Cyber Physical System Design

William Willie Wells
University of Nebraska-Lincoln, Lincoln, NE 68588-0115
wwells@cse.unl.edu

September 2016

1 Controller Design

1.1 Derive Equations of Motion

If there is a change in an object's position, displacement ($\Delta\vec{s}$), with respect to a change in time (Δt) then that object has moved, $\vec{v} = \Delta\vec{s}/\Delta$t. If the net force acting upon an object is non zero ($\sum\vec{F} \neq$ 0), then there is a change in motion of the object with respect to a change in time ($\sum\vec{F} \propto \Delta\vec{v}/\Delta$t). Which leads to experiments to determine $\sum\vec{F} =$ m($\Delta\vec{v}/\Delta$t) [4].

In classical mechanics displacement is represented as a vector in Euclidean space ($\Delta\vec{s} =$ < Δx, Δy, Δz>), but can also be represented as a change in the vector of rotational angles that describe the orientation of the object ($\Delta\vec{s} =$ < $\Delta\phi$, $\Delta\theta$, $\Delta\psi$ >). With $\phi =$ atan(z/y), $\theta =$ atan(x/z), $\psi =$ atan(y/x), and $\vec{\omega} = \Delta\vec{s}/\Delta$t. Thus $\sum\vec{F} =$ m($\Delta\vec{v}/\Delta$t) becomes $\sum\vec{\tau} =$ J($\Delta\vec{\omega}/\Delta$t). Kinetic Energy (K) becomes K = (1/2)(J)ω^2. Assume Potential Energy (P) becomes P $\propto \vec{s}^2$. The Differential Equation: m($\delta\vec{v}/\delta$t) - $b\vec{v} - \delta\vec{P}/\delta\vec{s} = \vec{F}_{applied}$ becomes J($\delta\vec{\omega}/\delta$t) - $\beta\vec{\omega} - \delta\vec{P}/\delta\vec{s} = \vec{\tau}_{applied}$. Where b and β represent lossy force (friction, drag, etc) constants [4].

Only the angles that share the z axis have potential energy terms–are effected by gravity ($\phi =$ atan(z/y), $\theta =$ atan(x/z)).

Through reverse engineering: the constants in the second term of the above Differential Equation are orbital angular velocity (ω_0) multiplied by the difference of the inertias of the other angles (e.g. for $\phi : J_\psi - J_\theta$). Similarly, the constants in the third term of the above Differential Equation are -3(ω_0^2) multiplied by the difference of the inertias of the other angles (e.g. for $\phi : J_\psi - J_\theta$). Given that $\delta\vec{s}/\delta\vec{t}$ for ϕ and ψ depend on each other (due to structural design), their respective lossy force terms are non-zero as a consequence of this dependency. The resulting equation for ϕ is in the form $\delta\vec{\omega}/\delta\vec{t} =$ c$_1(\omega_\psi) + c_2$ and similarly for ψ.

1.2 Design a controller of some kind using the equations of motions.

Define $\vec{x} =$ < $\phi, \theta, \psi, \omega_\phi, \omega_\theta, \omega_\psi$ >, $\delta\vec{x}/\delta\vec{t} =$ A$\vec{x}$ + B$\vec{u}$, and $\vec{u} =$ -k$\vec{x}$. Where A and B are matrices of constants. Since the target controller is a design of $\vec{\tau}_{applied}$ or $\vec{M} = \vec{u}$, the units of -k$\vec{x}$ should be in $kg(m^2)/s^2$. Where M stands for moments and τ stands for torque. The symbol J is used for the inertia matrix.

The first design implemented compensated directly for the K and P energy terms, added ω_0/Δt terms, and added $1/\Delta$t terms. Not only were the units off but this controller was unstable. The next design did not consider units directly but aimed at acheiving a stable system. The third design incorporated units and stability but involved $\tau's >>$ 50 Nm. The fourth and final iteration of the controller design reduced the required $\tau's <$ 50 Nm [2].

The final value of k and u:

$$k = \begin{vmatrix} 3(J_\psi - J_\theta) + J_\phi & 0 & J_\phi & J_\phi & 0 & J_\psi - J_\theta + J_\phi \\ 0 & 3(J_\psi - J_\phi) + J_\theta & 0 & 0 & J_\theta & 0 \\ -J_\psi & 0 & (J_\theta - J_\phi) & -(J_\theta - J_\phi) & 0 & J_\psi \end{vmatrix}$$

$$u = \begin{vmatrix} -(3(J_\psi - J_\theta) + J_\phi)\phi - J_\phi\psi - J_\phi\omega_\phi + (J_\theta - J_\psi - J_\phi)\omega_\psi \\ -(3(J_\psi - J_\phi) + J_\theta)\theta - J_\theta\omega_\theta \\ J_\psi\phi - (J_\theta - J_\phi)\psi + (J_\theta - J_\phi)\omega_\phi - J_\psi\omega_\psi \end{vmatrix}.$$

1.3 Prove stability of the controller

While designing the controller MATLAB was used and eig(A-Bk) was evaluated against the stability condition such that $\forall\lambda, \lambda < 0$.

$$\lambda = \begin{vmatrix} -0.213373552315616 + 1.415601799285033j \\ -0.213373552315616 - 1.415601799285033j \\ -0.786626447684385 + 0.509411206085334j \\ -0.786626447684385 - 0.509411206085334j \\ -0.500000000000000 + 1.774823116820377j \\ -0.500000000000000 - 1.774823116820377j \end{vmatrix}.$$

2 Slewing Maneuver or Guidance

Write a paragraph about how you might control the vehicle to perform a slewing maneuver. In this case, because the equations have been linearized, they are only "valid" for small angles of rotation. If the slewing maneuver requires a large angular maneuver you will need a way of issuing small commands to your designed controller such that the controller can accomplish the designed maneuver. This is the essence of a "guidance" layer in your architecture.

The largest angular velocity one can issue is governed by the largest possible rotation (π) multiplied by the top layer trajectory control rate. Though, if this angular velocity is greater than the physical RPM limit, which for this problem it is, then the largest commanded angular velocity is further limited by the maximum rpm of the reaction wheels [2]. The greater the ratio of (trajectory control rate/controller control rate) the smaller the angular rotation per control command.

Compute e = r - y. Where e = error, r = top level reference trajectory, and y = state of the plant. Adjust e and r for the range [-π, π], feedback adjusted r for future guidance calculations. Set angular velocity limit. Set state target angular velocity components based on angular velocity limit and e. Set state target angles = previous state angles + controller control interval * target angular velocities. Pass target state to controller.

Obstacle avoidance should be handle in the top layer trajectory control. Checking sensor data, making a decision to change any forward looking trajectory points, and updating the next trajectory point should be handled in the top level trajectory control, which justifies a long execution time.

"... that takes a long time and by long time I mean a nanosecond," Sina Balkir, Very Large Scale Integration instructor.

3 Real Time System Design

3.1 Pseudocode

Write pseudocode for a Control task and a Guidance task. Include the task period you have chosen in the design. In each case justify the task period and demonstrate how it connects to the corresponding layer you have designed.

Control Task Pseudocode:

```
void controlTask(float &x, float &dxdt){
    // define constants
    J=...;
    w0=...;
    A=...;
    B=...;
    k=...;

    // compute the time derivative of state x
    dxdt = A*x + B*(k*x); // + noise
}
```

Guidance Task Pseudocode:

```
void guidanceTask(float &x, float &y, float &r, float dt){
    // define angular velocity constraint
    velocity_constraint=...;

    // compute error between target trajectory angles and current angles
    float ... e ...; // allocate space for e
    e=r(0:2) - y(0:2);

    // maintain angles in range [−π, π]
    for(int i=0; i<3; i++){
        if(e[i] > π){
            e[i]=...;
            r[i]=e[i]+y[i];
        }
        else if(e[i] < −π){
            e[i]=...;
            r[i]=e[i]+y[i];
        }
    }

    // apply velocity constraint and set next target angular velocities
    // then set next target angles
    for(int i=0; i<3; i++){
        if(e[i]/dt > velocity_constraint){
            x[i+3]=velocity_constraint;
        }
        else if(e[i]/dt < −velocity_constraint){
            x[i+3]=-velocity_constraint;
        }
        else{ x[i+3]=e[i]/dt; }
        x[i]=dt*x[i+3]+y[i];
    }
```

... e; // free space allocated for e
}

Every 1 ms the Guidance task executes then the Control task executes. The Guidance period is faster than the top level trajectory period in order to command smaller angular rotations than π while maintaining a reasonable control period. The Control task always executes after the Guidance task. The control task has the same period as the Guidance task but lags the Guidance task by the amount of time it takes to execute the Guidance task.

3.2 Operating System

Choose an OS or RTOS. Explain why you chose that OS or RTOS.

Though not that familair with available microcontroller operating systems. An operating system designed for microcontroller use is a desired operating system for this cubish satellite with its given dimensions of 0.3 m by 0.1 m by 0.1 m. After a short review, FreeRTOS seems like an idea candidate operating system for this cubish satellite because of its versatility, community support, design target of small embedded systems, supports real time tasks, supports existing hardware for small embedded systems, and claimed ease of use [1].

3.3 Scheduling Algorithm

Choose a scheduling algorithm. Explain why you chose that scheduling algorithm.

A rate-monotonic scheduling algorithm with the guidance task set at a higher rate than the navigation task (top level trajectory planning task). The control task should then be called at the end of the guidance task or respond to some event, E, that is triggered by the guidance task completing. Any sensor updates should be scheduled at the beginning of the navigation task. This system only involves scheduling of the navigation and the guidance tasks. The navigation task is preempted (split into multiple time slots) by the guidance task during its execution. The guidance task is guaranteed to start execution close to its set execution rate with a minimal worse case execution time, which in turn minimizes the deviation in execution times of the controller task. [3]

3.4 Worst Case Execution Time

Estimate the worst case execution time (WCET) of each task or describe how you might determine it.

The WCET for the Guidance task is:

$$t_G = t_{preemption} + t_{G\_execution}.$$

Where $t_{preemption}$ is the total time required to preempt the navigation task and start executing the guidance task and $t_{G\_execution}$ is the max time required to execute the guidance task. The WCET for the Control task is:

$$t_C = t_G + t_{C\_execution}.$$

The WCET for the Navigation task is:

$$t_N = n * t_{preemption} + n * t_{resume} + t_{N\_execution}.$$

Where n = the number of times the navigation task is preempted and t_{resume} is the time required to resume execution of the navigation task from where it was preempted.

3.5 Schedulable?

Consider a 3 task system. Planner task has execution time = 3 ms, and a task period = 10 ms. Show that your tasks are schedulable.

As long as $t_{preemption}$ and t_{resume} are much smaller than the Planner task execution time, the guidance and navigation tasks are schedulable. The navigation (Planner) task will be preempted at least twice by the guidance task during each planner task period. If $2*t_{preemption}+2*t_{resume}+t_C < 1ms$, then a third preemption will not occur. Furthermore, if $t_N+10*t_C < 10ms$ and there are no other tasks or interrupts, this 3 task system is schedulable.

4 Simulate the system and show plots.

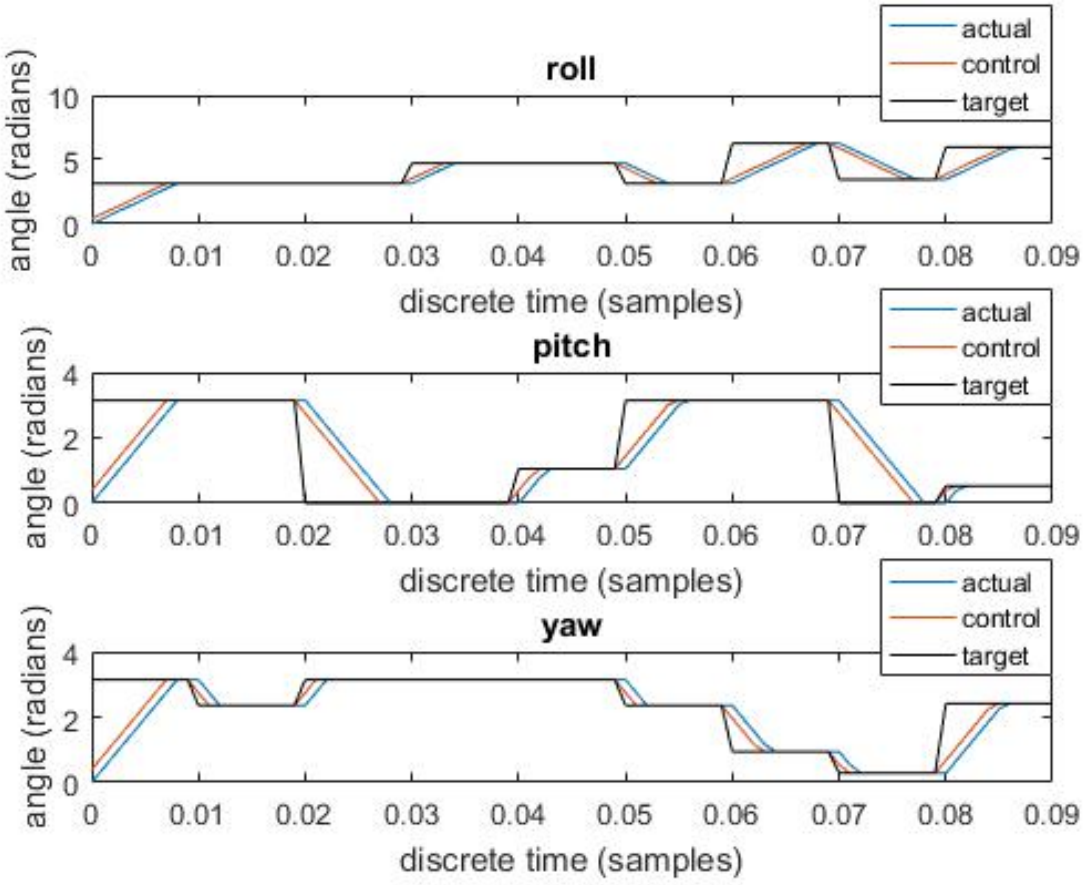

Figure 1: angles

A short navigation task is simulated with nine distinct roll, pitch, yaw combinations pseudo-randomly picked. Perfect timing is assumed, thus, no timing variations or WCETs are present in the simulation. Every 10 ms the current target angle triple is replaced by a new target

angle triple. The simulation code is written in MATLAB. In the above figure the cyan line is y, the full-state feedback, while the red line represents the computed control input. The black line is the target state.

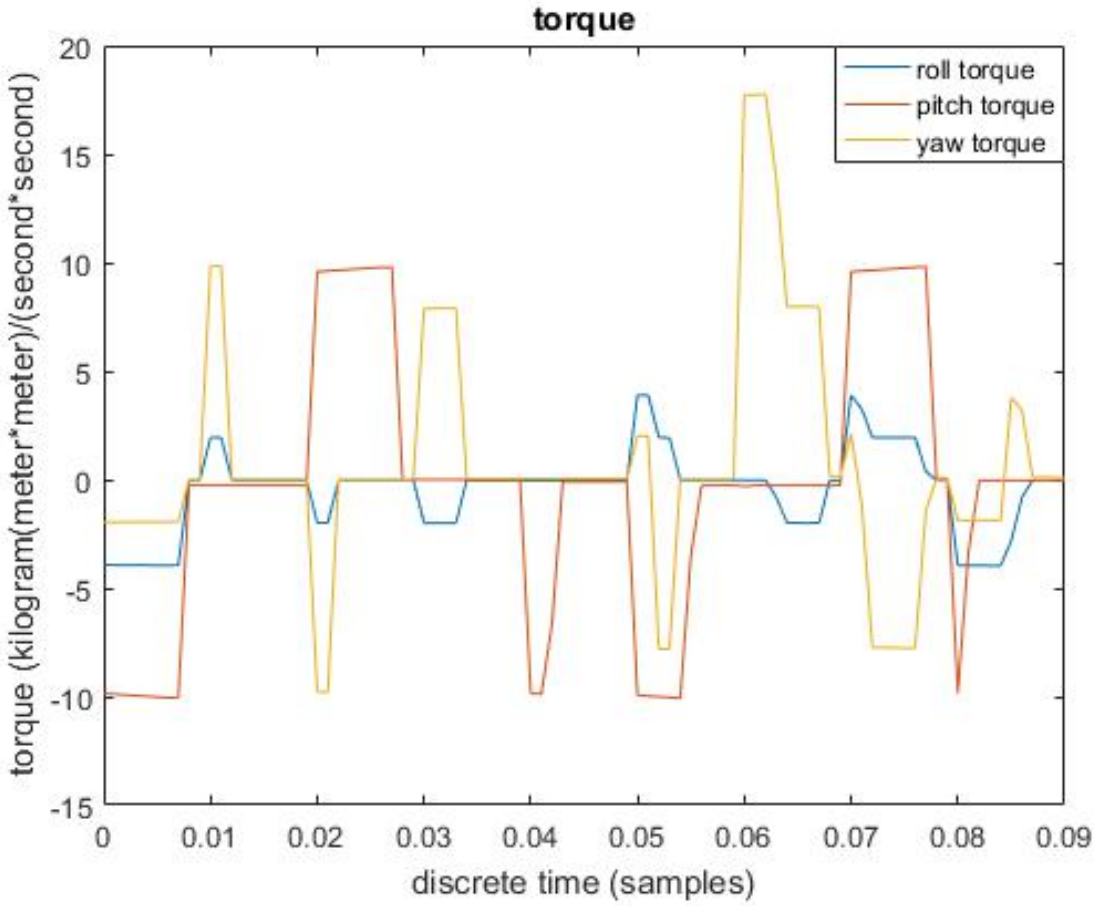

Figure 2: torque

The above figure displays torque, u, values while rotating to the selected targets. For this target set max torque in the roll direction is significantly less than the max torque in the pitch and yaw directions, which is most likely a consequence of $J_\phi < J_\psi = J_\theta$, which is in turn a consequence of the asymmetric dimensions of the not exactly cube satellite. Differences in max torque may also be attributed to the choice of k as well.

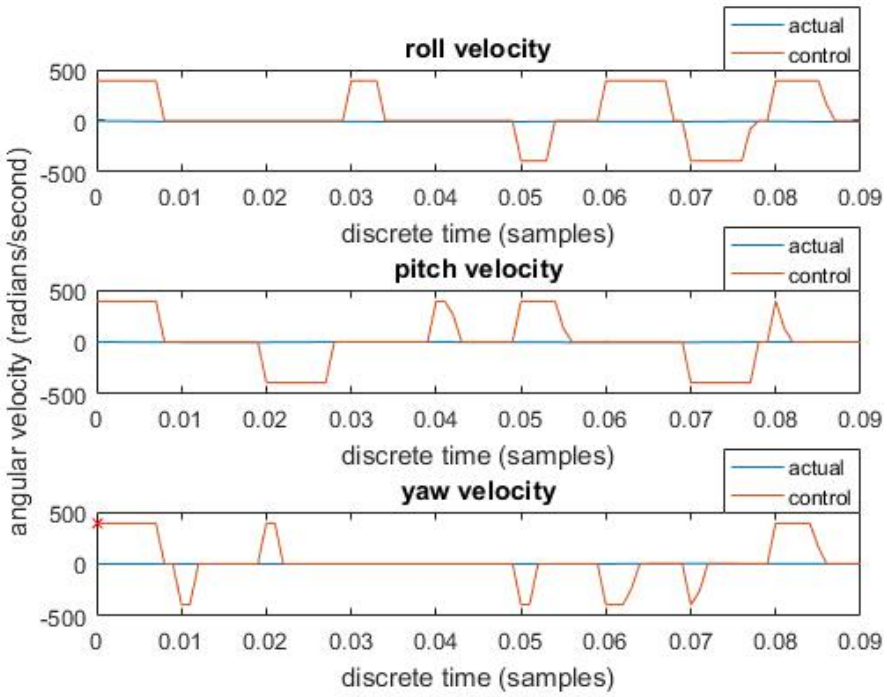

Figure 3: angular velocity

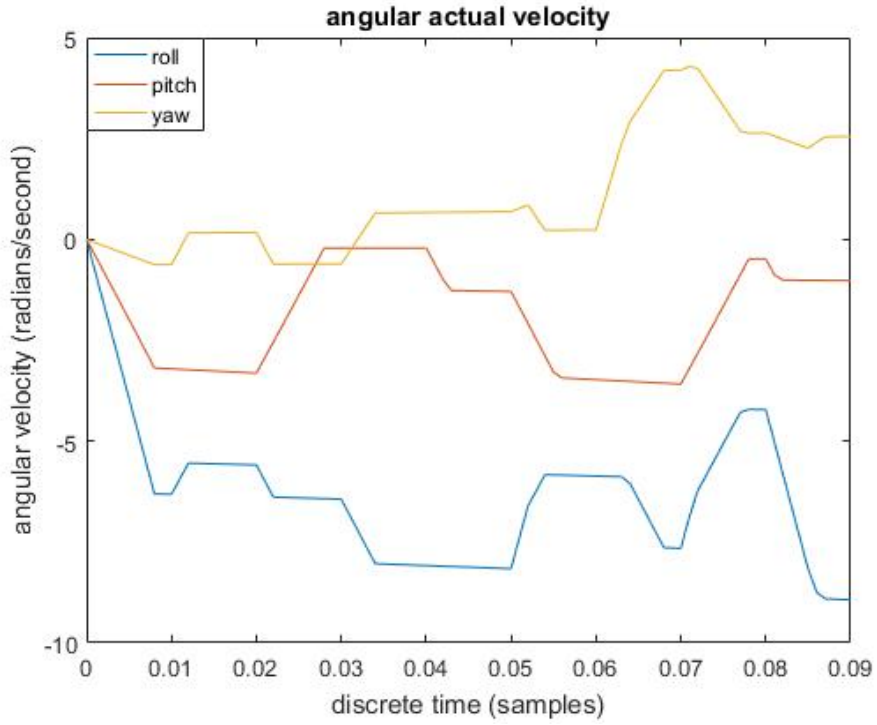

Figure 4: actual angular velocity

In figure 3 state angular velocities are depicted. Actual velocities values seem to remain constant, but in the next figure the actual velocities are shown to change but do not reach the control velocities due to the time it takes a reaction wheel to reach max velocity and the rate at which the next control signal is sent.

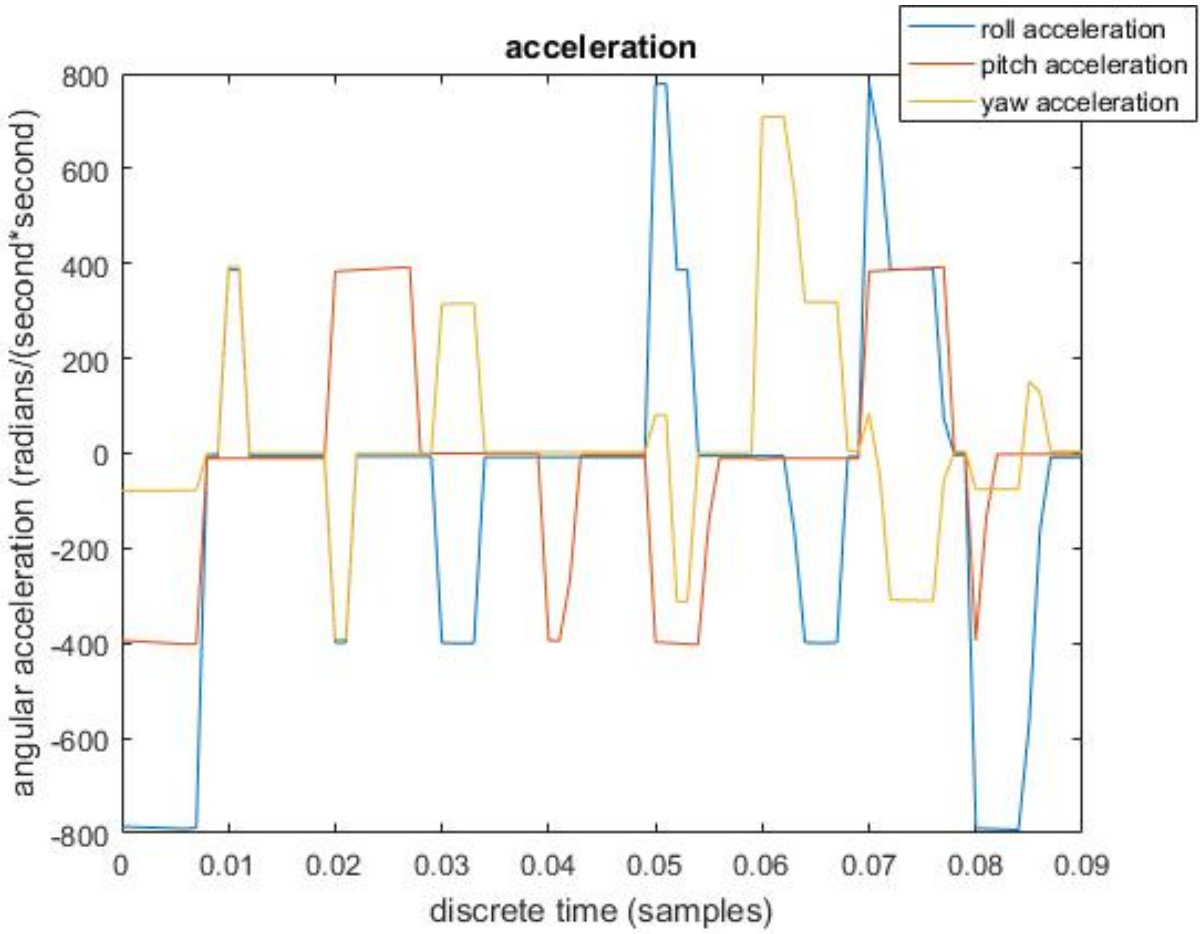

Figure 5: angular acceleration

The last figure shows angular acceleration.

5 Future Work

Write a couple paragraphs on which parts were easy and which parts were tough. What do you need to learn going forward to design this system well?

Deriving equations of angular motion for a cube satellite took the longest time and is still incomplete (e.g. how the constant are derived). Fighting LaTeX was tough and time consuming. By the time this report is finalized there will probably be errors and warnings still. Picking a scheduling algorithm was easiest, followed by picking an RTOS but only because the RTOS picked was mentioned in class.

An in depth explanation behind the drag force and potential energy constants would be illuminating.

Designing such a system and witnessing the performance of the system would be helpful in being able to design this system well.

References

[1] "FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions." [Online]. Available: http://www.freertos.org/. [Accessed: 23-Sep-2016].

[2] "High Motor Torque Momentum and Reaction Wheels." [Online]. Available: http://www.rockwellcollins.com/sitecore/content/Data/Products/Space_Components/Satellite_Stabilization_Wheels/High_Motor_Torque_Momentum_and_Reaction_Wheels.aspx. [Accessed: 23-Sep-2016].

[3] Silberschatz, A., Galvin, P. B., Gagne, G. Operating System Concepts. 9th edition. Wiley. Hoboken, N.J; 2013.

[4] Young, H.D., Freedman, R.A. University Physics. 11th edition. Pearson Addison Wesley, San Francisco, USA; 2004.