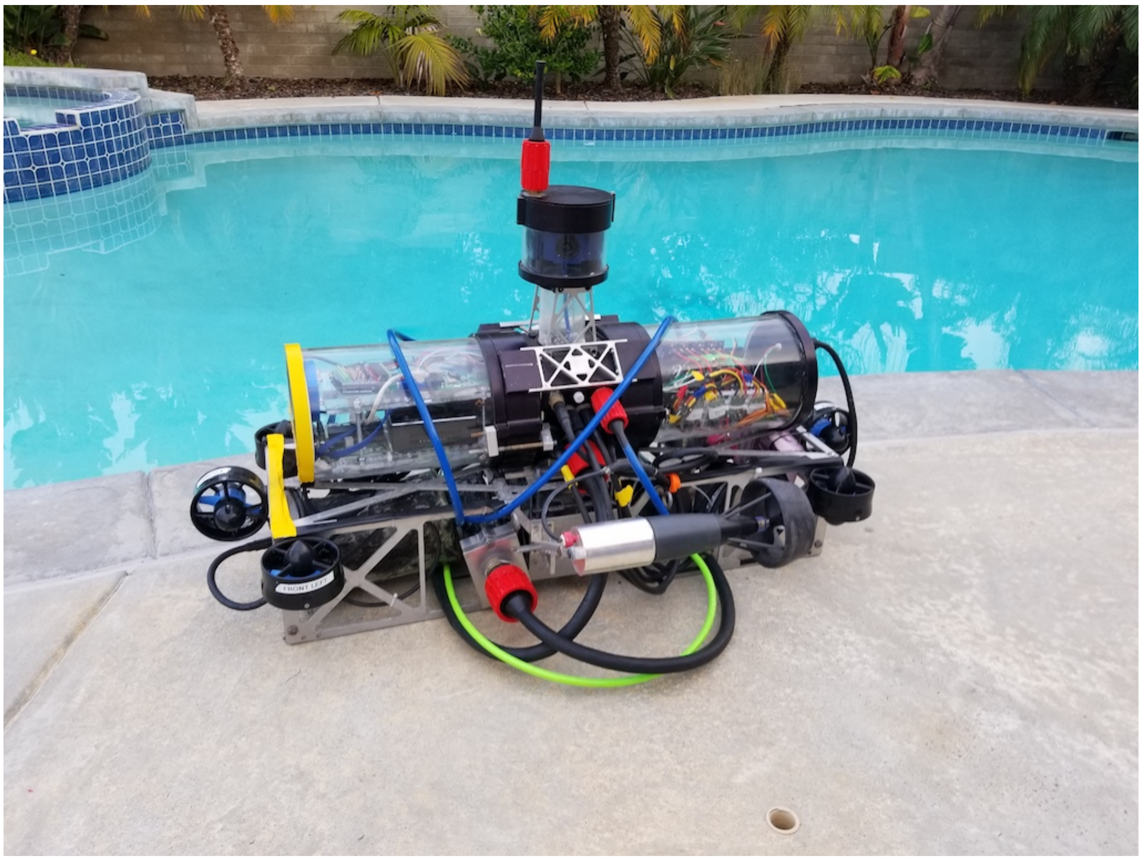
ECEN 5458 Project Report: Control of an Underwater Autonomous Vehicle

Kyle Harlow

May 18, 2018



Contents

1 Introduction

Autonomous Underwater Vehicles (AUVs) are complex systems with interesting control properties. Given a variety of sensors and state estimators providing information at varying rates, yet with continuous motor outputs and outside forces this system becomes a sampled-data control system. Through this project we will analyze how to mathematically model the system, and then show how it can be controlled using direct digital state space control methods.

2 Analysis

2.1 System Identification

The primary differential equation for modeling a 6 degree of freedom (DOF) underwater vehicle is [1]

$$\begin{aligned} M\dot{v} + C(v)v + D(v)v + g(\eta) &= \tau \\ \eta &= [N, E, D, \phi, \theta, \psi] \\ \vec{v} &= [\dot{x}, \dot{y}, \dot{z}, \dot{r}, \dot{p}, \dot{q}] \end{aligned} \tag{1}$$

With global coordinates being represented by η in a North-East-Down left hand frame and body frame coordinates being represented by $\vec{v}$ with x-y-z, roll-pitch-yaw in a left hand frame. We can relate η to $\vec{v}$ by the Euler Jacobian as $\dot{\eta} = J(\eta)\vec{v}$ assuming small angles to avoid singularities in J. In order to linearize Equation 1 we make several assumptions [2].

1. Only Small Roll and Pitch angles are possible
2. The vehicle is low speed
3. The vehicle is symmetrical across the xz-plane, yz-plane, and xy-plane

Using these assumptions we can diagonalize M, ignore the Coriolis affect $C(v)$, diagonalize and linearize D such that $D(v) \approx D$ and diagonalize G [2]. It also allows us to make the transformation $\dot{\eta} = J(\eta)\vec{v} \approx P(\psi)\vec{v}$ where

$$P(\psi) = \begin{bmatrix} R(\psi) & 0_{3x3} \\ 0_{3x3} & I_{3x3} \end{bmatrix}$$

Since P is also a projection matrix with $P^T = P^{-1}$ we can transform our world coordinates into body relative coordinates with $\eta_p = P^T(\psi)\eta$ and then make the simplification that $\dot{\eta_p} = v$ [1].

This gives us a simplified model

$$M\dot{v} + Dv + G\eta_p = \tau \tag{2}$$

Since M, D, and G are diagonal we can separate this equation into six 2DOF systems with states [3]

$$\begin{aligned} \hat{x}_1 &= \begin{bmatrix} N \\ \dot{x} \end{bmatrix} \\ \hat{x}_2 &= \begin{bmatrix} E \\ \dot{y} \end{bmatrix} \\ &\vdots \\ \hat{x}_6 &= \begin{bmatrix} \psi \\ \dot{q} \end{bmatrix} \end{aligned} \tag{3}$$

and continuous-time state space matrices

$$\begin{aligned} A_i &= \begin{bmatrix} 0 & 1 \\ \frac{-g_{ii}}{m_{ii}} & \frac{-d_{ii}}{m_{ii}} \end{bmatrix} \\ B_i &= \begin{bmatrix} 0 \\ \frac{1}{m_{ii}} \end{bmatrix} \\ C_i &= \begin{bmatrix} 1 & 0 \end{bmatrix} \\ D_i &= \begin{bmatrix} 0 \end{bmatrix} \end{aligned} \tag{4}$$

2.2 Desired Control Parameters

The desired control parameters are position and velocity of the AUV in N,E,D directions and yaw. Roll and pitch are stable, and negligible due to the above assumptions. The state space model A_i B_i C_i D_i were converted to the equivalent discrete state space model using c2d Matlab function and appropriate sample period T.
There are two sample period T in this system, T = 1/8 for the DVL sensor which is hardwired into the system and T = 1/40 state estimator's frequency which is chosen for this system. Modeling and simulation is done for both the sample rates for comparison.
Then the model is checked for controllability and Observability by checking whether the Controllability and Observability matrix are full rank. The Controllability(C) and Observability(O) Matrix are given by

$$C = \begin{bmatrix} B & AB & A^2B & ... & A^{n-1}B \end{bmatrix} \quad O = \begin{bmatrix} C \\ CA \\ CA^2 \\ ... \\ CA^{n-1} \end{bmatrix} \tag{5}$$

$$rank(\mathcal{C}_\rangle) = rank(\mathcal{O}_\rangle) = 2 \quad \forall i \in 1, 2, ..., 6 \tag{6}$$

The poles are calculated from the state space equations for each states and stored in a variable. The aim is to control these separate six 2DOF equations using full state feedback controller using the reference input. The feedback controller is defined as

$$\overline{N} = N_u + KN_x \tag{7}$$

A simple block diagram of the controller is shown in Figure 1.

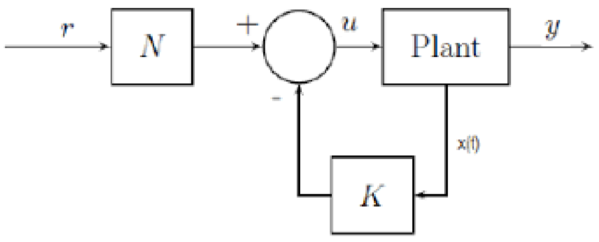


Figure 1: Example of Feedback Control

Each state has a different controller with different K and N. First, the estimated pole is calculated for the desired overshoot and rise time and calculating ω_n and ζ respectively.This pole is then converted to z-domain pole using sample time T. The inbuilt "place" command uses the model's A_i,B_i and the estimated z-domain pole to compute the gain vector K and place the full state feedback for u= -Kx at the estimated pole. After computing K, the N_u and N_x is calculated as

$$\begin{bmatrix} N_x \\ N_u \end{bmatrix} = \begin{bmatrix} \psi - I & \Gamma \\ H & J \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{8}$$

Then using equation 8, $\bar{N}$ is calculated and multiplied with input reference and fed to the system.
The closed loop poles for observer gain, L, is defined by using place function. The poles that are inputted to the place function are 100 times the original poles and then converted to z-domain. The gain matrix M is calculated as M = Γ N. This is the gain which is multiplied by reference input and added to estimator along with observer gain.

The desired poles locations are p = [0.993-0.00894i, 0.993+0.00894i] for T = 1/40 and p = [0.96-0.048i, 0.96+0.048i] for T = 1/8. These locations are chosen based on the overshoot and rise time specifications. The desired Overshoot is 10% and rise time is 4sec which are suitable given the actual operating range of the AUV. Based on overshoot and rise time, W$_n$ and ζ are calculated for the 2DOF system. Overall, the system is rather slow due to its large weight and the high drag, but in order to conduct fine movements low overshoot is necessary. Based on these, the characteristic equation for a second order system is found out and the poles of this equation is mapped to z-domain using $z = e^{sT}$. This is how we get the specified poles for T=1/8 and T=1/40.

Similarly the desire pole locations for the Observer gain is calculated for poles which are 100 times the original poles in the s-domain and converted to z-domain, which satisfies the required damping and rise time. The observer poles are found out to be p_{obs} = [0.3174+0.245i,0.3174-0.245i] and p_{obs} = [-0.0108+0.00257i,-0.0108-0.00257i] for T=1/40 and T=1/8 respectively. These poles were found with a desired rise time of approximately 0.4sec and a desired overshoot of 1%, significantly faster poles than the feedback control poles.

The poles which are found out in both the cases, with and without observer, are both inside the unit-circle in z-plane and hence system is stable. We also predict that we will be able to meet our desired specifications of 10 percent overshoot and a 4 second rise time.

2.3 Controller

The model of the system using controller only is shown in Figure 2. Here, the gain, calculated

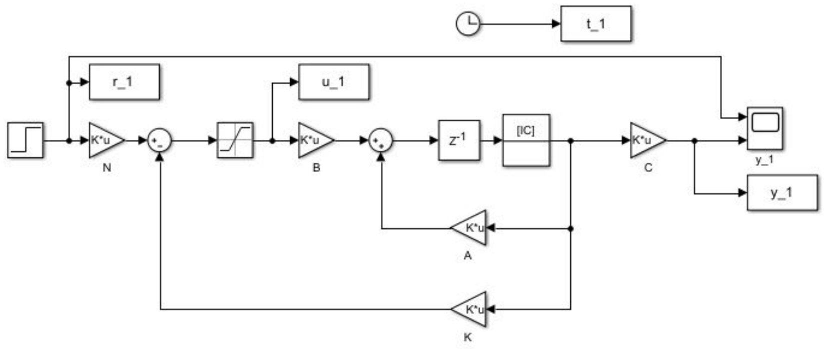


Figure 2: State Space Feedback Controller

using place function, is subtracted from the input and fed into the system. Also, the reference input is multiplied by the $\bar{N}$ computed earlier. The plant is a simple state space plant with A,B and C of the discrete time system. The output of the system is compared to the step input and plots are plotted. Simulation time t1 is set using clock.The same controller and model is used for X,Y,Z directions and yaw.

2.4 Observer

The model of the system using controller and observer is shown in Figure 3. Here, in addition to the original plant, we have an observer plant which has reference gain M and Observer gain L. The estimated output from the system $\hat{y}$ is subtracted from y and fed back through the estimator using the estimated states as the feedback through K. This signal is added to the original input multiplied with N. Such is the design of the plant with observer and controller. We expect this will improve the response to feedback control, especially given non-zero initial conditions.

3 Results

Throughout the course of this project we ran several experiments in order to determine optimal pole placement. The primary experiments were based around collecting step responses for the North, East, and Down axis, as well as the yaw of the vehicle. We first ran these step responses for both our 8 Hz and 40 Hz with feedback control. Then we ran the step responses again with the observer implemented, and with non-zero initial conditions for our variables at 8 Hz and finally 40 Hz.

Overall this resulted in 26 different experiments, of which we will be analyzing the north experiments, and the yaw experiments. The north experiments generalize to both the east and down degrees of freedom. The different saturation limits, and lower drag on the north movement do not cause the results to differ significantly. The yaw experiment is the most unique, with a

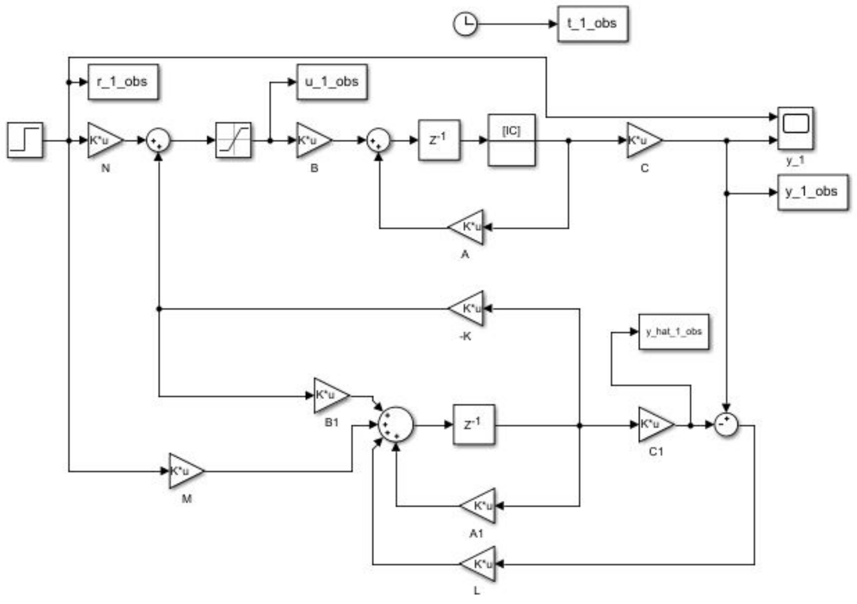


Figure 3: Full State Estimator Feedback Controller

$\pi/2$ step response, and with undershoot given full state variable feedback. Implying non-minimum phase zeros.

Overall, our systems behaved as expected running the first set of experiments with T=1/8, and T=1/40 and simple feedback control. The rise time for all systems with T=1/8 was less than 3.74 seconds, and the maximum overshoot of any system was 8.039% on east movements. With T=1/40 the rise time was actually slightly slower, with all controllers having a rise time of 4.1378 seconds.The maximum overshoot was still under our desired specifications with all responses having 8.741% overshoot.

Given full state variable feedback at T=1/8, and initial conditions for our state vectors at 0.02, 0.03, the performance specifications were still met. The highest rise time came from north movements, at 3.85 seconds and highest overshoot came from down movements at 8.3%. The most interesting results come from T=1/40 with many performance specifications being missed. The largest rise time was for north movements at 4.61 seconds, and the largest overshoot was for yaw movements at 12.474%.

Overall these still fall in an acceptable range of values, especially given that the system may not be perfectly modeled. But additional tuning may be necessary in future iterations.

3.1 Feedback Control

To look closer at the results we will analyze several plots produced from Figure 2. Starting with our T = 1/8 North Step response in Figure 4

3.1.1 North

The feedback control with an 8Hz sample time performed quite well overall. The rise time for the system was $T_r = 3.7396s$, and the overshoot $M_p = 8.3028\%$ meeting both of the desired performance characteristics. The settling time, however was quite long at $T_s = 12.7230s$ which could have an affect on the continuous movements necessary for the vehicle. This was not, however, an aspect of the vehicle we were directly trying to control

Despite looking smoother, the T=1/40 North Step response actually performs a little worse. This response, shown in figure 5, has a rise time of $T_r = 4.1378s$, overshoot of $M_p = 8.7409$, and a settling time of $T_s = 14.0891s$. The system is still quite close to meeting all the desired specifications, just missing the rise time of 4 seconds. This would not likely affect the overall system's ability to perform tasks.

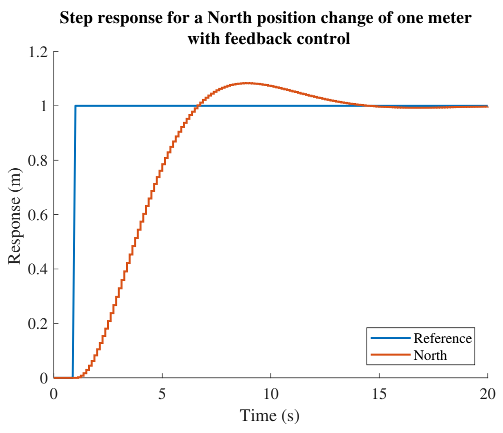


Figure 4: North Step Response with Feedback at 8Hz

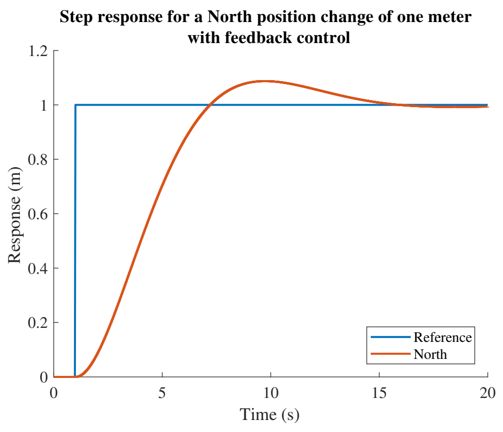


Figure 5: North Step Response with Feedback at 40HZ

3.1.2 Yaw

Instead of simply using 1 as the step response for a yaw motion, we chose to use $\pi/2$ to represent a $90°$ rotation about the z-axis. The 8Hz feedback response is shown in Figure 6. This system met all of our performance characteristics with a rise time of $T_r = 3.7380s$, and overshoot of $M_p = 8.3039\%$. The settling time was around $T_s = 12.6659s$.

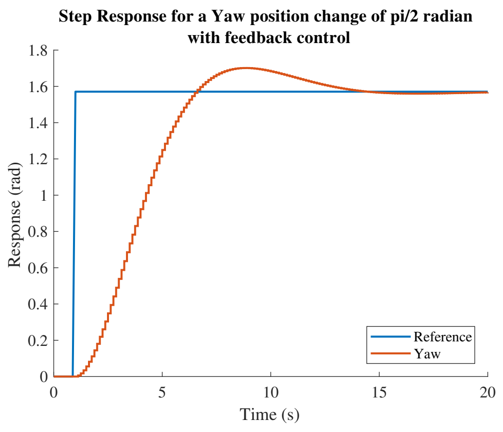


Figure 6: Yaw Step Response with Feedback at 8Hz

The 40Hz response for yaw is shown in Figure 7. Once again, despite looking smoother the system actually preformed slightly worse. The overshoot was still within our desired specifications, only increasing to $M_p = 8.7409\%$, but the rise time missed our desired target at $T_r = 4.1378s$ and the settling time increased to $T_s = 14.0801s$.

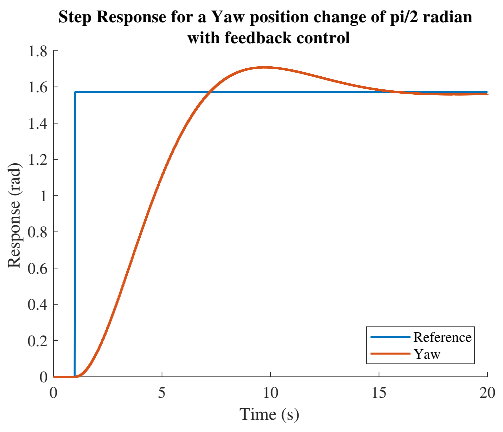


Figure 7: Yaw Step Response with Feedback at 40HZ

3.2 Full State Estimator Feedback Control

While the system responded quite well to state feedback control, it is often important to include an estimator to account for potential modeling error or non-zero initial conditions.

3.2.1 North

In order to properly show the effects of estimator feedback we first conducted an experiment using standard feedback with the initial conditions $[0.02, 0.03]^T$. The results of this experiment for T $=$ $1/8$ are shown in Figure 8. The rise time for this sytem was $T_r = 3.8272s$ and the overshoot was $M_p = 7.976\%$.

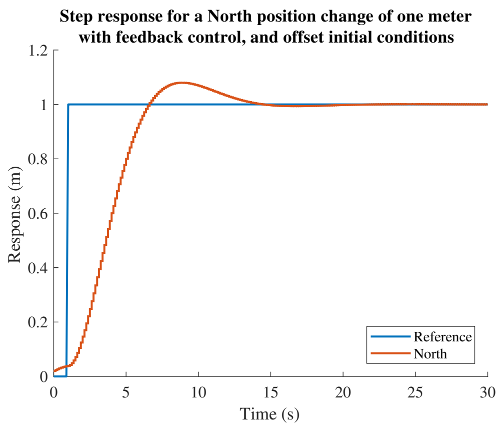


Figure 8: North Step Response with Feedback at 8Hz, and $[0.02, 0.03]^T$ initial conditions

For T $= 1/40$ shown in Figure 9 the rise time was $T_r = 4.2417s$, and the overshot was $M_p = 8.3901\%$. In both cases the non-zero initial conditions slowed the response time as well as the increased the overshoot.

In order to increase the robustness to these problems we can implement full state estimator feedback using the controller and observer shown in Figure 3. The step response for T $= 1/8$ with full state estimator feedback is shown if Figure 10. Despite the non-zero conditions the estimator allowed us to reduce the overshot slightly, at $M_p = 7.9393\%$, the rise time was slightly longer at $T_r = 3.8493s$, however this is still inside of our desired specifications.

A similar story appears in the T $= 1/40$ response, shown in Figure 11. The overshoot decreases slightly from the non-zero initial condition example to $M_p = 8.2774\%$ but the rise time increased to $T_r = 4.3166s$. The settling time interestingly enough decreased from 14.00s to 13.93 seconds.

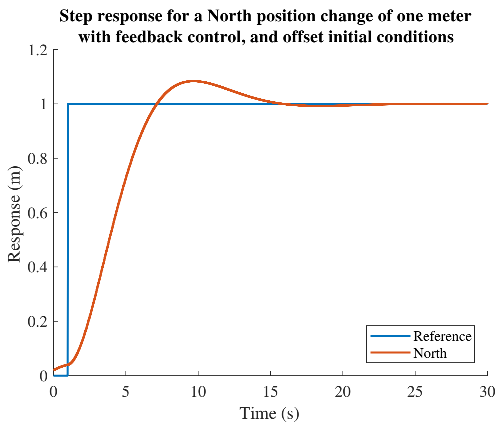


Figure 9: North Step Response with Feedback at 40Hz, and $[0.02, 0.03]^T$ initial conditions

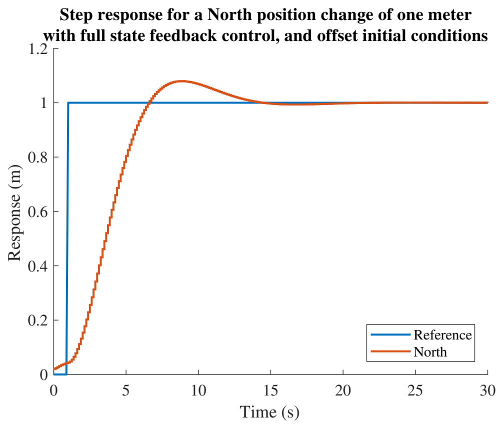


Figure 10: North Step Response with Estimator Feedback at 8Hz, and $[0.02, 0.03]^T$ initial conditions

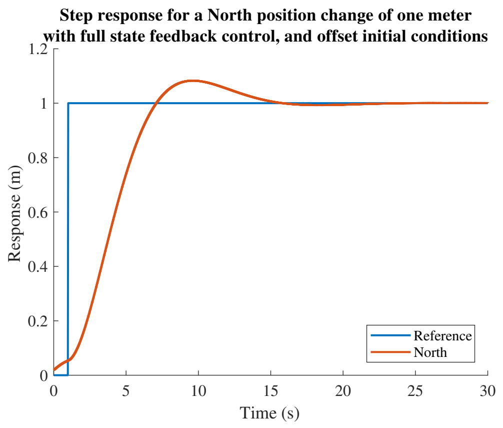


Figure 11: North Step Response with Estimator Feedback at 40Hz, and $[0.02, 0.03]^T$ initial conditions

Another way to analyze the observer is to see how quickly the estimator error converges to 0. For T $= 1/8$, shown in Figure 12, the estimator takes 4 time samples or 0.5 seconds to converge to zero. For the estimator at T $= 1/40$ the 7 time samples but only 0.175 seconds to converge to zero, as shown in Figure 3.2.1

These convergence times could likely explain the difference in rise time between the original feedback controller with initial conditions and the estimator feedback controller.

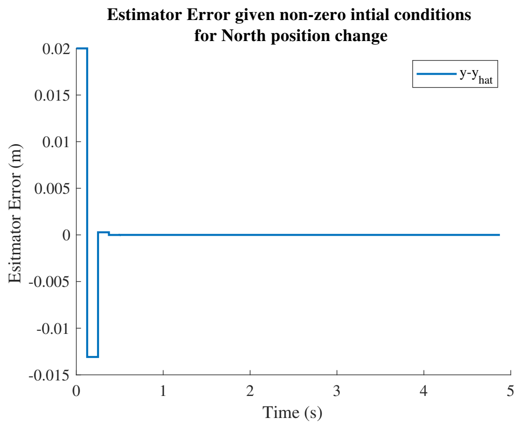


Figure 12: North Step Estimator error at 8Hz, and $[0.02, 0.03]^T$ initial conditions

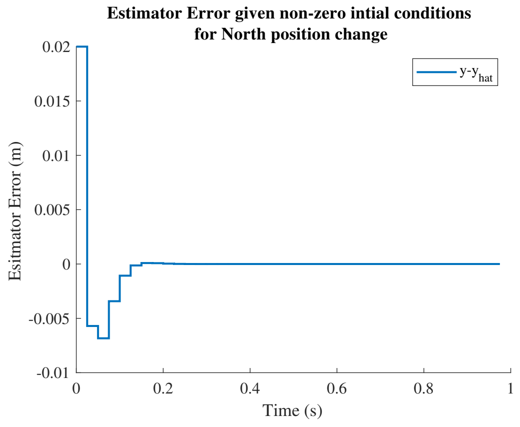


Figure 13: North Step Estimator error at 40Hz, and $[0.02, 0.03]^T$ initial conditions

3.2.2 Yaw

The T = 1/8 step responses for yaw acted similarly to the other responses. The step response shown in figure 14 had a rise time of $T_r = 3.7350s$, and an overshoot of $M_p = 8.2034\%$.

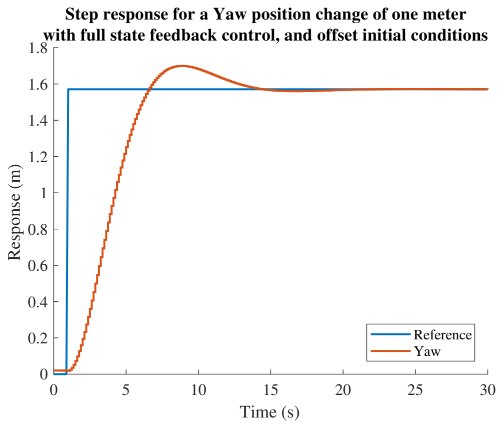


Figure 14: Yaw Step Response with Estimator Feedback at 8Hz, and $[0.02, 0.03]^T$ initial conditions

The estimator error converged in just 3 time steps or 0.375 seconds as shown in Figure 15.

The most interesting result from the experiments occurred at T = 1/40 on our yaw estimator feedback control. The system experience undershoot, implying that a non-minimum phase zero had been introduced to the system. The undershoot, shown in figure 16 was quite significant at 42%. It however had an extremely fast rise time, at $T_r = 3.0422s$. However, taking into account

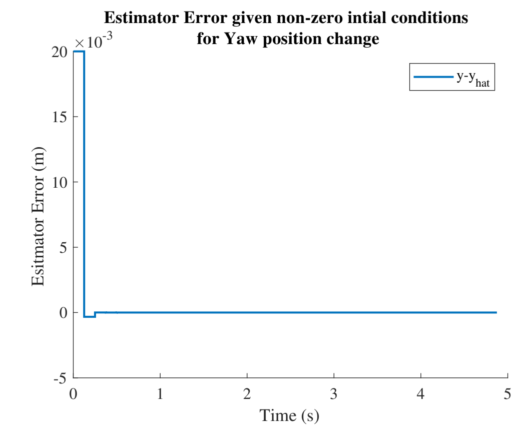


Figure 15: Yaw Step Estimator error at 8Hz, and $[0.02, 0.03]^T$ initial conditions

the undershoot and the rise time it took a full $6.3s$ before the vehicle reached $\pi/2 \cdot .9$. The difference in these times is likely based on how matlab calculates rise time from a step response.

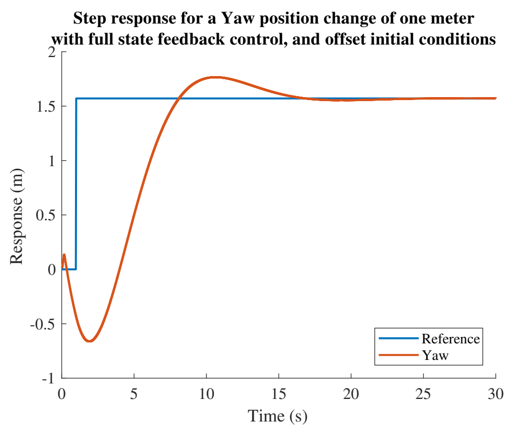


Figure 16: Yaw Step Response with Estimator Feedback at 40Hz, and $[0.02, 0.03]^T$ initial conditions

The estimator error shown in Figure 17 took significantly longer than the north estimator error. It took 13 time samples or 0.325 seconds to converge.

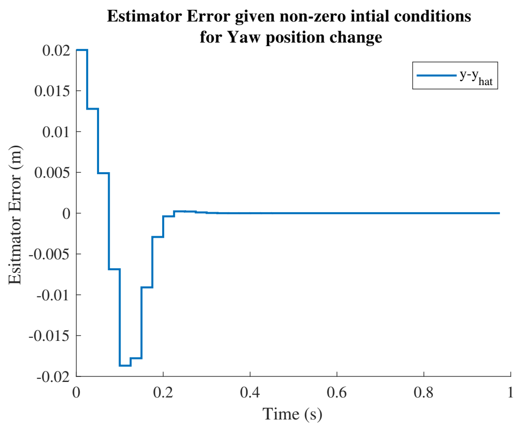


Figure 17: Yaw Step Estimator error at 40Hz, and $[0.02, 0.03]^T$ initial conditions

4 Discussion

A question for every controls engineer is often about how well the theory applies in both simulations and in real life. A mathematical controller may be designed to operate under ideal conditions, and mathematically be acceptable, however, the same may not always be true in even simulations let alone in the real world. As such, it is important to analyze these characteristic difference.

When looking through the results two interesting difference appear for our system in question. Across almost all tests, the 8 Hz time sample seems to perform better, beating our desired control parameters for every step response. This is most easily by comparing the difference between overshoot and rise time for our feedback controllers in Figures 4 and 5. By increasing our sample rate, the overshoot increased to 8.7409% from 8.3028% approximately 0.44%. And the rise time increased by .4 seconds as well from 3.74s to 4.14s.

These differences could be a result of one of two issues, or a combination of both. The first is that the controller with T = $1/8$ likely has more aggressive motor inputs u_i. As a result of the larger sample times the error for any given sample as it was increasing was likely also larger. This both created a less smooth overall response, but also could have the benefit of decreasing the rise time and overshoot slightly.

A second possibility is that the given a continuous plant, rather than a discrete plant that the interpolations might show less difference. Since we used an entirely digital state space there is likely inter-sample modeling errors where the physical vehicle would still be in motion, changing its positions and velocities continuously rather than in discrete intervals. This is less likely to cause significant changes though, since in a .4s difference their are still 3 samples at T = $1/8$, and 16 samples at T = $1/40$.

Since we generally were under the overshoot parameters by a few percent, it is possible that more aggressive poles could have allowed us to still meet both specifications. Since the rise time itself is quite slow, there is certainly room for improvement in this criteria, that still leaves relatively low overshoot. This is especially true due to the high damping from the drag experienced by the vehicle, in addition to the motor saturation limits. Further iteration could have possibly improved this responses slightly.

One of the most interesting divergences from our theory was the introduction of a non-minimum phase zero in our yaw plant. No other plant, including those only shown in the appendices, had this phenomenon occur. One possibility for it's introduction is modeling error, however careful comparisons between the Simulink models for yaw, and north movements do not yield any difference. Another possibility though, is that the non-linearity of the motor saturation causes problems for the model. For Yaw the saturation limits were set at a torque $\pm 75Nm$, based on the $50N$ motors positions away from the center of rotation. Changing these saturation limits to 100, or decreasing the initial conditions from by a factor of 10 eliminates the non-minimum phase. As such we predict that the initial undershoot is caused by the motors reacting, and saturating to the initial conditions. This is shown in Figure 18. Comparatively, setting the saturation limits to 100, the motor does not reach the maximum torque, and does not over-correct as severely, returning rather to 0 torque instead of starting from $-60Nm$

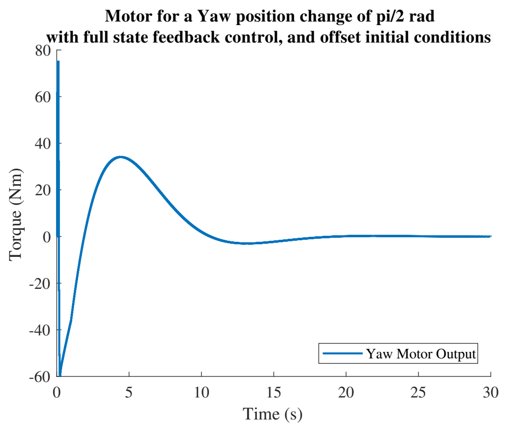


Figure 18: Yaw Motor Response at 40Hz, and $[0.02, 0.03]^T$ initial conditions

Overall the state space design method still appears to be incredibly useful. Controlling it in an actual system, such as the AUV relatively simple, just involving feeding back the states through a gain estimator after converting them to the proper output torques. Most programming languages and cpu architectures are also optimized around doing fast multiplications and additions, so even in the Linux Based Robot Operating System (ROS) a state space controller would be relatively easy to implement.

In addition, we can replace the observer with our Kalman Filter State Estimator. This would allows our system to be more robust to sensor noise disturbances vs. a standard observer state estimator. While we were unable to test this system on physical hardware, sensor noise, and modeling error are likely the two most likely sources of error in this project. Since the simulations were modeled primarily using SolidWorks there are likely a number of errors in estimations on the vehicles weights, inertia, and drags. The SolidWorks model in question did not include any cables, and had now way to calculate the constants for added mass. A full fluid simulator may be able to help in this regard, but this likely contributed to the issues with the full 12 DOF model being largely uncontrollable and unobservable.

5 Conclusion

This project provided many learning opportunities. Modeling the AUV took significantly more effort than expected. Both linearizing the non-linear differential equation, as well as determining parameters to place in the new non-linear model proved quite difficult. Initially, the model was largely uncontrollable even after being broken into the 2DOF models. This defied the general physical intuition that a buoyant vehicle in water generally wants to sit at the surface, not moving. However, careful analysis of signs eventually caught errors in both the drag matrix, where the forces where acting with their respective torques, rather than against them; and the gravity matrix, where buoyancy and gravity had become additive quantities rather than opposing forces. Fixing these errors brought the once unstable initial plant to an extremely stable one. With only a few purely mathematical instabilities, such as the North, and East poles being located a z=1, left over.

Another mistake found in the process of picking the desired controller poles was in creating the characteristic equation for z. Since the plots in the book used lines of constant damping, we initially picked poles by only converting ω_n using $e^{\omega_n T}$. In essence the characteristic equation was

$$z^2 + 2\zeta e^{\omega_n T} z + e^{2\omega_n T}$$

This, however, left incredibly large overshoots, often over 40%. The final poles used throughout the report instead took the s domain poles and converted them in a more standard way approximating them with e^{sT} instead, and the building a new characteristic equation from the poles given.

$$(z - e^{s_{p1}T})(z - e^{s_{p2}T})$$

These poles generally matched the theory more closely as expected without mathematical error.

Then, we learned how significant of an affect saturation can be, creating extreme non-linearities. In essence saturation limits caused a non-minimum phase zero to emerge in our yaw controller. Data such as this will be quite useful in debugging strange maneuvers with the vehicle if we ever test these systems on the actual AUV.

Finally, we learned that the vehicle is controllable across a broader range of T than originally assumed. Due to the earlier modeling errors mentioned we had initially assumed the controller was only capable of producing a controllable system at T=1/40, with some degrees of freedom requiring 100Hz sample rates in order to be controllable. After fixing these mathematical errors the model fit our initial intuition, which states that the vehicle is stable and controllable across all T. This is quite useful for the AUV since it means we can rely more heavily on our slower but more accurate sensor in order to control the vehicle, without significantly affecting the performance. This is likely due to how slow the system is overall, and more experiments with even slower T would make excellent future extensions for this project.

Other possible extensions could include attempting more aggressive rise and settling time constraints, while attempting to limit overshoot as we did throughout this project. In these cases, the faster sample times may show significantly better performance. In addition, testing the controllers developed through Simulink on the actual vehicle would be extremely interesting.

References

[1] Thor I. Fossen. *Handbook of Marine Craft Vehicle Hydrodynamics and Motion Control*. John Wiley And Sons Ltd., The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom, 2011.

[2] Kyle Harlow. ECEN 5458 Project 2 Summary, 2018.

[3] Kyle Harlow. ECEN 5458 Project 3 Summary, 2018.

A Additional Figures

A.1 8Hz

A.1.1 Down

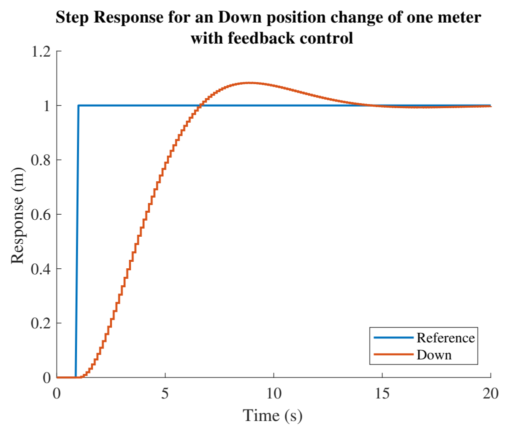


Figure 19:

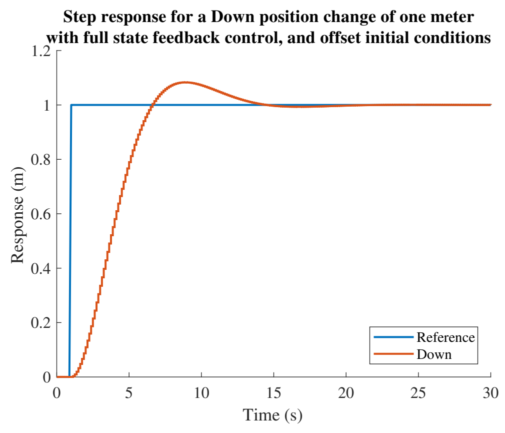


Figure 20:

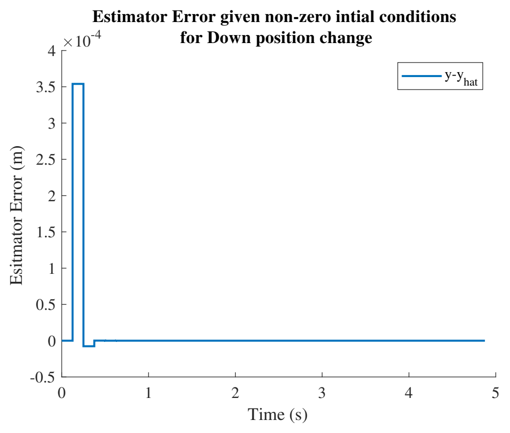


Figure 21:

A.1.2 East

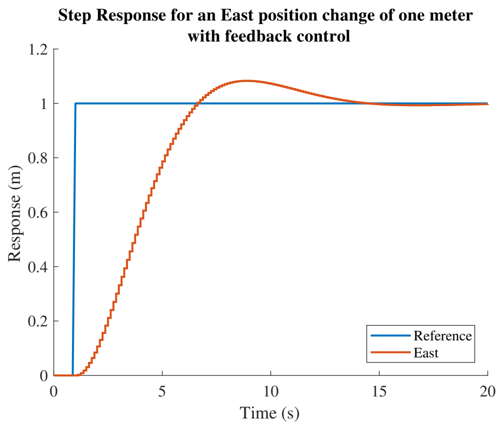


Figure 22:

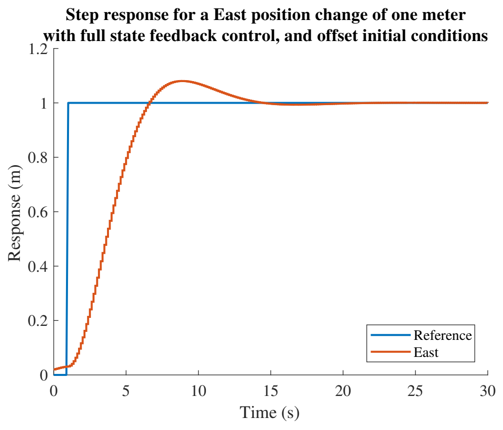


Figure 23:

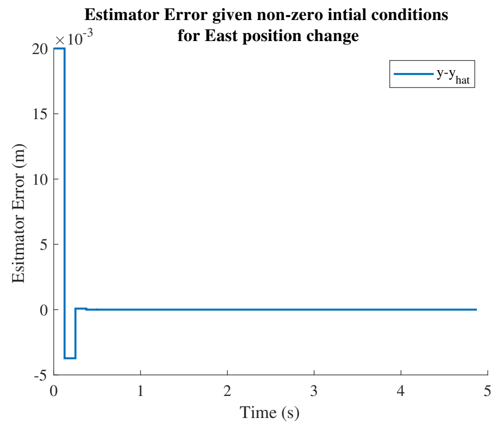


Figure 24:

A.2 40Hz

A.2.1 Down

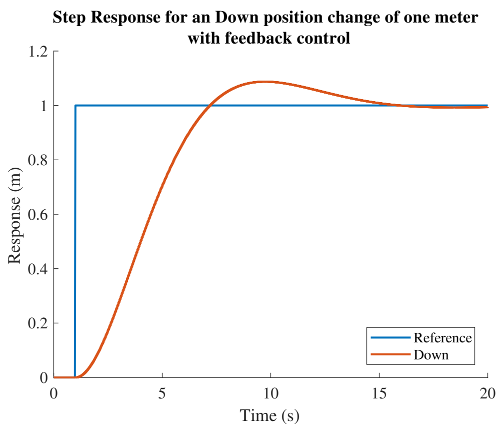


Figure 25:

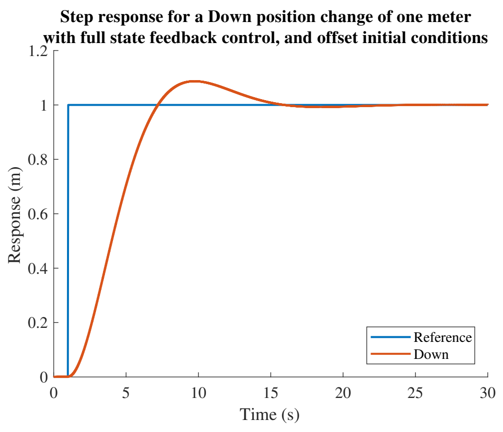


Figure 26:

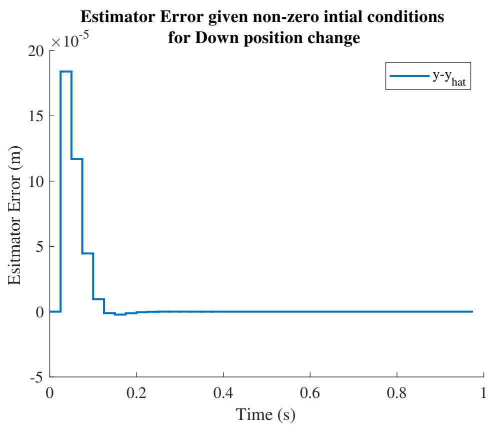


Figure 27:

A.2.2 East

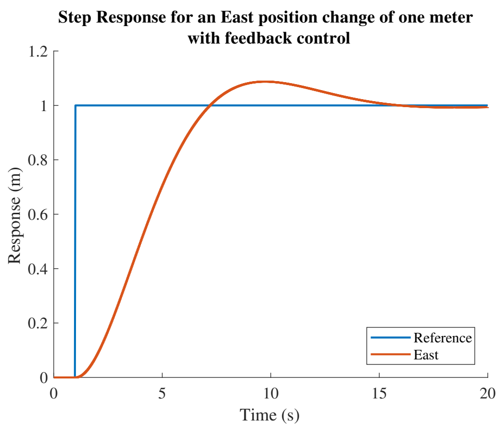


Figure 28:

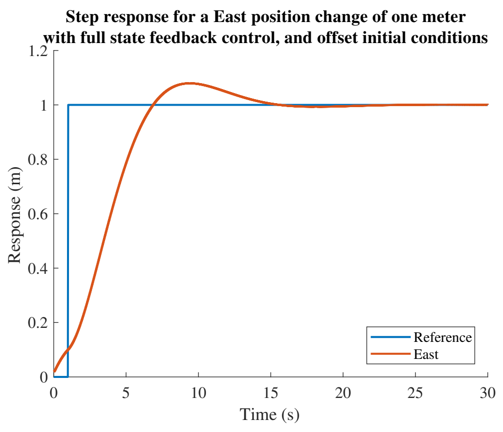


Figure 29:

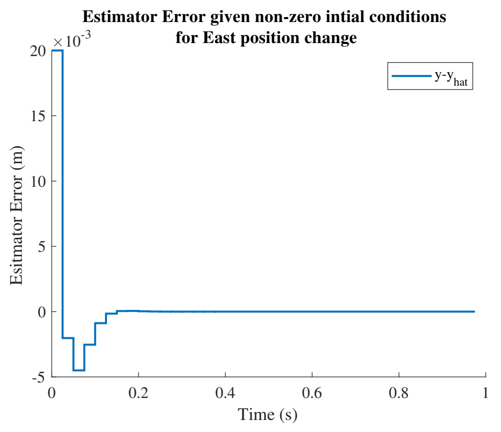


Figure 30:

B Code

```
%% Clear old variables
clear all
close all
%clc


% % Define mapping from motors to inputs
% I1 = 0.377952;
% I2 = 0.28702;
% I3 = 0.03506;
% I4 = 0.162052;
% L = [0 0 1 1 0 0 0 0;
%     1 1 0 0 0 0 0 0;
%     0 0 0 0 1 1 1 1;
%     0 0 0 0 -I4 I4 -I4 I4;
%     0 0 I3 I3 I1 I1 -I1 -I1;
%     I1 -I1 -I2 I2 0 0 0 0];


%% Set up vehicle properties
% Center of Mass (Cg)
Cg = [0.5392,0.2286,0.2789]';
% Co Origin
Co = Cg;
rg = Co-Cg;


% Center of Bouyancy (Cb)
Cb = [0.5392,0.2286,0.3906]';


% Vehicle mass
m = 25.8618; %kg

% Vehicle Inertia from Cg
I = [0.7628 0.0008 0.0025;
    0.0008 1.1921 0.0068;
    0.0025 0.0068 1.1852];

Ix = 0.7628;
Iy = 1.1921;
Iz = 1.1852;
% % Mass Added (estimate from fluid friction force (SolidWorks Flow) 1m/s
% % velocity)
% Ma_x = 0.0702;
% Ma_y = 0.731;
% Ma_z = 0.463;
% % Defining rotaitonal mass to be the same "faces" as their corresponding
% % linear masses. I.E. yaw flow is over "y" plane (probably not the best
% % approximation, experimental data needed).
% Ma_r = 0.731;
% Ma_p = 0.463;
% Ma_q = 0.731;

Ma_test = -diag([0.4*m, .75*m, .9*m, .9*Ix, .9*Iy, .8*Iz]);

% Define "Moment" matrix
% Rigid Body Matrix
Mb = [m*eye(3,3), zeros(3,3);
    zeros(3,3),diag(diag(I))];
% Ma = -diag([Ma_x, Ma_y, Ma_z, Ma_r, Ma_p, Ma_q]);
% Mass Added Matrix
```

```
M = Mb+Ma_test;



% Define Gravity Matrix
g = 9.81; %m/s^2
% Difference between Cg and Cb
pq_bouyancy = Cg(3)-Cb(3);
% Vehicle weight in Newtons
W = g*m;
% Vehicle bouyancy in Newtons
B = -g*(32);
% Gravity Matrix
G = -diag([0,0,B+W,pq_bouyancy*W,pq_bouyancy*W,0]);

% Define Drag Matrix

% Linear drags (estimated from fluide force (SolidWorks Flow) 1m/s
% velocity)
Force_x = -27.569;
Force_y = -67.911;
Force_z = -66.522;

% Defining rotaitonal drags to be the same "faces" as their corresponding
% linear drags. I.E. yaw flow is over "y" plane (probably not the best
% approximation, experimental data needed).
Force_r = -67.911;
Force_p = -66.522;
Force_q = -67.911;

% Drag Matrix
D = -diag([Force_x, Force_y, Force_z, Force_r, Force_p, Force_q]);


%% Set up Continuous Model
% Model defined with all 12 dof

% Model dropped as it is uncontrollable, and unobservable

% Define State Space Model (Note C is an I since we want to see every state)
% A = [zeros(6,6), eye(6,6); -inv(M)*G, -inv(M)*D];
% B = [zeros(6,6), inv(M)]';
% C = [eye(12,12)];
% D = [zeros(12,6)];
%
% model = ss(A,B,C,D);
% Co = ctrb(model);
% Ob = obsv(model);
% unco = length(A)-rank(Co);
% unob = length(A)-rank(Ob);
% stabilizable = rank([eig(A).*eye(12,12) B]);
% detectable = rank([eig(A).*eye(12,12); C]);

%% Set up Seperable SISO Models
% Model defined with 6x2dof siso models
unco = zeros(6,1);
unob = zeros(6,1);
A_SISO = cell(6,1);
B_SISO = cell(6,1);
C_SISO = cell(6,1);
D_SISO = cell(6,1);

Model_SISO = cell(6,1);
```

```
P = zeros(2,6);

for i = 1:6
    A_SISO{i} = [0, 1; -G(i,i)*1/(M(i,i)), -D(i,i)*1/(M(i,i))];
    B_SISO{i} = [0; 1/(M(i,i))];
    % Desired output is position
    C_SISO{i} = [1 0];
    D_SISO{i} = zeros(1);
    Model_SISO{i} = ss(A_SISO{i},B_SISO{i},C_SISO{i},D_SISO{i});
    % Check models for controllability and observability
    unco(i) = length(A_SISO{i})-rank(ctrb(Model_SISO{i}));
    unob(i) = length(A_SISO{i})-rank(obsv(Model_SISO{i}));
    % Check poles of each model
    P(:,i) = pole(Model_SISO{i});
end


%% Define Descrete Models based on T
% Define Time Delay
T = 1/8; % DVL Update in Seconds
% T = 1/100
% T = 1/40; % State Estimator Update in Seconds

D_Model_SISO = cell(6,1);
unco_D = zeros(6,1);
unob_D = zeros(6,1);

P_D = zeros(2,6);

for i = 1:6
    D_Model_SISO{i} = c2d(Model_SISO{i},T);
    % Check discrete models for controllability and observability
    CoD = ctrb(D_Model_SISO{i});
    ObD = obsv(D_Model_SISO{i});
    unco_D(i) = length(A_SISO{i})-rank(CoD);
    unob_D(i) = length(A_SISO{i})-rank(ObD);
    % Check poles of each model
    P_D(:,i) = pole(D_Model_SISO{i});
    % If the model is uncontrollable or unobservable check if it is
    % stabilizeable or detectable
    stabilizable(i) = rank([eig(D_Model_SISO{i}.A).*eye(2,2) D_Model_SISO{i}.B]);
    detectable(i) = rank([eig(D_Model_SISO{i}.A).*eye(2,2); D_Model_SISO{i}.C]);
end

%% Define Control Gain K
% Set desired poles given omega_n = 0.45, zeta = 0.6
if(T == 1/40)
    p_test = [0.993-0.00894i, 0.993+0.00894i];
elseif(T == 1/8)
    p_test = [0.96-0.048i, 0.96+0.048i];
end
% K_10 = cell(3,1);
% K_20 = cell(3,1);
% K_40 = cell(3,1);
K_test = cell(6,1);
N = cell(6,1);

for i = 1:6
%    K_10{i} = place(D_Model_SISO{i}.A, D_Model_SISO{i}.B, p_des_overshoot10);
%    K_20{i} = place(D_Model_SISO{i}.A, D_Model_SISO{i}.B, p_des_overshoot20);
%    K_40{i} = place(D_Model_SISO{i}.A, D_Model_SISO{i}.B, p_des_overshoot40);
   K_test{i} = place(D_Model_SISO{i}.A, D_Model_SISO{i}.B, p_test);
   N_vec = inv([D_Model_SISO{i}.A-eye(size(D_Model_SISO{i}.A)), ...
   D_Model_SISO{i}.B; D_Model_SISO{i}.C, [0]])*[0;0;1];
   N{i} = N_vec(3)+K_test{i}*N_vec(1:2);
end

% K_test{6} = place(D_Model_SISO{6}.A, D_Model_SISO{6}.B, p_test);
% N_vec = inv([D_Model_SISO{6}.A-eye(size(D_Model_SISO{6}.A)), ...
% D_Model_SISO{6}.B; D_Model_SISO{6}.C, [0]])*[0;0;1];
% N{6} = N_vec(3)+K_test{6}*N_vec(1:2);

%% Simulate step response to gain K
IC = [0;0];
sim_time = 20;
sim('X_Model.slx')
sim('Y_Model.slx')
sim('Z_Model.slx')
sim('Yaw_Model.slx')

step_information{1} = stepinfo(y_1,t_1,1);
step_information{2} = stepinfo(y_2,t_2,1);
step_information{3} = stepinfo(y_3,t_3,1);
step_information{6} = stepinfo(y_6,t_6,pi/2);


%% Plot Step Response with Gain K
figure
hold on
plot(t_1, r_1,'Linewidth', 1.5)
stairs(t_1,y_1,'Linewidth', 1.5)
legend('Reference','North','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step response for a North position change of one meter', newline,'with feedback control'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off

figure
hold on
plot(t_2, r_2,'Linewidth', 1.5)
stairs(t_2,y_2,'Linewidth', 1.5)
legend('Reference', 'East','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step Response for an East position change of one meter', newline,...
'with feedback control'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off

figure
hold on
plot(t_2, r_2,'Linewidth', 1.5)
stairs(t_3,y_3,'Linewidth', 1.5)
legend('Reference','Down','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step Response for an Down position change of one meter', newline,...
'with feedback control'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off
```

```
figure
hold on
plot(t_6, r_6,'Linewidth', 1.5)
stairs(t_6,y_6,'Linewidth', 1.5)
legend('Reference','Yaw','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (rad)')
title(['Step Response for a Yaw position change of pi/2 radian', newline,...
'with feedback control'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off


%% Define Observer Gain L

% 2x Poles
% if(T == 1/40)
%     p_test_obs = [0.986-0.0178i, 0.986+0.0178i];
% elseif(T == 1/8)
%     p_test_obs = [0.93-0.084i, 0.93+0.084i];
% end

% % 10x ples (damping ratio 1%, tr_des = .4)
% if(T == 1/40)
%     p_test_obs = [0.912-0.0616i, 0.912+0.0616i];
% elseif(T == 1/8)
%     p_test_obs = [0.601-0.211i, 0.601+0.211i];
% end



% 100x poles (damping ratio 1%, tr_des = .04)
if(T == 1/40)
    p_test_obs = [0.3174+0.245i,0.3174-0.245i];
elseif(T == 1/8)
    p_test_obs = [-0.0108+0.00257i,-0.0108-0.00257i];
end

M = cell(6,1);
L_test = cell(6,1);
for i = 1:6
    L_test{i} = place((D_Model_SISO{i}.A)', (D_Model_SISO{i}.C)', p_test_obs)';

    M{i} = D_Model_SISO{i}.B*N{i};
end


%% Simulate with Observer Gain L
IC = [0.02; 0.03];
sim_time = 30;
sim('X_Model.slx')
sim('X_Model_OBS.slx')
sim('Y_Model_OBS.slx')
sim('Z_Model_OBS.slx')
sim('Yaw_Model_OBS.slx')

step_information_obs{1} = stepinfo(y_1_obs,t_1_obs,1);
step_information_obs{2} = stepinfo(y_2_obs,t_2_obs,1);
step_information_obs{3} = stepinfo(y_3_obs,t_3_obs,1);
step_information_obs{6} = stepinfo(y_6_obs,t_6_obs,pi/2);

%% Plot Step Response North with full state feedback
```

```
figure
hold on
plot(t_1, r_1,'Linewidth', 1.5)
stairs(t_1,y_1,'Linewidth', 1.5)
legend('Reference','North','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step response for a North position change of one meter', newline,...
'with feedback control, and offset initial conditions'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off

figure
hold on
plot(t_1_obs, r_1_obs,'Linewidth', 1.5)
stairs(t_1_obs,y_1_obs,'Linewidth', 1.5)
legend('Reference','North','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step response for a North position change of one meter', newline,...
'with full state feedback control, and offset initial conditions'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off

figure
hold on
stairs(t_1_obs(1:40),y_1_obs(1:40)-y_hat_1_obs(1:40), 'Linewidth', 1.5)
legend('y-y_{hat}')
xlabel('Time (s)')
ylabel('Esitmator Error (m)')
title(['Estimator Error given non-zero intial conditions', newline,...
'for North position change'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)

%% Plot Step Response East with full state feedback
figure
hold on
plot(t_2_obs, r_2_obs,'Linewidth', 1.5)
stairs(t_2_obs,y_2_obs,'Linewidth', 1.5)
legend('Reference','East','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step response for a East position change of one meter', newline,...
'with full state feedback control, and offset initial conditions'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off

figure
hold on
stairs(t_2_obs(1:40),y_2_obs(1:40)-y_hat_2_obs(1:40), 'Linewidth', 1.5)
legend('y-y_{hat}')
xlabel('Time (s)')
ylabel('Esitmator Error (m)')
title(['Estimator Error given non-zero intial conditions', newline...
'for East position change'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)

%% Plot Step Response Down with full state feedback
```

```
figure
hold on
plot(t_3_obs, r_3_obs,'Linewidth', 1.5)
stairs(t_3_obs,y_3_obs,'Linewidth', 1.5)
legend('Reference','Down','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step response for a Down position change of one meter', newline,...
'with full state feedback control, and offset initial conditions'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off

figure
hold on
stairs(t_3_obs(1:40),y_3_obs(1:40)-y_hat_3_obs(1:40), 'Linewidth', 1.5)
legend('y-y_{hat}')
xlabel('Time (s)')
ylabel('Esitmator Error (m)')
title(['Estimator Error given non-zero intial conditions', newline,...
'for Down position change'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)

%% Plot Step Response Yaw with full state feedback
figure
hold on
plot(t_6_obs, r_6_obs,'Linewidth', 1.5)
stairs(t_6_obs,y_6_obs,'Linewidth', 1.5)
legend('Reference','Yaw','Location', 'southeast')
xlabel('Time (s)')
ylabel('Response (m)')
title(['Step response for a Yaw position change of one meter', newline,...
'with full state feedback control, and offset initial conditions'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
hold off

figure
hold on
stairs(t_6_obs(1:40),y_6_obs(1:40)-y_hat_6_obs(1:40), 'Linewidth', 1.5)
legend('y-y_{hat}')
xlabel('Time (s)')
ylabel('Esitmator Error (m)')
title(['Estimator Error given non-zero intial conditions', newline,...
'for Yaw position change'])
set(gca, 'FontName', 'Times')
set(gca, 'FontSize', 12)
```