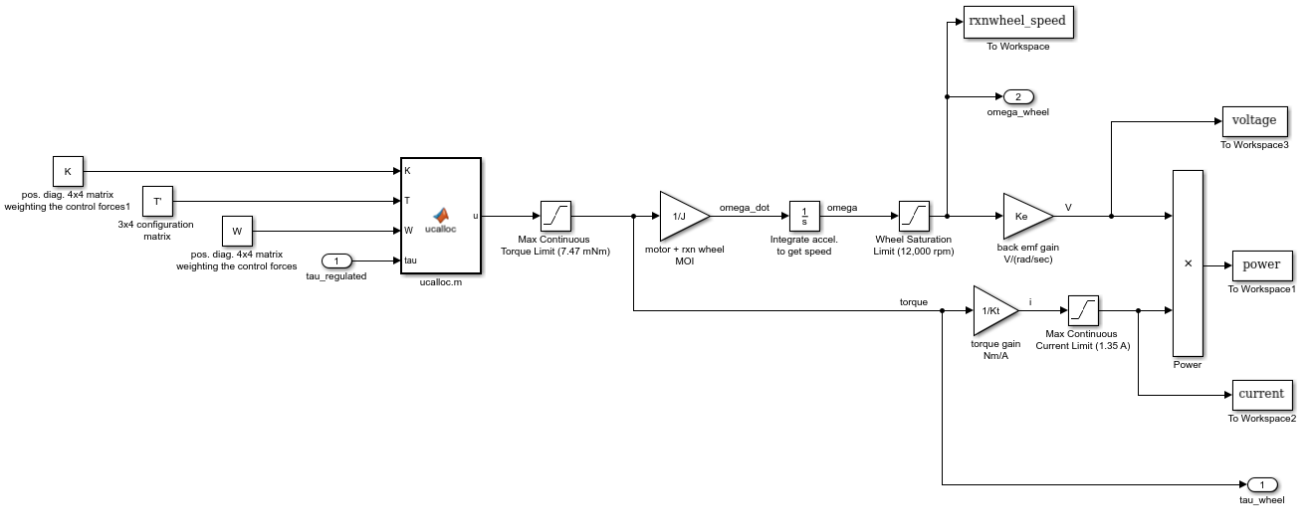

Figure 6: Control Allocation Block. Regulated body-fixed frame torques are generated from the quaternion feedback controller and defined as the input to the Control Allocation block. The regulated torque is then distributed to individual reaction wheel torques via the `ucalloc.m` function. Then using the torque relations provided by the motor's datasheet, wheel speeds, wheel torques, current, voltage, and power can be determined.

`ucalloc.m` – Unconstrained control allocation function, adapted from the Marine Guidance, Navigation, and Control Toolbox, Copyright© 2008, Thor I. Fossen and Tristan Perez. This function distributes the regulated body torques to individual reaction wheel torques, based on the distribution matrix and weighting matrices defined in `Init_Sim.m`.

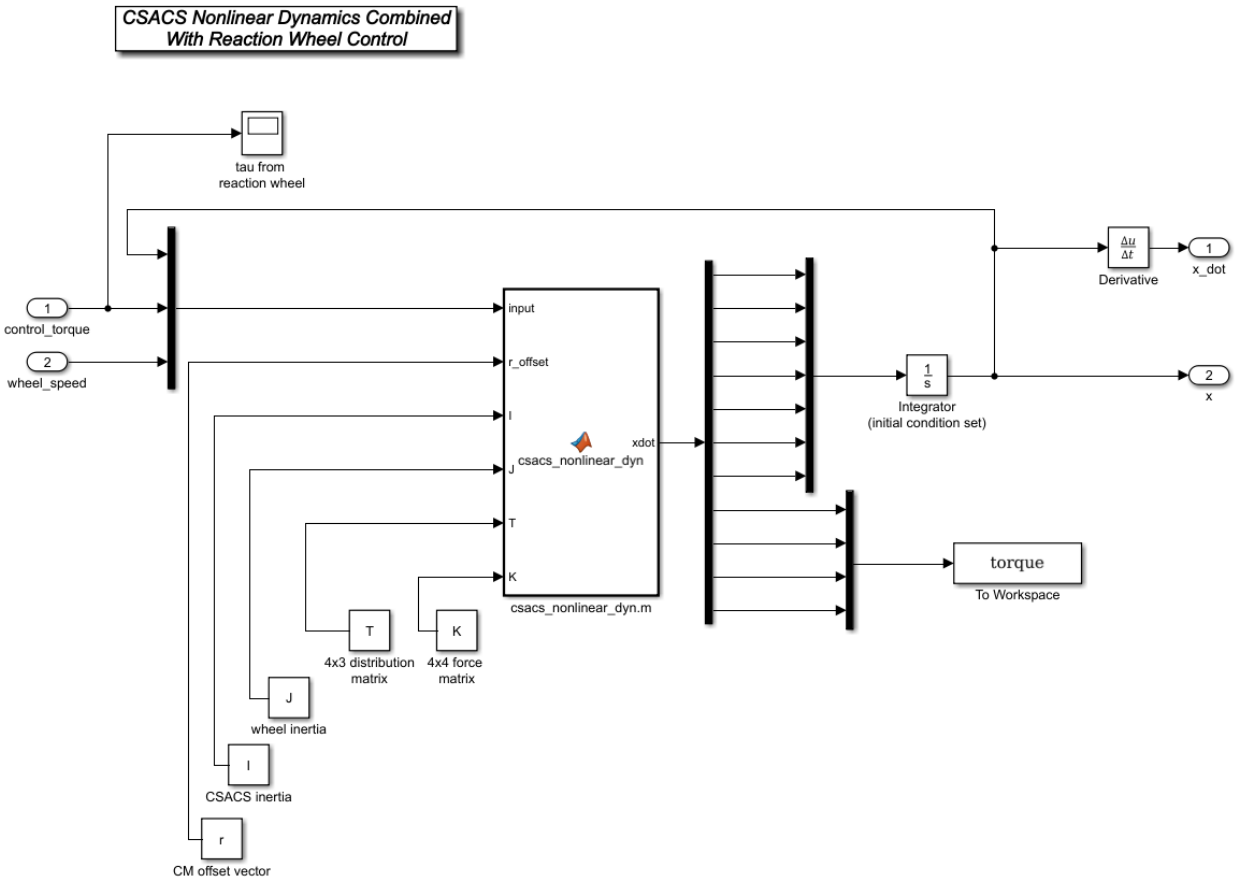

Figure 7: CSACS Nonlinear Dynamics Combined With Reaction Wheel Control Block. Distributed control torques and reaction wheel speeds from the Control Allocation block are now used as inputs for the full nonlinear dynamics model function, `csacs_nonlinear_dyn.m`.

`csacs_nonlinear_dyn.m` – Function that calculates the nonlinear dynamic equations of the CSACS, coupled with the RWA. Gravitational torque is the only disturbance torque considered, see thesis for full derivation.

`Rquat.m` – Sub-function used by `csacs_nonlinear_dyn.m` that generates the 3-2-1 rotation matrix sequence from the unit quaternions, where ψ is the first rotation, θ is the second rotation, and ϕ is the third rotation.

`skew.m` – Sub-function that is used by `csacs_nonlinear_dyn.m`, along with the Attitude Error Quaternion and Attitude Error Quaternion Derivative blocks, located in the Quaternion Feedback Controller (PD) block from Fig. 5. This function generates the skew-symmetric representation of a 3×1 column vector.

- `figures.m` – After `RWA_Controller.slx` successfully simulates the trajectory tracking problem, executing this file will generate the time-history plots for the Euler angles, body-rates, regulated torques, reaction wheel torques, reaction wheel speeds, currents, and voltages. Additionally, the overall power consumption for the specified maneuver is printed.

- `animate.m` – Similar to before, after `RWA_Controller.slx` generates the simulation results, `animate.m` can be called to generate a 3D animation of the maneuver. The script is self-contained, so all the user needs to do is run the code. The only requirement is that `RWA_Controller.slx` must be successfully executed in order for `animate.m` to extract the trajectory and wheel speed data. In the case of the quaternion feedback controller, reaction wheel speeds are no longer null like they were for the auto-balance routine. Multiple screenshots of the animation are shown in Fig. 8, where the dynamics of the CSACS are simulated according to "Case 1 – Four Functioning Wheels." That is, the initial Euler angles and angular velocity are zero, desired yaw-maneuver is -90°, settling time $t_s = 60$ sec, and damping ratio $\zeta = 1$. Any questions as to why certain values were chosen, please refer to my thesis. Lastly, note that the functions `R1.m`, `R2.m`, and `R3.m` are used by `animate.m` to rotate the various geometries about the 1, 2, and 3 axes, respectively, thus producing dynamic, 3D spatial rotation animations.

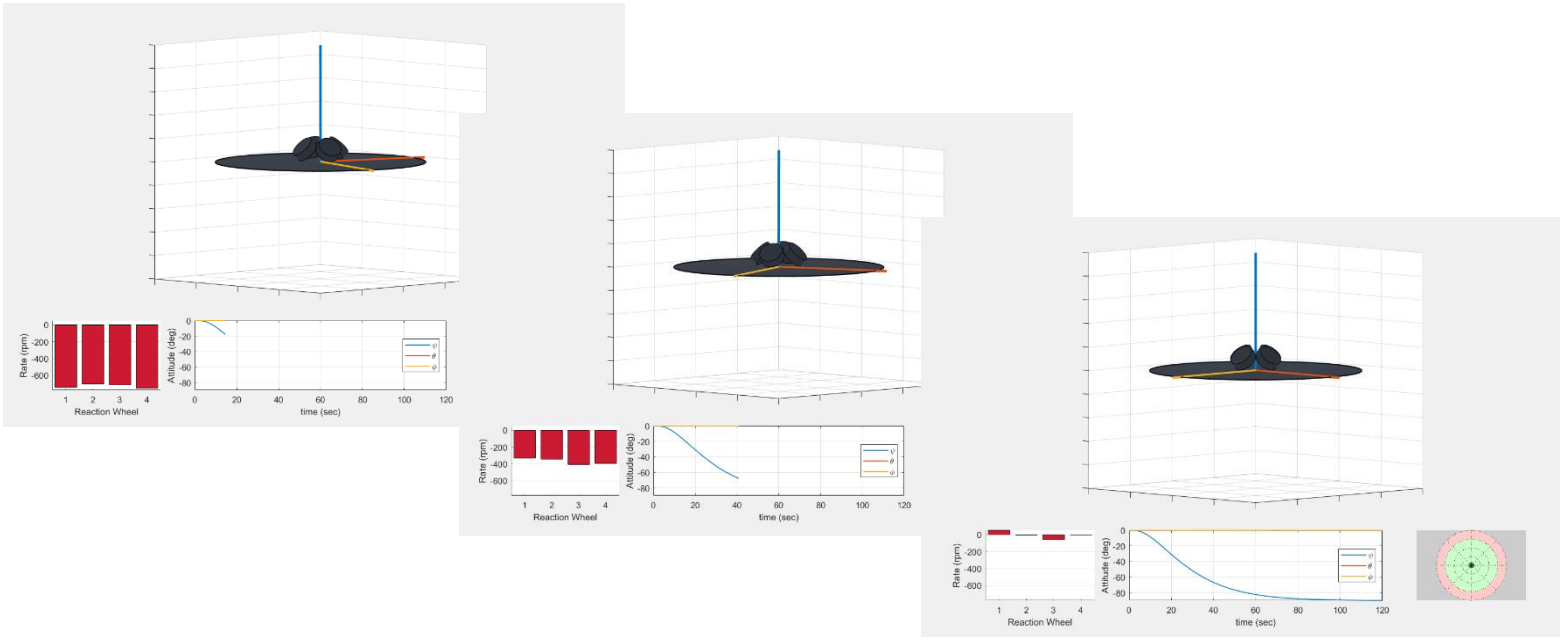

Figure 8: 3D Animated Display of the CSACS Trajectory Tracking Controller. Three screenshots at varying times during the tracking controller simulation show that the -90° yaw-maneuver is performed in adequate time, while keeping the CSACS perfectly balanced on the origin.