Impedance Control for Use in Autonomous Decommissioning

Lucille Hosford, Chris Welch, Brian Wilcox, Manjinder Singh
Department of Mechanical Engineering, Massachusetts Institute of Technology
2.151 Advanced System Dynamics & Control
Fall 2015

Abstract

The goal of this project is to determine the feasibility of achieving desirable endpoint impedances to promote stable and robust interactions from a free-floating vehicle equipped with a backdrivable manipulator, such as in the task of autonomous decommissioning of underwater structures. Because such a vehicle does not exist yet, analyses are drawn from two similar systems that together encapsulate the desired system: a fixed-base anthropomorphic robot with a redundant backdrivable manipulator (Baxter), and a free-floating raft with a non-backdrivable manipulator (Dexter), both constrained to planar motion. Dexter affords the ability to investigate contact tasks from a free-flight vehicle, whereas Baxter's manipulator more realistically exemplifies the desired dexterity of a light intervention autonomous underwater vehicle. Through experimentation, Dexter provides a concrete example of achievable manipulator impedance characteristics for a desirable interaction response, creating the performance parameters that are then matched by Baxter's more complex manipulator. Physical experimentation is complemented by simulations of the two impedance systems, using the physical parameters determined for both Dexter and Baxter.

I. Introduction

The Gulf of Mexico has become a hotspot for offshore production in the oil and gas industry, with approximately 3,500 offshore facilities in the Gulf of Mexico, contributing 17% of the total crude oil production for the United States. Dozens of these offshore structures in the Gulf of Mexico reach the end of their operational lives every year, and decommissioning, i.e., dismantling and removing, these structures in increasingly challenging environments has pushed the limits of Remotely Operated Vehicle (ROV) and diver capabilities. Decommissioning is often required for reasons such as: irreparable damage from natural events, degradation of structural integrity due to corrosion, or the lack of economic benefit to maintain production.

Decommissioning today is both expensive and life threatening. The cost of ROV missions can cost on the order of hundreds of thousands of dollars per day, and when this rate is maintained over the course of a typical decommissioning time frame of several months, the costs become rather steep. Additionally, by the nature of the work,

there is no guarantee of having an ROV-friendly work environment, as unforeseen obstructions could potentially sever the tether. More importantly, near death experiences are not uncommon for commercial divers. Severed oxygen umbilical cords and high-pressure water jet injuries are just a couple of many ways a routine commercial dive can go wrong. In some cases, such as arc-oxygen torch explosions, these accidents can be fatal. As decommissioning environments become more severe over time, the industry is beginning to turn more towards autonomy to conduct these projects as cost-effectively and safely as possible, namely by removing ROVs and divers from the process. However, in order to do so, AUVs must demonstrate their ability to sense and manipulate their environment at least as well as the ROVs and divers that they are replacing.

When it comes to the complex tasks pertaining to the manipulation aspects of decommissioning, some of the most crucial goals that must be completed revolve around: securing a vehicle to a biofouled surface (if even possible), such as flat, cylindrical, and saddle-shaped geometries (i.e. junction of cylindrical members), removing biofouling from complex geometries, which may involve potential interactions with complex geometry, and engaging in non-destructive testing (NDT), and evaluating the functionality of valves and levers. Many of these tasks strongly suggest the use of impedance control, and this report details the feasibility of achieving desirable endpoint impedances to promote stable and robust interactions using a free-floating vehicle equipped with a backdrivable manipulator.

II. System Model

Together, Baxter (a fixed-base, backdrivable manipulator) and Dexter (a floating-base, non-backdrivable manipulator) encapsulate the characteristics of an idealized light intervention autonomous underwater vehicle. Whereas Dexter affords the ability to investigate contact tasks from free-flight, Baxter's manipulator more realistically exemplifies the desired dexterity of a manipulator to be used for the plethora of tasks involved in decommissioning. The following section provides system models for both Baxter and Dexter, and sets the framework for the analysis of the key components involved in developing a free-floating AUV with a backdrivable manipulator for impedance-controlled decommissioning tasks.

Open loop model of Dexter

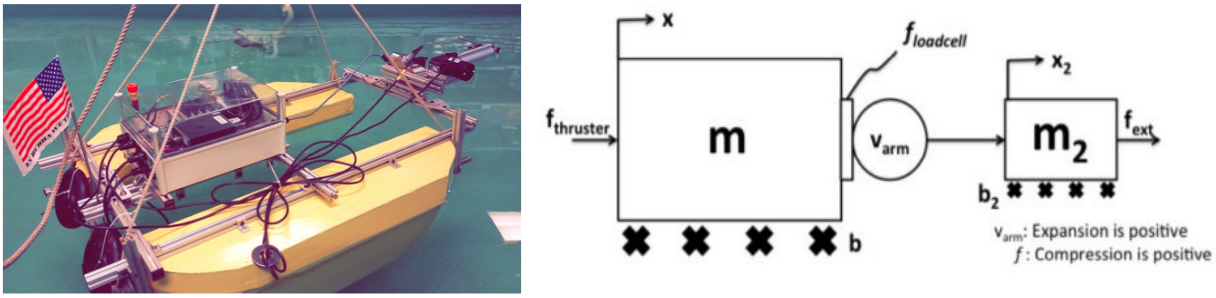


Figure 1: Photo of Dexter (left) and its complementary system model (right).

The dynamics of the open loop Dexter plant can be written as :

$$m_1\ddot{x_1} = f_t - f_l - b_1\dot{x_1}$$
$$m_2\ddot{x_2} = f_l + f_e - b_2\dot{x_2}$$
$$x_2 = x_1 + x_{arm}$$

- m1: mass of the vehicle
- m2: mass at the end of the linear actuator
- x1: linear position of vehicle from reference
- x2: position of end of linear actuator from vehicle
- b1 and b2 damping
- ft: thruster force
- fl: Force coupling the dynamics of the two masses

All together, the state space representation of the system given the aforementioned assumptions can be found below:

$$\begin{bmatrix} \frac{d(x1)}{dt} \\ \frac{d^2(x1)}{dt^2} \\ \frac{d(x2)}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \frac{-(b1+b2)}{m1+m2} & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} x1 \\ \frac{d(x1)}{dt} \\ x2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \frac{-b2}{m1+m2} & \frac{-m2}{m1+m2} & \frac{1}{m1+m2} \\ 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \frac{d(x2)}{dt} \\ \frac{d^2(x2)}{dt^2} \\ \mathrm{Ft} \end{bmatrix}$$

C=[1 0 1]; D =[0] Y = the position of the end actuator. The system is both linear and controllable. However, it is not fully observable.

Due to the fact there is no mass attached to the linear actuator, and the frequency response of the linear actuator seemed to show little effect by its own mass or damping, m2 and b2 are both assumed to be zero. Values for the m1 and b1 were determined by analyzing the raft's position response to a step input of thruster force, which resulted in values of 50 kg and 35 kg/s, respectively. Step response plots are shown below.

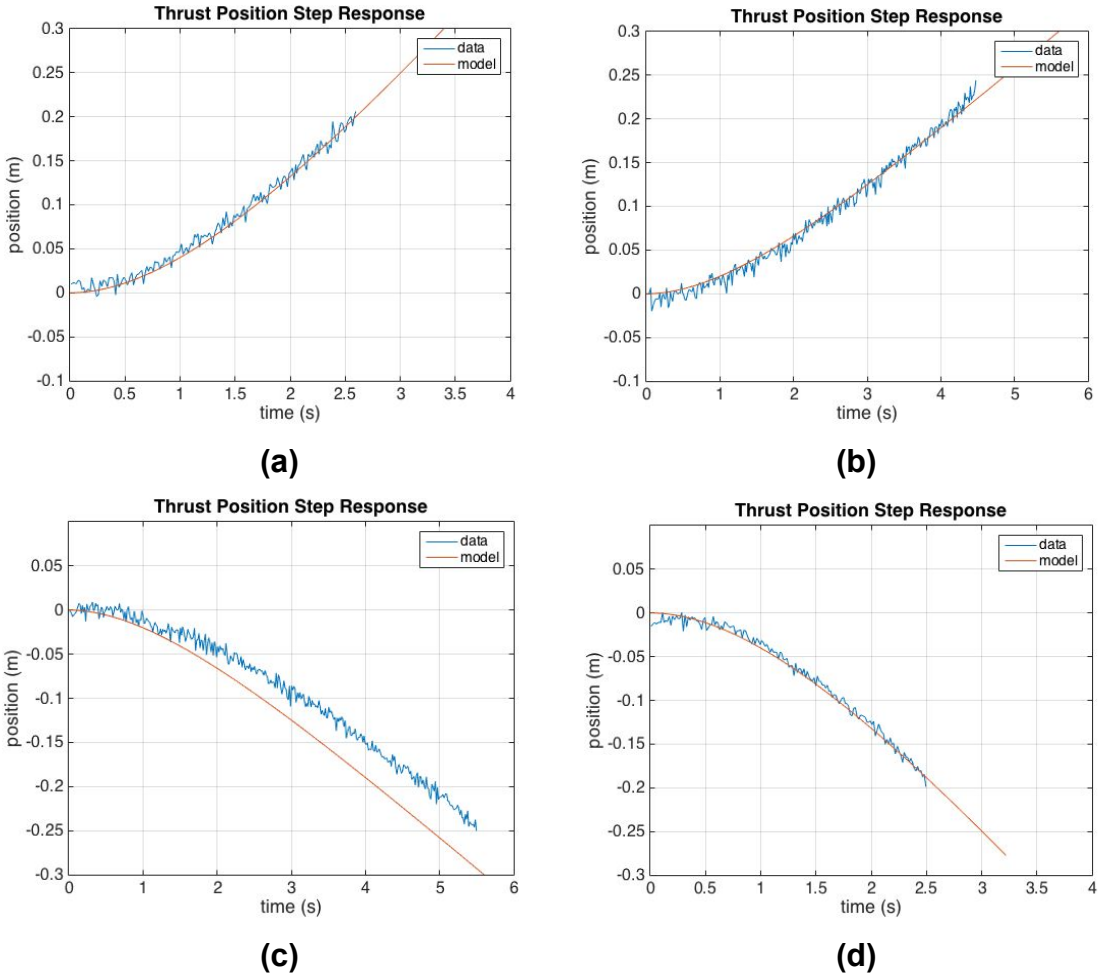


Figure 2: Plots a-d show the vehicle's response in position to force step inputs of -5 N (a), -2.5 N (b), 2.5 N (c), and 5 N (d), compared against the modeled step response with m1 = 50 kg and b1 = 35 kg/s. Although the data shows good agreement with the model for trials a,b, and d, the discrepancy in trial c may be due to the nonlinear friction of the apparatus that keeps the vehicle normal to the interaction surface (pictured below), which becomes most apparent at slow movements of the vehicle.

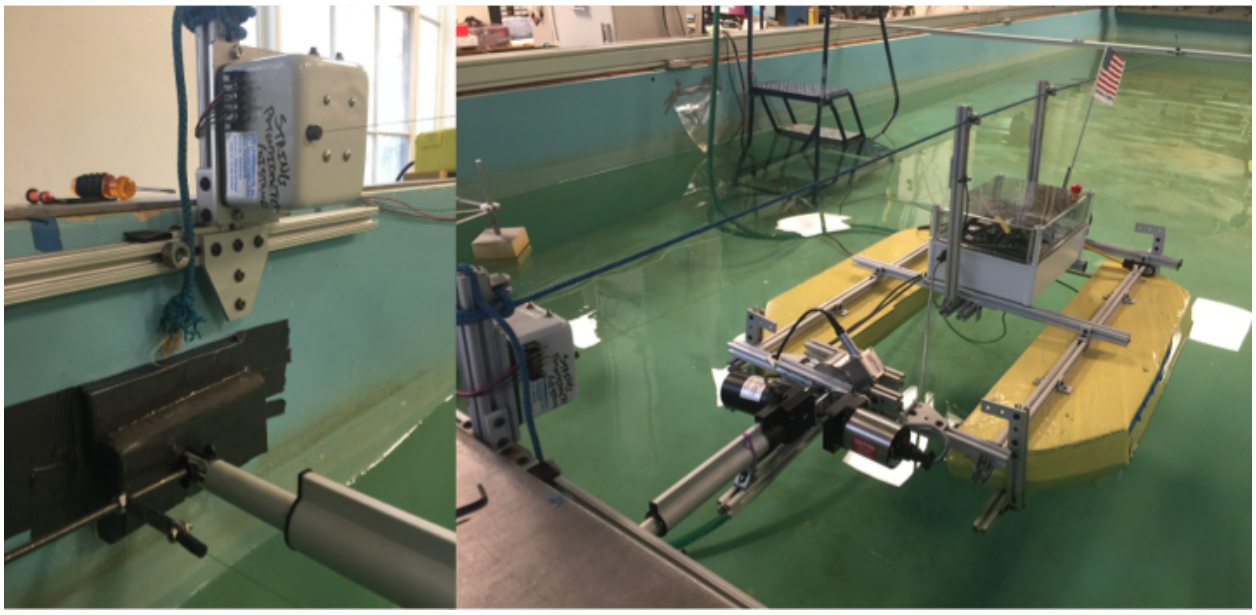


Figure 3: Dexter is kept normal to the interaction surface by sliding along a low-friction tether.

Open loop model of Baxter

The general equations of motion for any number of linkages which make up a robotic arm as seen in joint space are

$$(M(q) + I_r)\ddot{q} + C(\dot{q}, q) + D\dot{q} + g(q) = \tau_{motors}$$

M is the joint space inertia matrix, Ir is the reflected inertia of the motors, C is the Coriolis force, D is any friction in the joints, g is the force due to gravity, and tau is the torque from the motors. Given that we are constraining the robot to small, planar motions about an operating point, neglect C and g. The equation simplifies to

$$(M(q) + I_r)\ddot{q} + D\dot{q} = \tau_{motors}$$

In state space form, this may be written as

$$\frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & -(M(q) + I_r)^{-1}D \end{bmatrix}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ (M(q) + I_r)^{-1} \end{bmatrix}\tau_{motors}$$

$$\dot{x} = [0\ J(q)]\begin{bmatrix} q \\ \dot{q} \end{bmatrix} + [0]\tau_{motors}$$

The joint space inertia matrix, M(q), and Jacobian, J(q), were both taken from Baxter's URDF. The reflected inertia of the motors, Ir, were determined experimentally. As we are operating about an equilibrium point, M(q) and J(q) may be considered a constant. The system may be treated as linear. This system is fully controllable but not fully

observable. Due to the fact that the modes are stable, the unobservability of the system will not be a concern.

A. Schematic of physical system and controller

Dexter

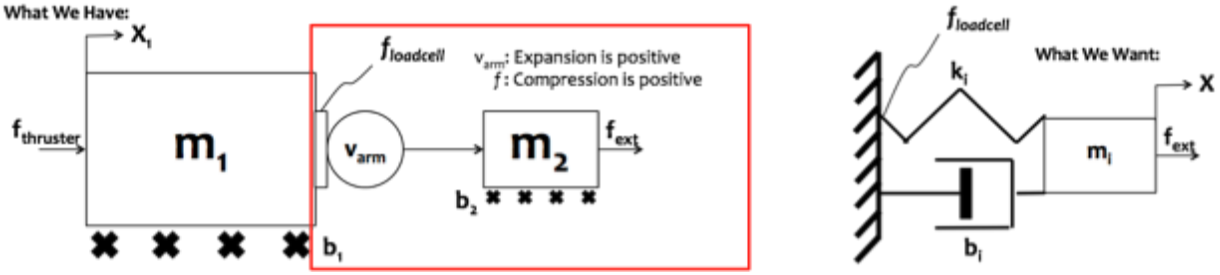


Figure 4: Model of physical system (left) and system with desired impedance characteristics (right).

For the sake of this project, Dexter's motion were constrained to one dimension. Inputs to the system include force commands and velocity commands to the ROV thrusters and linear actuator, respectively. Outputs from the system include the force reading from the load cell and the absolute position reading of the raft and the position reading of the linear actuator's end effector relative to the raft, both determined from string potentiometers.

Baxter

Figure 5: Baxter Research Robot constrained to planar motion.

For the purposes of this project, the Baxter Research Robot was constrained to planar motion by only allowing joints S1, E1, and W1 (circled) to move. The inputs to the system are torques to each of these three joints. The outputs are the endpoint position and velocity in the x direction.

B. Practical Implications

Dexter

Force between the raft and the linear actuator is measured using the ATI SI-660-60 Force/Torque Sensor, with the front end attached to the base of the linear actuator, and the back end fixed to the stationary raft. This load cell is capable of measuring forces/torques in six degrees of freedom, but only forces in only one dimension (axially) were taken into account. The ATI SI-660-60 Force/Torque Sensor features a maximum axial load of 1980 N, resolution of 0.25 N, and a bandwidth of at least 200 Hz.

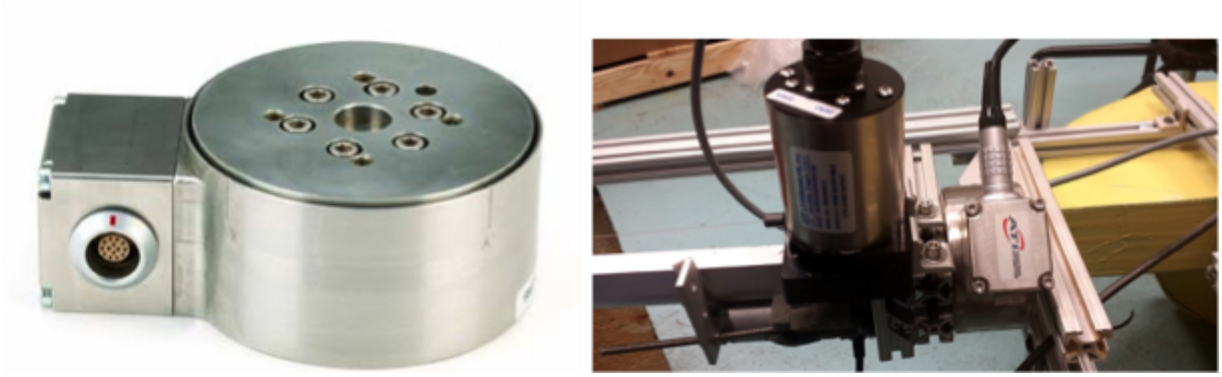


Figure 6: ATI SI-660-60 Force/Torque Sensor

Unimeasure's HX-P510 String Potentiometers are incorporated into the experimental setup in order to provide absolute position information of the vehicle, and also to compare the end effectors actual position and velocity against the model's predictions. It features a 200 cm stroke length, 5-10 VDC output range, and a wire tension of 5.8 N. Additionally, the incorporation of this string potentiometer provides the opportunity to provide state feedback to the system model.

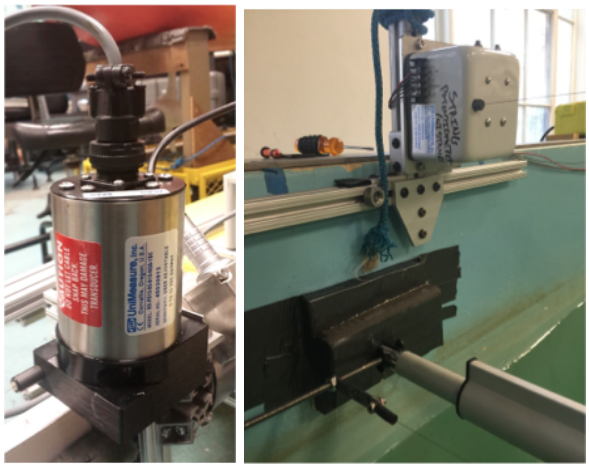


Figure 7: Unimeasure HX-P510 String Potentiometer

Firgelli Automations Light Duty Rod Actuator has an 8” stroke length, stroke speed of 3 cm/s, and a 200 lbs maximum dynamic force. It was chosen for its ability to simplify the analysis to solely one dimension (axially), for its non-backdrivability, and for its actuation speeds comparable to those of many underwater manipulators used in the offshore industry. The design of the linear actuator simply features a DC motor, a gear reduction, and a worm drive.

Figure 8: Dissection of the linear actuator shows the simple components that comprise the mechanism.

The linear actuator that moves the second mass expresses characteristics that allow it to be considered an ideal velocity source with pure time delay for the frequencies of interest. This assumption was made by first linearizing the actuator speed output with respect to a motor controller power command input, as shown below:

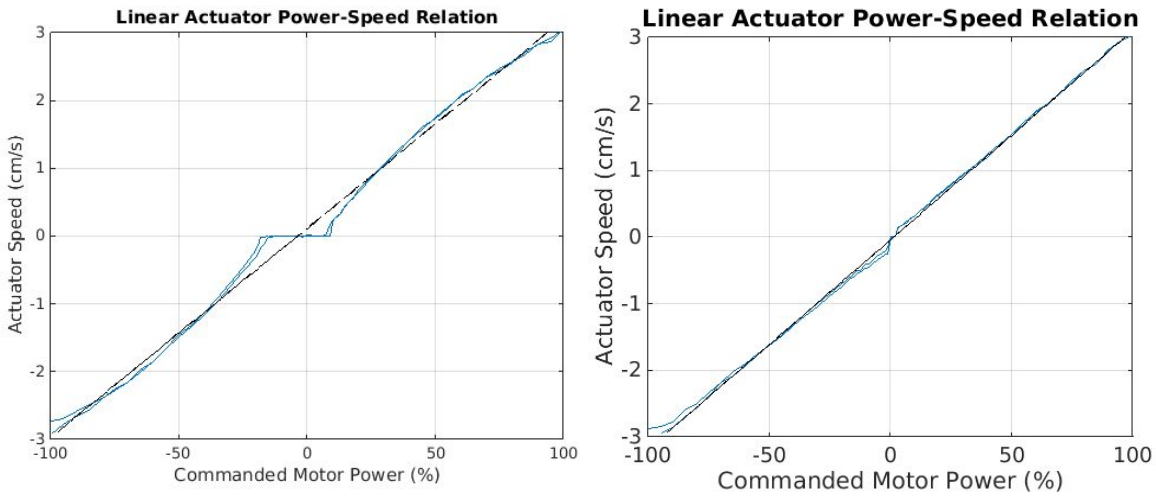


Figure 9: At low commanded powers, an asymmetric deadband exists that introduces a nonlinearity. Additionally, commanded power exhibits a non-linear relationship with respect to speed. By compensating for the mappable nonlinearities that were present in the actuator's speed response, the nonlinearities are markedly reduced.

With the actuator speed output response to commanded power input linearized, the following Bode plot was developed, indicating approximately unity gain up to 10 radians per second, with a time delay of approximately 0.135 seconds:

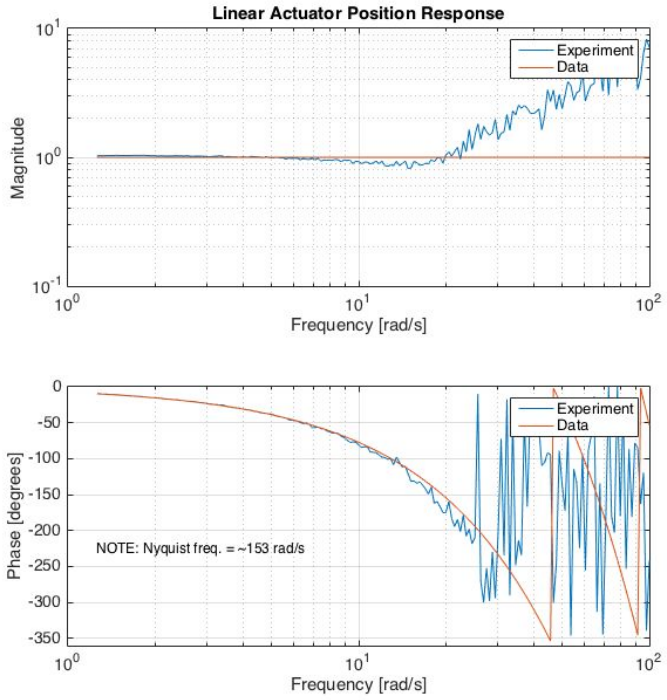


Figure 10: Frequency response of the linear actuator from a sinusoidal velocity command with an amplitude of 3 cm/s. The input motor power commands sent to the controller are derived from

feeding the desired velocity commands through the mapped nonlinearity corrections, which ensures an approximately linear mapping between commanded motor power and actuator speed. This means that the actual power command sent to the motor controller is not that of a sinusoid, but rather a corrected value to produce a sinusoidal response in actuator speed.

The Inuktun SM3300-4 ROV Thrusters that are mounted on the raft are rated for 150 watts at 24VDC. Since the configuration of these thrusters is asymmetric, a maximum of 59 N of force is provided in the forward direction, while a maximum of 11 N of force is provided in the reverse direction. The design of the thruster is simply comprised of a ducted propeller attached to the shaft of a brushed DC motor in a waterproof housing.

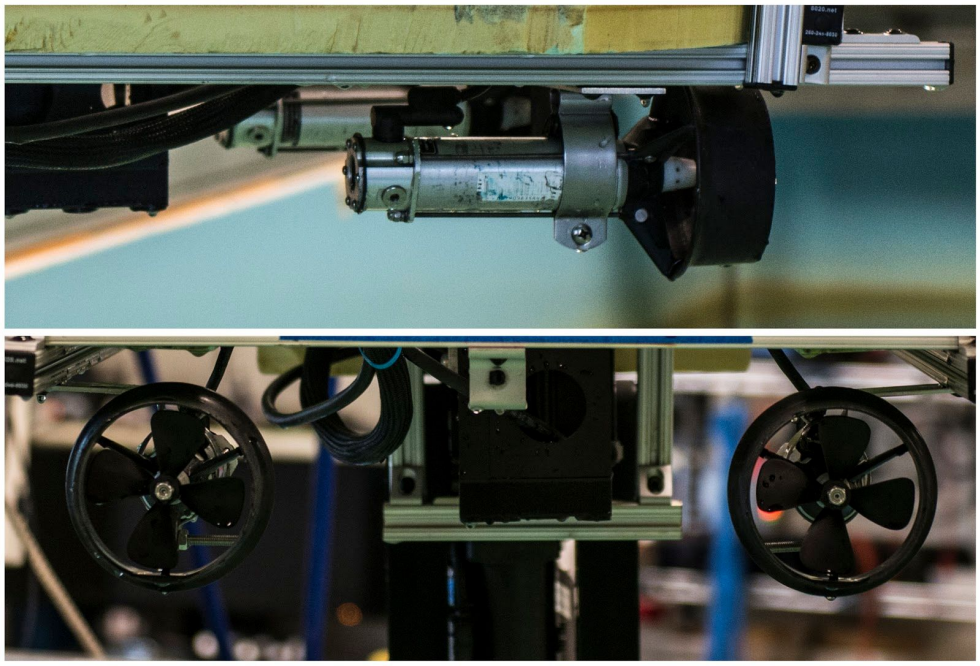


Figure 11: A profile view (top) and rear view (bottom) of the thrusters show two Inuktun SM3300-4 ROV Thrusters mounted to the aluminum frame on the bottom of the raft.

The ROV thrusters were treated as an ideal force source, and this assumption was validated through a similar process as the linear actuator. First, the force output response was linearized with respect to a motor controller command input, and then a bode plot was created, indicating a magnitude response of approximately unity up until about 10 radians per second, with a time delay of 0.075 seconds.

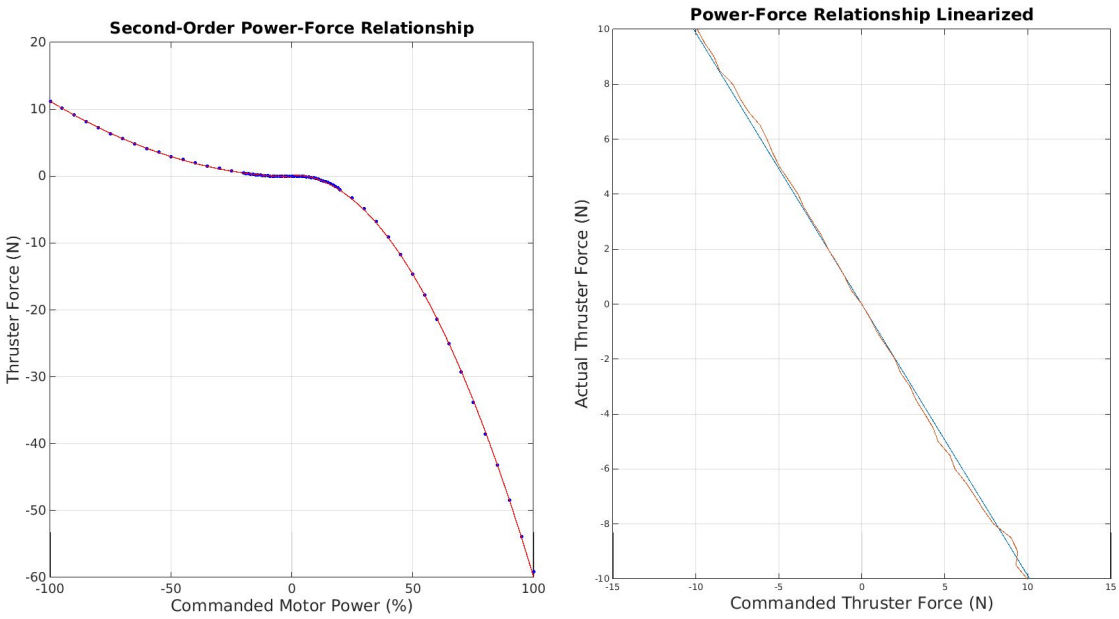


Figure 12: Fitting the static thruster data shows a typical square relationship between power and thrust. Since the thrust response to input power proved to be repeatable, it allowed for a corrective mapping of its non-linear response.

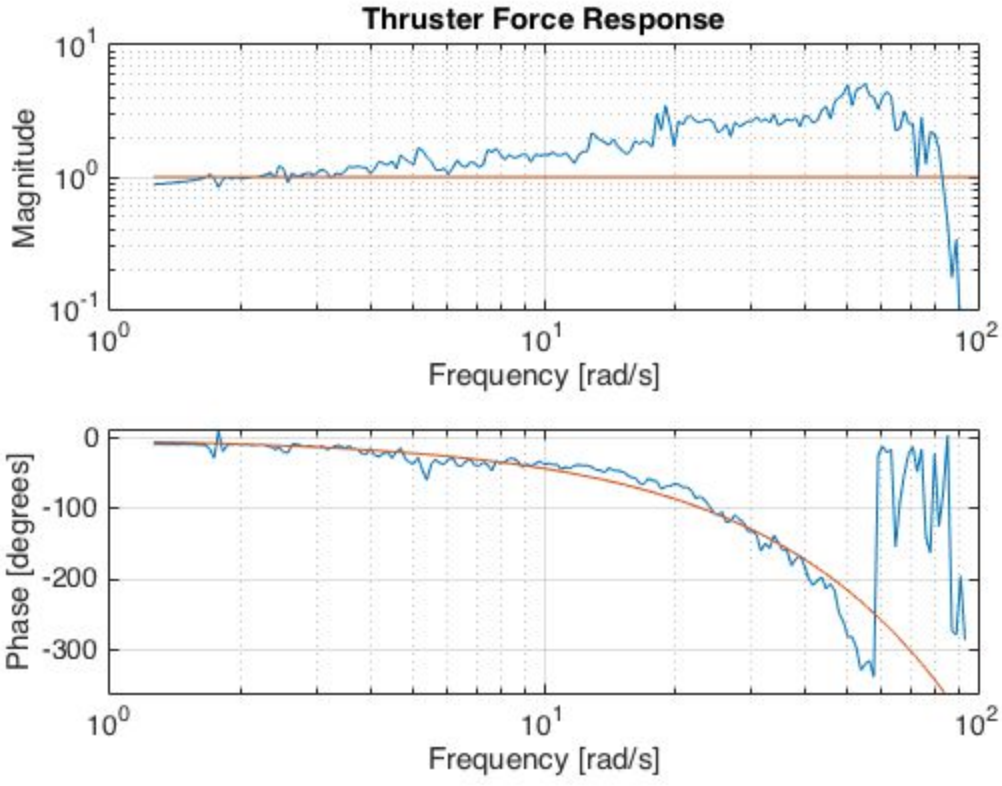


Figure 13: Similar to the linear actuators, the thrusters exhibit time delay characteristics. However, irregular traits, such as a steady rise in magnitude given a steady increase in phase lag, indicate

the nonlinear dynamics in the thrusters. However, similar to the linear actuator, these nonlinearities are ignored due to desired operating frequencies on the order of 1 Hz.

The linear actuator and ROV thrusters are controlled by the RoboteQ SDC2130 motor controller, a brushed DC motor controller that also comes equipped with 0-5V analog inputs, one of which is used to transmit the analog position data coming from the string potentiometer that reports the linear actuator's position.

Figure 14: The RoboteQ SDC2130 Motor Controller acts as a hub through which motor commands are sent to either the linear actuator or the thrusters.

Baxter

Baxter has encoders at each joint which provide both position and velocity data. Furthermore, the Series elastic actuators (SEAs) at each joint provide force feedback. Full state feedback can be achieved without the need for observers.
The SEAs make each joint a fourth order system. However, for the purposes of this project they were assumed to act like a second order system within the operating frequency of the robot. This assumption, will be evaluated later in the report.

III. Controller Design

To determine the range of impedances for which Baxter and Dexter could match, both systems first needed an impedance controller.

General example of an impedance controller

In the open loop model, both Baxter and Dexter may be modeled as a general lumped mass with some inherent damping. The overall all goal of impedance control is to mask

that inertia and damping of the real system and replace it with a desired inertia, stiffness, and damping.

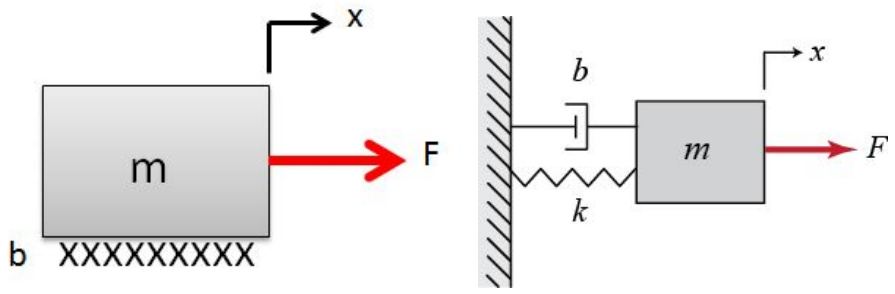


Figure 15: General concept of impedance control applied to both Baxter and Dexter

To accomplish this, you take the equations of motion of the specific system and set it equal to the desired reaction to some external force. The equation of motion for the general mass/damping system is

$$m\ddot{x} = -b\dot{x} + F_{motors} - F_{external}$$

The desired reaction to some input force is

$$M_d\ddot{x} = -B_m\dot{x} + K_d(x_d - x) - F_{external}$$

Setting the two equal to each other

$$m\ddot{x} = -b\dot{x} + F_{motors} + M_d\ddot{x} - K_d(x_d - x) + (B_m - b)\dot{x}$$

The force commanded to the motors is

$$F_{motors} = (m - M_d)\ddot{x} + K_d(x_d - x) + (b - B_m)\dot{x}$$

To implement this control scheme, the object's, position, velocity, and acceleration are required. This can either be determined by sensors on the system or through the creation of observers. For the purposes of this project, the inertia of the system was not shaped by setting the desired inertia, Md, to the actual inertia, m. This simplifies the control signal to

$$F_{motors} = K_d(x_d - x) + (b - B_m)\dot{x}$$

Ideally, the damping in our model, b, would be equal to the damping of the actual system. However, there will most likely be some small error. To differentiate between the modeled friction and the actual friction, the modeled friction will be called b_tilda from here on out. The new equation of motion the system is

$$m\ddot{x} = -b\dot{x} - F_{external} + K_d(x_d - x) + (\tilde{b} - B_m)\dot{x}$$

Written in state space, the closed loop equation of motion is

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}(B + \breve{b} - b) \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix}[Kx_d - F_{external}]$$

This overall control scheme of force based impedance control may be represented in the following diagram

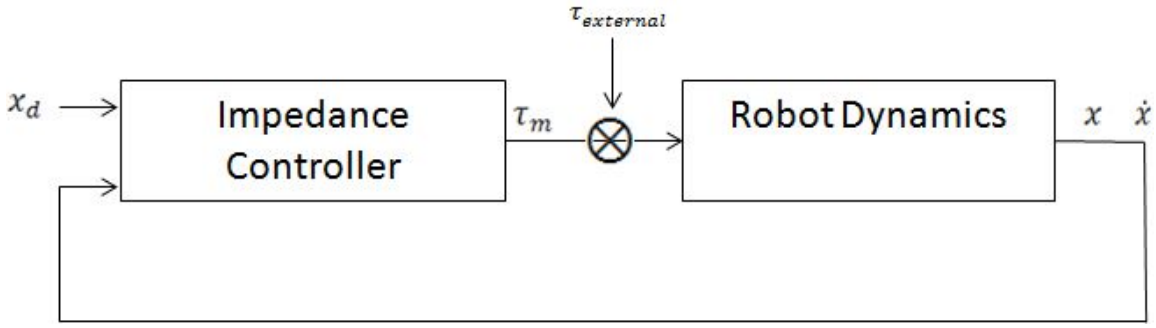


Figure 16: Simplified Diagram of a force based impedance controller

Closed Loop Control of Baxter

Expanded to the concept to the specific case of Baxter, the state space equation may be written in configuration space as

$$\frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -(M + I_r)^{-1}J^T KJ & -(M + I_r)^{-1}(\breve{D} - D + J^T BJ) \end{bmatrix}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ (M + I_r)^{-1} \end{bmatrix}[U^T KJq_d - \tau_{external}]$$

Closed Loop Control of Dexter

In order to determine the necessary impedance characteristics Baxter must emulate, a closed loop controller was designed around solely the linear actuator and its interactions with an external environment. A fairly simple model can be used to describe the non-backdrivable linear actuator. At first, a model approach was taken treating the linear actuator as a free mass, m, attached to a fixed velocity source, v_{arm}, directly attached to load cell reading force, $f_{loadcell}$ or f_l. However, as previously explained, the linear actuator demonstrated that the change in force due to the actuators acceleration was not discernible from noise, and therefore made the actuator's mass negligible. This further simplified the model by removing the free mass term and absorbing any mass associated with the base of the linear actuator into the mass of raft, or in this particular analysis, neglecting it altogether as part of the fixed base.

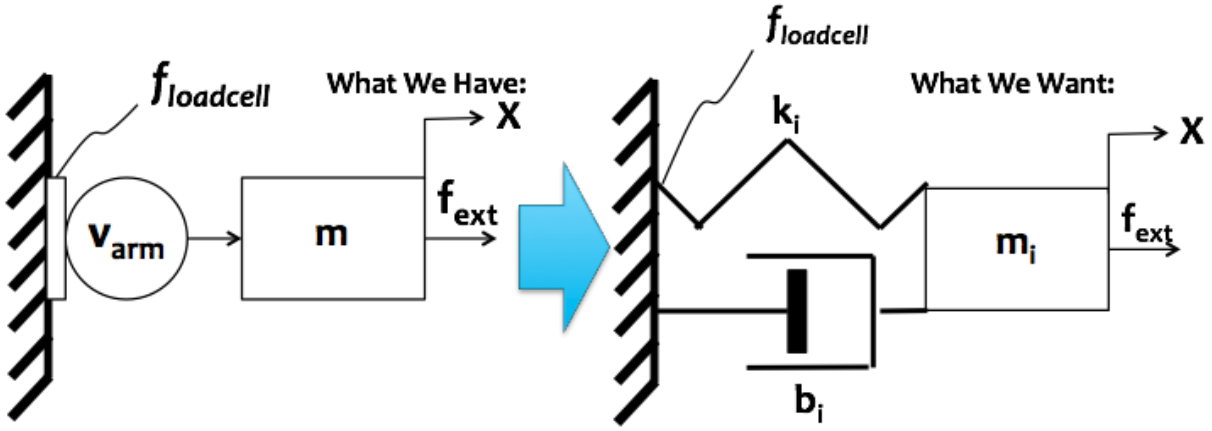


Figure 17: Model of physical system (left) and system with desired impedance characteristics (right).

By writing out the equations of motion for our actual system and out desired system, solving each for the external force, and then setting them equal to each other, the relationship between the two systems is established based on both actual and ideal parameters, which can then be implemented as the controller. The combined equations of motion are as follows:

$$m_i\ddot{x} + b_i\dot{x} + k_i x = m\ddot{x} + f_l$$

Once again, given that the linear actuator mass, m, is negligible, differential equations for the state vector [x x']' can be found:

$$\ddot{x} = -\frac{b_i}{m_i}\dot{x} - \frac{k_i}{m_i}x + \frac{1}{m_i}f_l$$

$$\dot{x} = \dot{x}$$

Finally, writing out the state equations in their entirety yield the following SISO system with its single input being load cell force and its single output being actuator velocity, thus providing a way to actuate the system that will emulate the idealized impedance system:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\frac{k_i}{m_i} & -\frac{b_i}{m_i}\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{m_i}\end{bmatrix}f_l$$

$$[\dot{x}] = [0 \quad 1]\begin{bmatrix} x \\ \dot{x} \end{bmatrix} + [0]f_l$$

Potential Inclusion of SEAs to Model of Baxter

As previously stated, the actual controller which is implemented on Baxter assumes the Series Elastic Actuators may be modeled as a second order system. This was primarily done because the Baxter Research Robot has an internal control loop operating at each joint is not known or available to users. As such it is not known how the dynamics of the actual SEAs are masked or altered by the control loop. However, it would still be valuable to at least conceptually understand both how the SEAs would affect the system, as well as how a controller could be designed for them if the need arises. To that end, a model of a single SEA was analyzed.

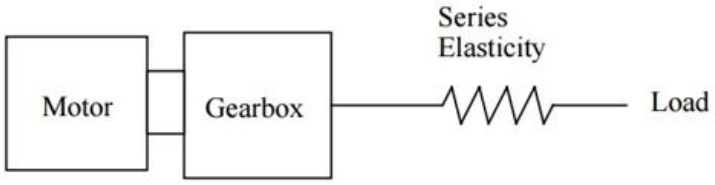


Figure 18: Diagram of a SEA [1]

The equation of motion for a single SEA may be represented in state space as

$$\frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & I & 0 & 0 \\ -M^{-1}K_{SEA} & -M^{-1}C & M^{-1}K_{SEA} & 0 \\ 0 & 0 & 0 & I \\ I_r{}^{-1}K_{SEA} & 0 & -I_r{}^{-1}K_{SEA} & -I_r{}^{-1}D \end{bmatrix}\begin{bmatrix} q \\ \dot{q} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ I_r{}^{-1} \end{bmatrix}\tau_{motor}$$

$$\dot{x} = [0\ J\ 0\ 0]\begin{bmatrix} q \\ \dot{q} \\ \theta \\ \dot{\theta} \end{bmatrix} + [0]\tau_{motor}$$

Where q is displacement on the link side and theta is displacement on the motor side. D and C are damping on the link side and the motor side respectively. The system is both fully controllable and fully observable.

The springs in the SEAs are extremely stiff, adding very little dynamics to the system. The frequency response for a single SEA is represented in the bode plot below. Because of the high stiffness of the spring, the SEA acts like a second order system up to approximately 100 rad/s. This is far above Baxter's maximum operating frequency.

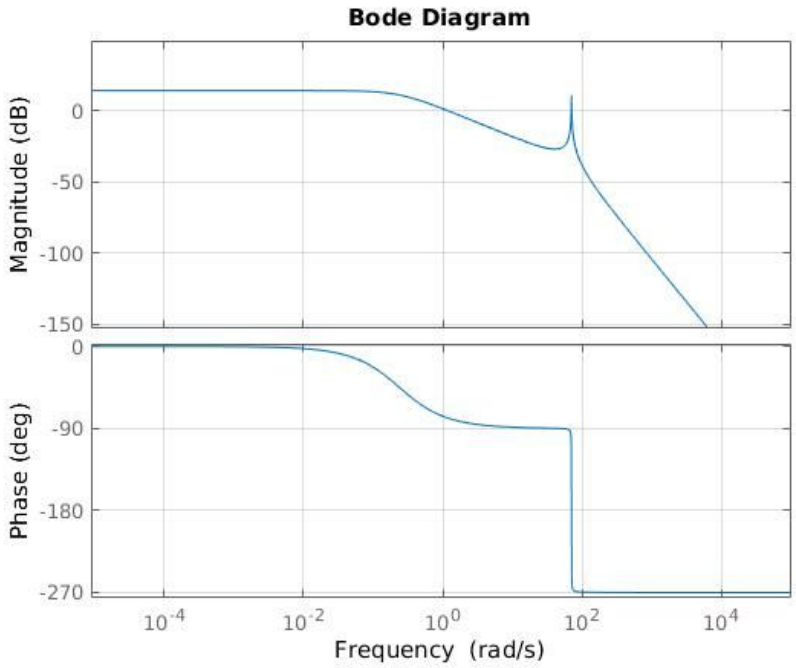


Figure 19: Bode plot for a single SEA. The second order assumption holds up to 100 rad/s

Despite the SEA's second order behavior in the frequency domain, the fact remains that there are two additional poles which could potentially be driven into the right half plane for high gains if they are not accounted for in the model. These poles could hypothetically be controlled by implementing full-state feedback. However, there is no direct access to the position and velocity of the SEA on the motor side. Observers would need to be introduced into the system.

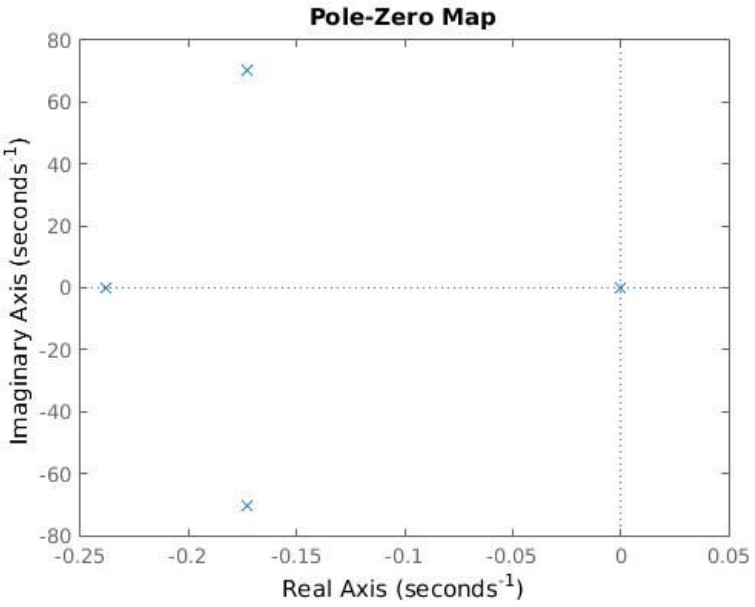


Figure 20: Open loop pole zero map of a single SEA. The real placement of these poles from the control system at the joint controller board is not known.

Observer Design for a Single SEA and Dexter

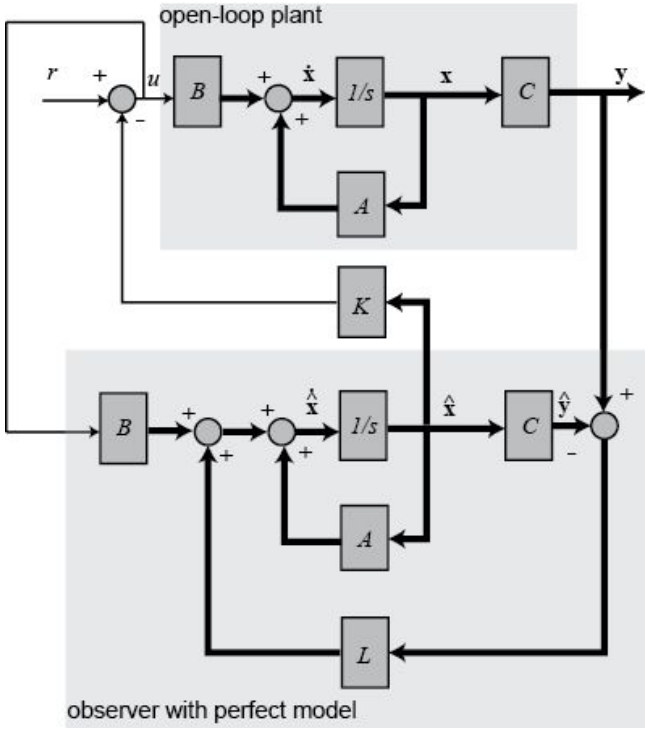


Figure 21: General outline of an observer

To add observers to the SEA Model and Dexter, full state feedback was achieved through simple LQR with **$Q = C^TC$ and $R = 1$.** This may be an example of the unknown control system.

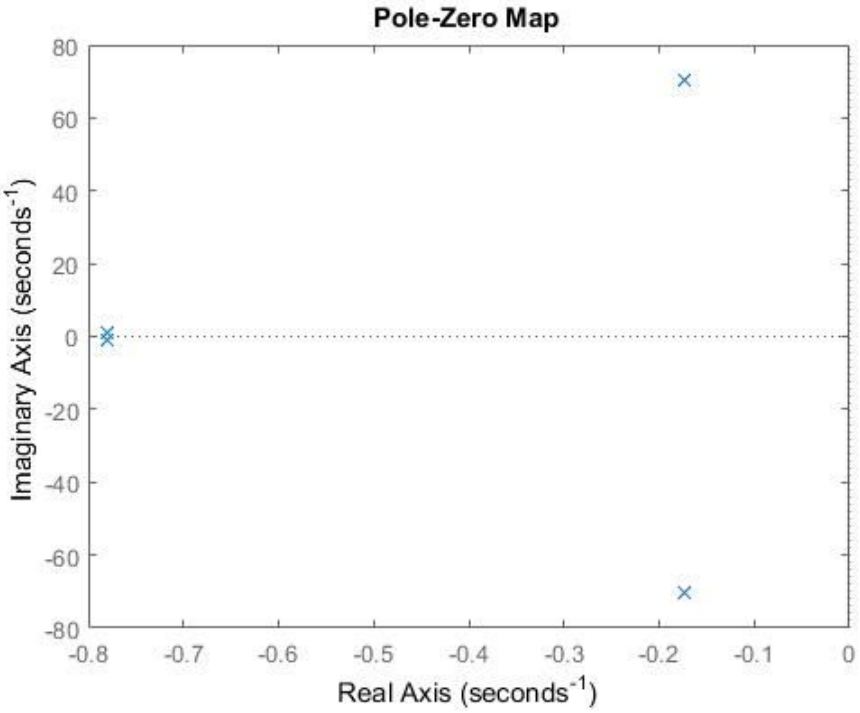


Figure 22: Pole-zero map of closed loop single SEA

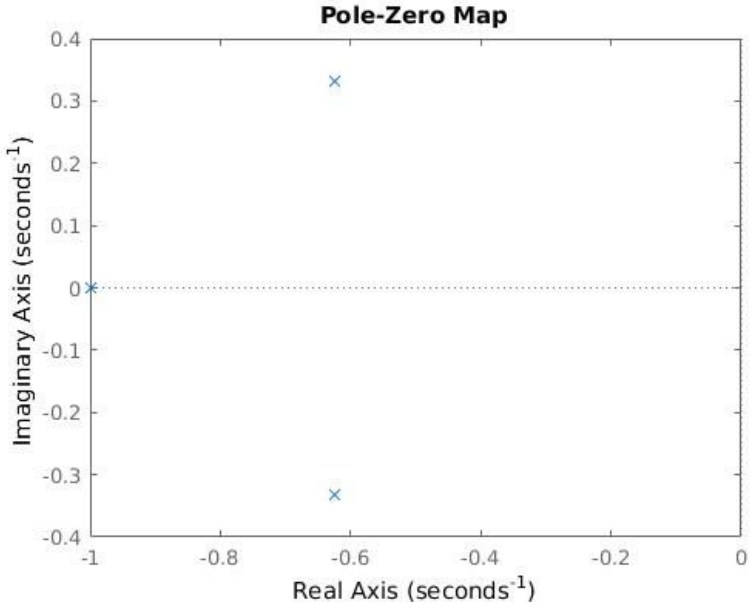


Figure 23: Pole-zero map of closed loop of Dexter

The observer gain, L, is determined such that the observer poles are 5 times faster than the dominant poles of the closed loop system. Using Matlab's *place* at for the new pole locations, the gain matrix L is determined and the new system matrix is formed with double the states.

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}\\ \hat{\mathbf{x}}\end{bmatrix}=\begin{bmatrix}\mathbf{A} & -\mathbf{BK}\\ \mathbf{LC} & \mathbf{A}-\mathbf{BK}-\mathbf{LC}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \hat{\mathbf{x}}\end{bmatrix}+\begin{bmatrix}\mathbf{B}\\ \mathbf{B}\end{bmatrix}\mathbf{r}$$

$$\mathbf{y}=\begin{bmatrix}\mathbf{C} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \hat{\mathbf{x}}\end{bmatrix}$$

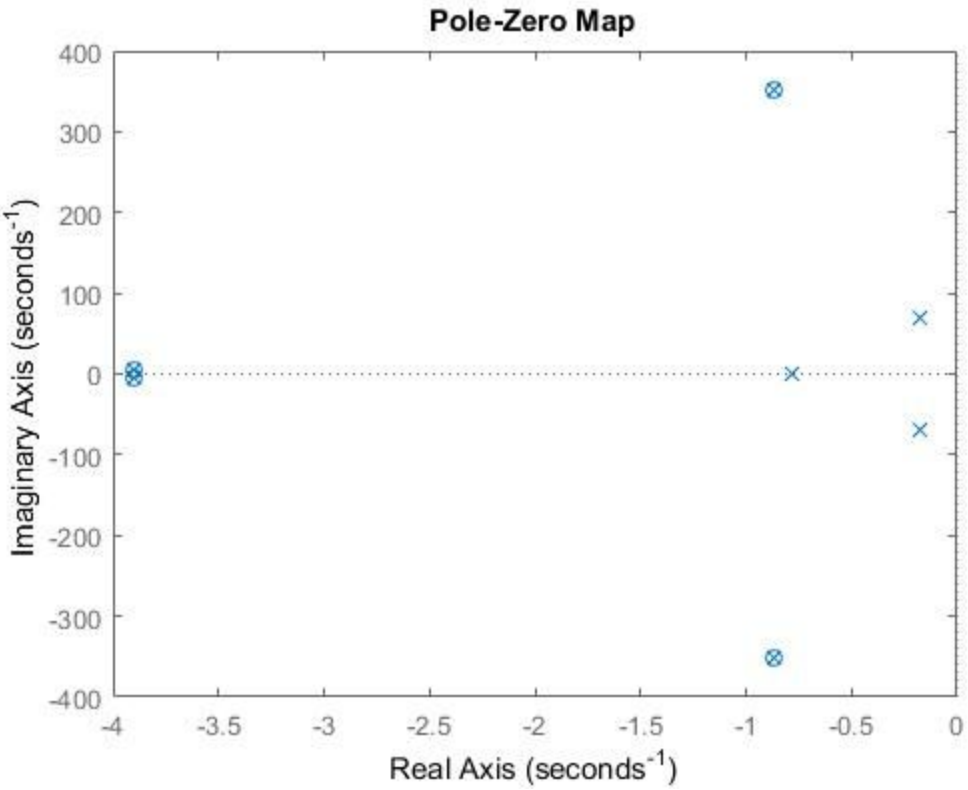


Figure 24: Pole-zero map of closed loop plus observers for single SEA

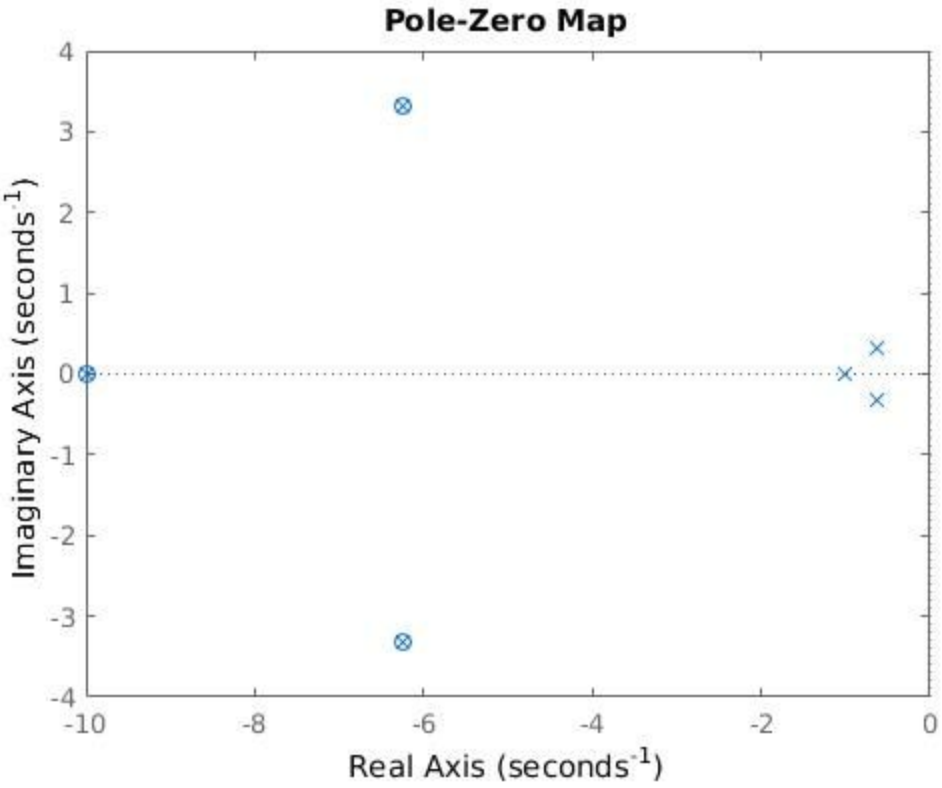


Figure 25: Pole-zero map of closed loop plus observer for Dexter

Kalman Filter Design for Dexter

Due to high amount of sensor noise present in the system, a Kalman filter was also designed. For our system, the sensor noise was obtained from physical model and actuator noise was assumed to be zero. The sensor noise was assumed to have a variation of 0.0011 for positions of both linear actuator and vehicle. The assumption of lack of actuator noise is most likely inaccurate, but no method of accurately approximating actuator noise was found.

$$\dot{x} = Ax + Bu + \Gamma w_i$$
$$y = Cx + w_o$$

Figure X: General State Space model with actuator and sensor noise.

Once the system parameters had been obtained, the optimal covariance matrix(Σ^o) was found using the Algebraic Riccati Equation:

$$0 = A\Sigma^o + \Sigma^o A^t + \Gamma W_i \Gamma^t - \Sigma^o C^t W_o^{-1} C \Sigma^o$$

The optimal covariance matrix was then used to find the optimal gains for the Kalman Filter using:

$$L^o = \Sigma^o C^t W_o^{-1}$$

The final state space model with the kalman filter looks very much like the state space model with the observer, just the system gains L replaced by L0.

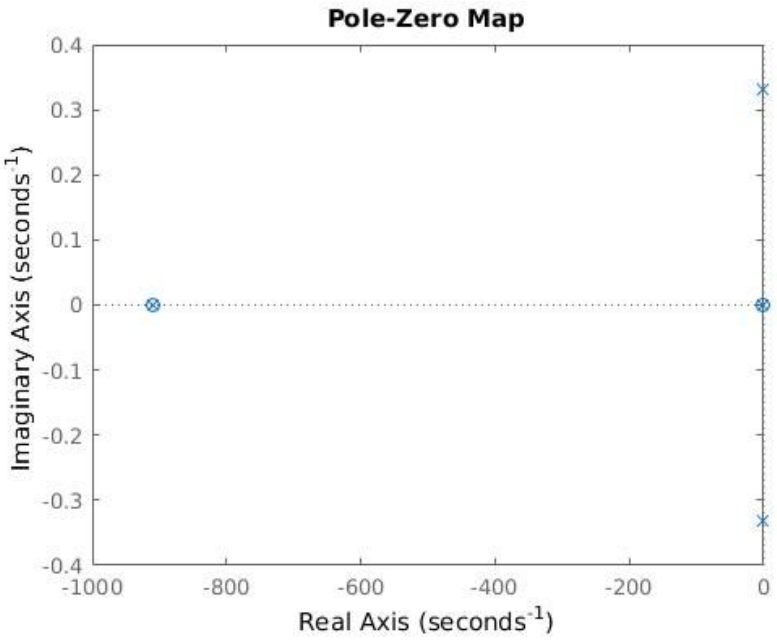


Figure 26: Pole-zero map of closed loop plus Kalman Filter for Dexter

LQR for mechanical constraints of the system

Baxter

Although impedance control can offer robustness and stability to certain performance criteria such as impact force and settling time, there is little intuitive consideration to the mechanical constraints of the system. LQR may be used to quantify the costs of control of the states of the system given limits on the control effort by the actuators. The cost function, V, is determined by the state weighting matrix, Q, and control weighting matrix R, according to the equation below.

$$\min_{\mathbf{u}(t)} V = \int_{\tau=t}^{\tau=T} \left(\mathbf{x}^t \mathbf{Q} \mathbf{x} + \mathbf{u}^t \mathbf{R} \mathbf{u} \right) d\tau$$

If the control input is a linear full-state feedback control law, **u = - Kx**, then the gain matrix, K, can be formed to minimize the cost of V.

Baxter's task space open loop transfer function (assuming no damping) has 6 poles at the origin.

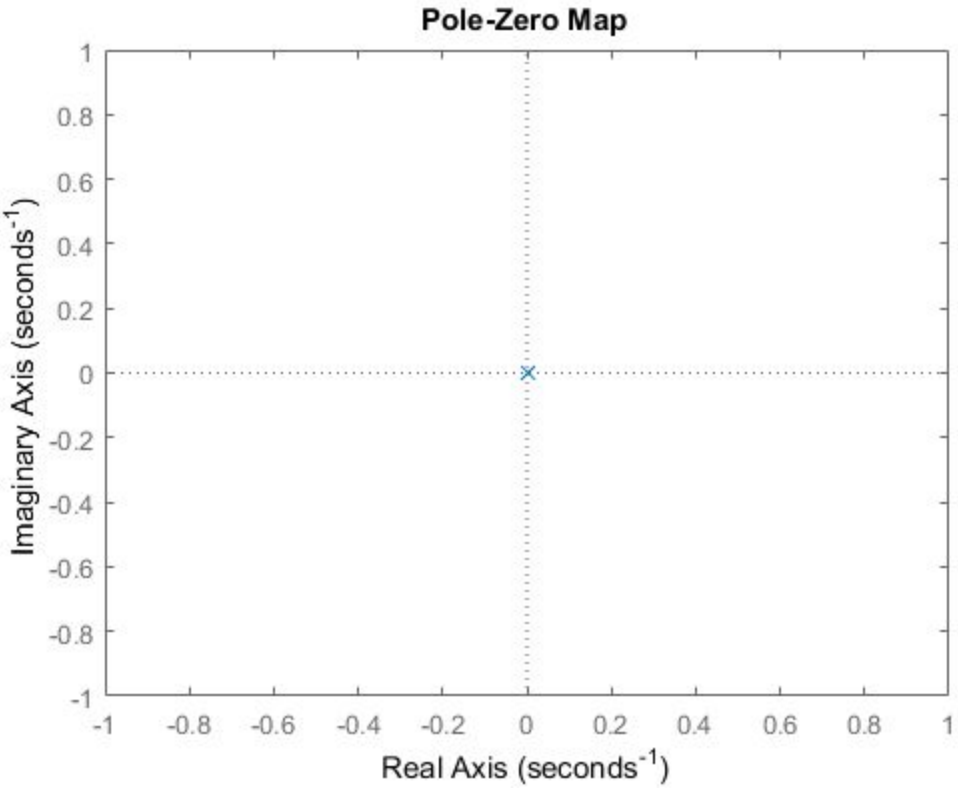


Figure 27: of pole-zero map of open loop of Baxter in task space neglecting any damping in the model at first. Six poles are at the origin.

For any mechanical system, there will reasonably be a limitation or constraint on the value of its states or on the maximum value of its input. For Baxter, the actuators are constrained by their maximum torque output.

Shoulder actuator Max Torque: 50 Nm
Elbow actuator Max Torque: 15 Nm
Wrist actuator Max Torque: 15 Nm

The states of Baxter are its x, y, and theta positions and velocities. For the task space arrangement of Baxter as a linear actuator model, the stroke length of the arm travel is the limitation. For simplicity, we can consider the stroke length to be the lengths of

Baxter's arms fully extended minus the equilibrium position of the arm (considered to be half the full length)

x position max = (L1 +L2+L3)/2

Bryson's rule for weighting matrices Q and R can be used to normalize the signals for the states and the control input.

$$Q = \begin{bmatrix} \frac{\alpha_1^2}{(x_1)^2_{\max}} & & & \\ & \frac{\alpha_2^2}{(x_2)^2_{\max}} & & \\ & & \ddots & \\ & & & \frac{\alpha_n^2}{(x_n)^2_{\max}} \end{bmatrix}$$

$$R = \rho \begin{bmatrix} \frac{\beta_1^2}{(u_1)^2_{\max}} & & & \\ & \frac{\beta_2^2}{(u_2)^2_{\max}} & & \\ & & \ddots & \\ & & & \frac{\beta_m^2}{(u_m)^2_{\max}} \end{bmatrix}$$

The x_n represent the states of the system. The u_m represent the control inputs to the system. Since we want to consider the mechanical constraints for Baxter we can normalize the weighting matrices about our limitations.

$x_1 = (L_{\text{upper arm}} + L_{\text{forearm}} + L_{\text{wrist}})/2 = 0.5214$ m
$u_1 = 50$ Nm
$u_2 = 15$ Nm
$u_3 = 15$ Nm

For the other x_n states, we can choose a weighting that is less compared to x_1 because we are more concerned with the performance of the x position. The selection of p determines the overall ratio between Q and R. Larger p values (expensive) penalizes the control effort while smaller p values (cheap) will increase performance at the cost of greater control effort. Given a set of performance criteria, the p value may be tuned to

achieve the desired response. For this project, cheap and expensive control were examined in order to discuss their implications.

Based on the x position constraint of the stroke length, the Q matrix for this control design is

```
Q =

    3.6787         0         0         0         0         0
         0    0.0100         0         0         0         0
         0         0    0.0100         0         0         0
         0         0         0    0.0100         0         0
         0         0         0         0    0.0100         0
         0         0         0         0         0    0.0100
```

and the R matrix is given as

```
R =

    0.0004         0         0
         0    0.0044         0
         0         0    0.0044
```

based on the constraints outline above.

To explore the implications of cheap and expensive control, we chose values of p (p = 0.01, 1, and 100). The pole-zero map shows how the open loop poles move for different p values.

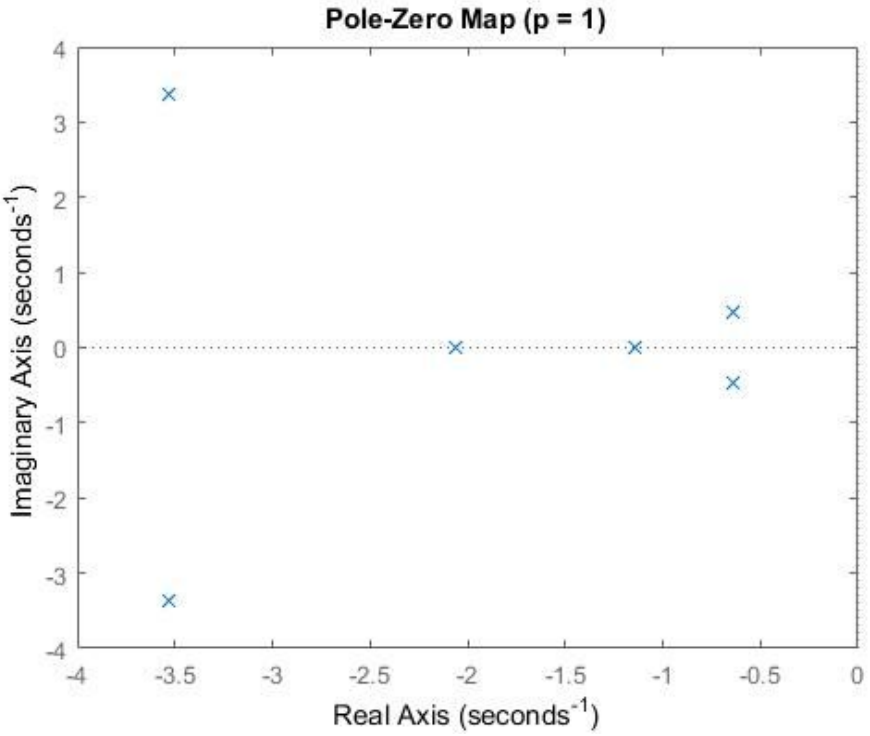


Figure 28: Pole-Zero map of closed loop LQR control with p = 1

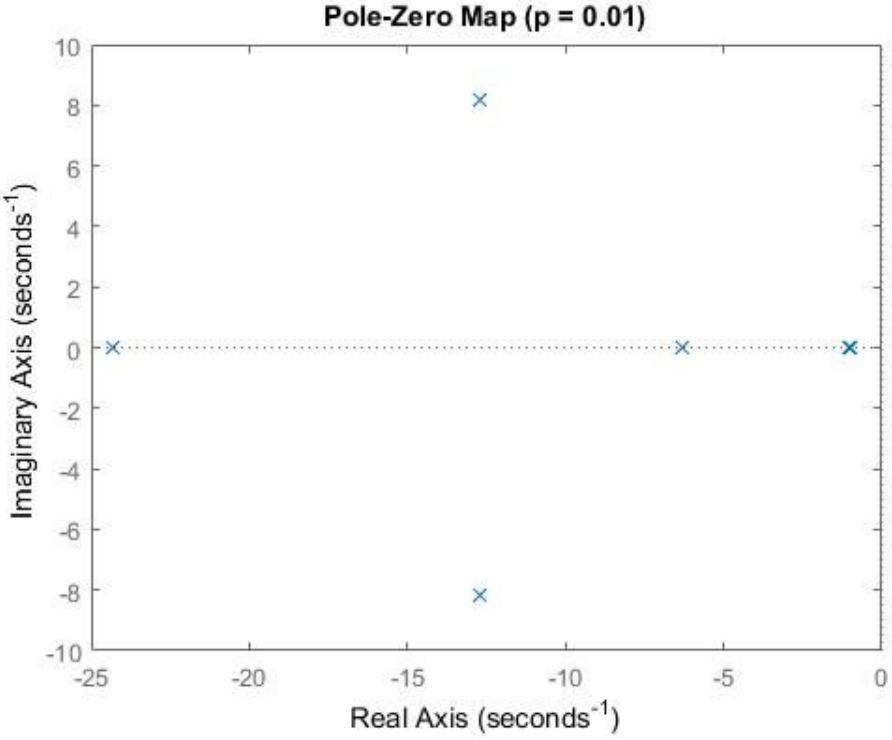


Figure 29: Pole-zero map of closed loop LQR with p = 0.01

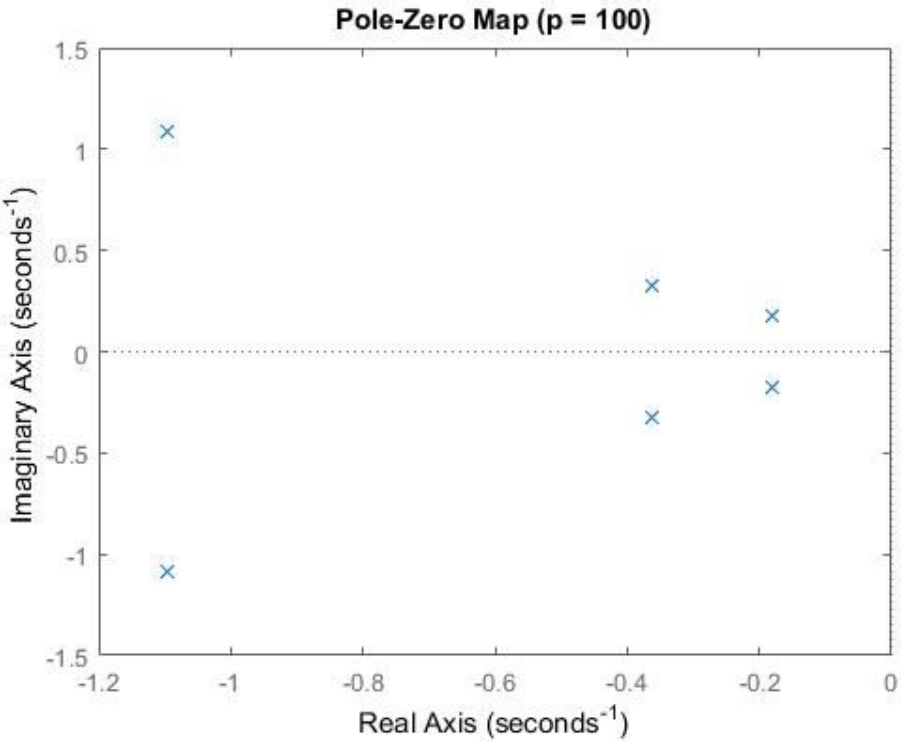


Figure 30: Pole-Zero map of closed loop LQR with p = 100

From the pole-zero maps, as p becomes smaller, the poles move further to the left of the imaginary axis and start to lie on the real axis. As these poles become faster for more cheap control, the control effort required to drive the system performance becomes greater. Conversely, more expensive control with poles closer to the imaginary axis (as in the case when p = 100) will provide a slower and more oscillatory performance but with significantly less control effort required.

Dexter

Used the same design method as use for LQR design for Baxter.
System Criterion:
Max Thrust(Ft): 10 Nm
Max Stroke Length(x2_max) = 10 in = 0.254 m
Costs for LQR based on restrictions above:
Q=[1 0 0; 0 1 0; 0 0 1/.254^2];
R=p*[1 0 0;0 1 0;0 0 1/10^2];
Three different weights for the costs were designed: p_cheap=0.001, p_equal=1, p_expensive=1000.

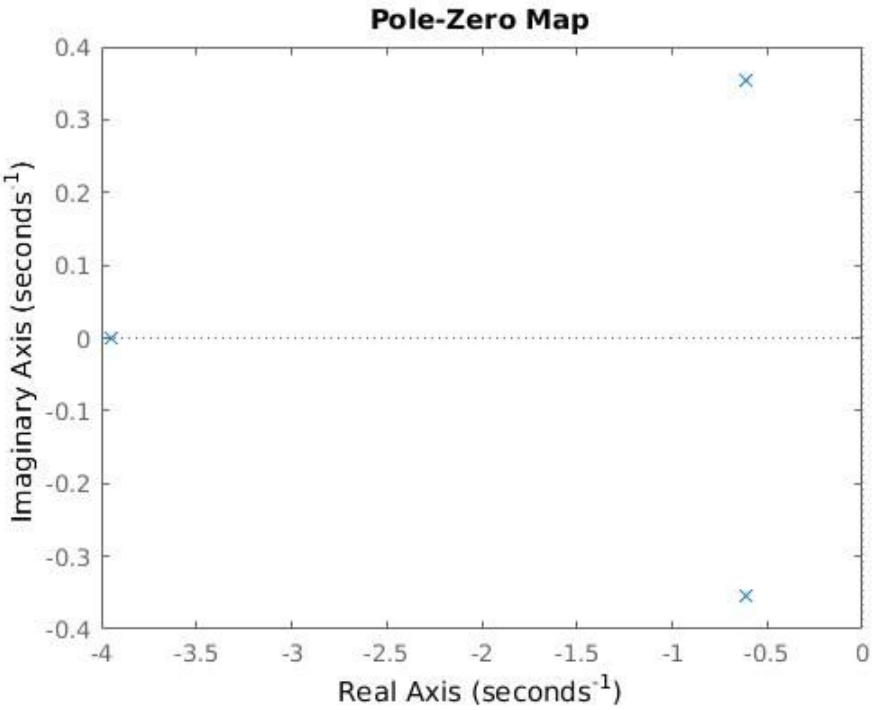


Figure 31: Pole-Zero map for LQR design with equal weighting p=1

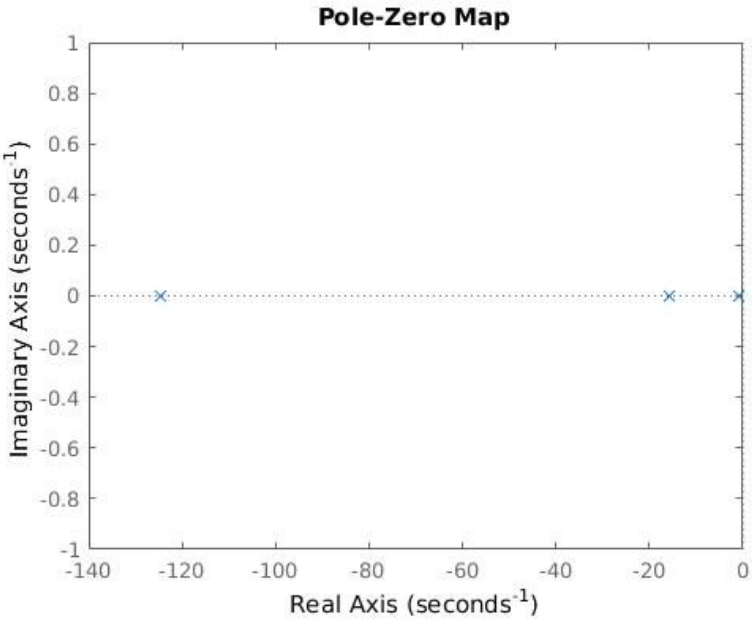


Figure 32: Pole-Zero map for LQR design with cheap weighting p=0.001

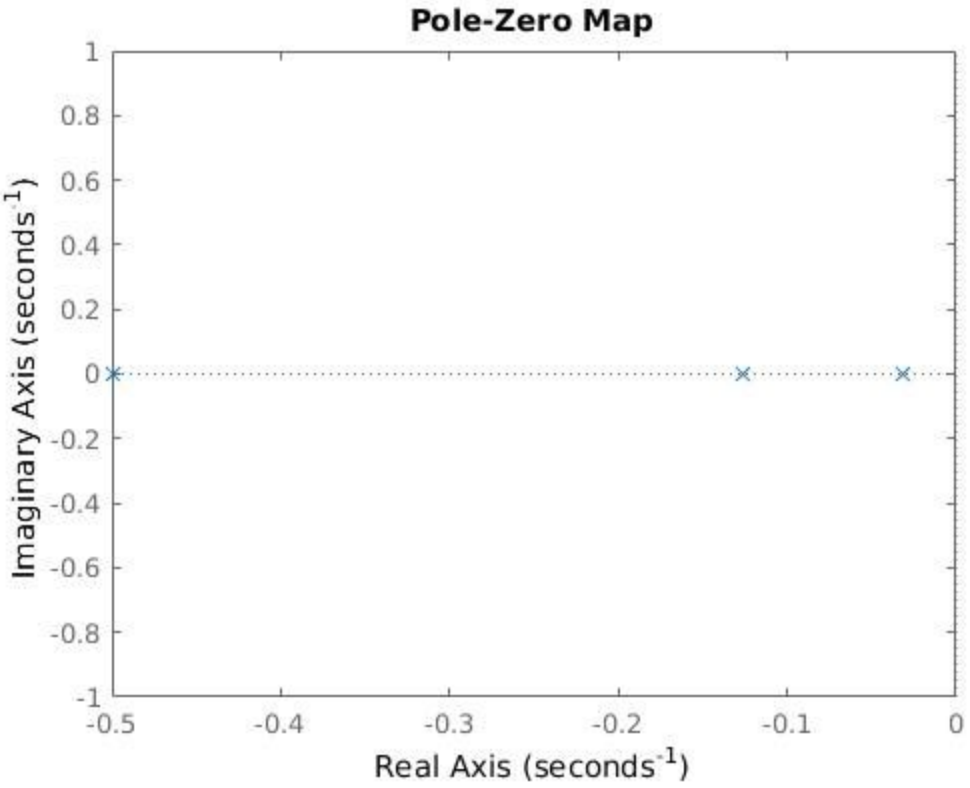


Figure 33: Pole-Zero map for LQR design with expensive weighting p=1000

IV. Results from Simulations and/or Experiments

Dexter

From previous work, it was determined that an impedance controller with a natural frequency of 1 Hz and a damping ratio of 0.443 would provided desirable interaction characteristics between a free-floating vehicle and its external environment. These impedance characteristics provided not only a reduction in peak impact force, but also provided a sharp reduction in settling time. A reduction in peak force allows for a controller to react to any unforeseen collisions, and alleviates the significant risks set upon valuable equipment onboard hard-to-reach AUVs. A reduction in settling time suggests a smaller influence by external disturbances, such as underwater currents, and minimizes unpredictable disturbances during any ongoing contact tasks.

Feeding these values into the controller result in an ideal mass value of mi = 500 kg, an ideal damping value of bi = 278 kg/s, and an ideal stiffness value of ki = 1,234 N/m. The controller's performance was then validated by comparing the measured position of the end effector relative to the raft against the developed state space model using the

same force step input. Because the positioning data is noisy, plots for natural frequencies of 0.5 Hz and 0.25 Hz are also included and plotted against simulation to better illustrate the data's fit to the model.

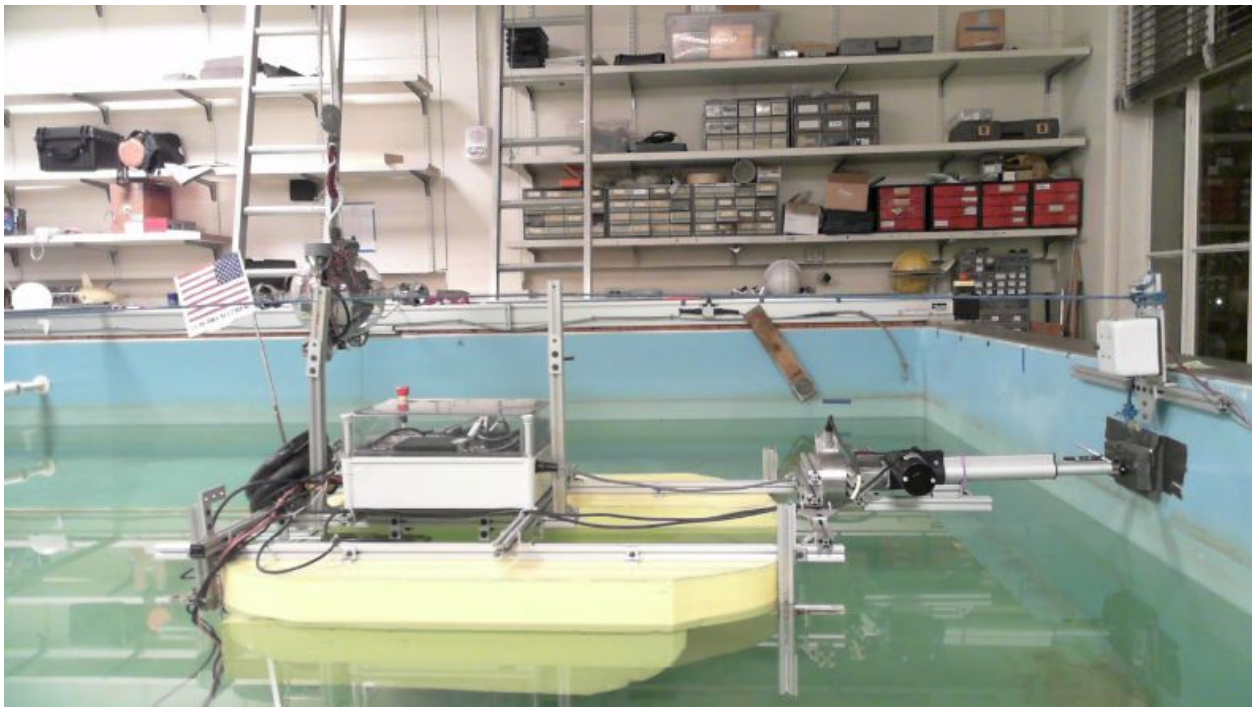
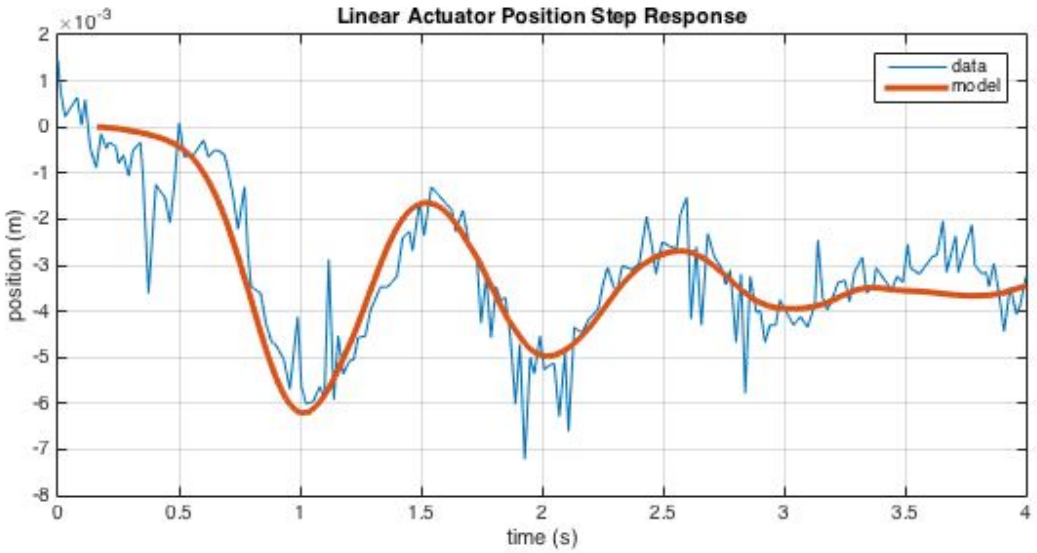
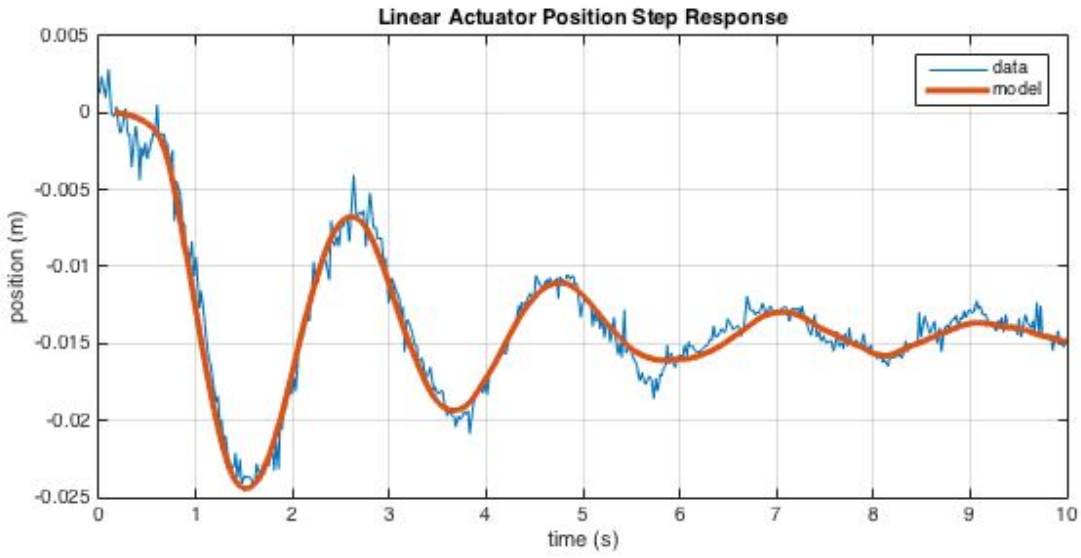


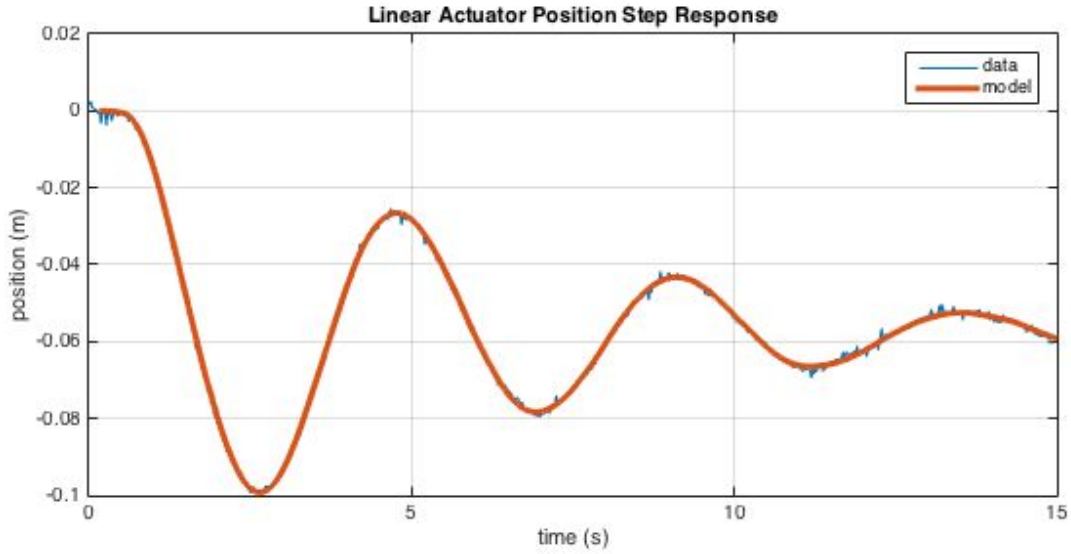
Figure 34: Experimental setup of Dexter applying force to external environment using ROV thrusters



(a)



(b)



(c)

Figure 35: Linear actuator stroke length in response to a step input of thruster force with a natural frequency of 1 Hz (a), 0.5 Hz (b), and 0.25 Hz (c), and a damping ratio of 0.0443.

With ideal impedance parameters defined and confirmed using Dexter's linear actuators, these same values are passed to Baxter in order to be reproduced.

Baxter

Comparison of Real System to The Model

To compare the real system to the simplified model that does not include the dynamics of the SEAs, Baxter was put in the planar configuration, and the the gains were set such

that it should oscillate about the equilibrium position at about 1Hz. The endpoint was then displace 0.1 m, and the resulting oscillations were recorded.

Overall, it was found that while the model predicted the correct natural frequency, Baxter's damping is dominated by stiction, rather than the linear damping that was included in the model. In future tests, this additional source of damping must be included to provide an accurate model of the behavior.

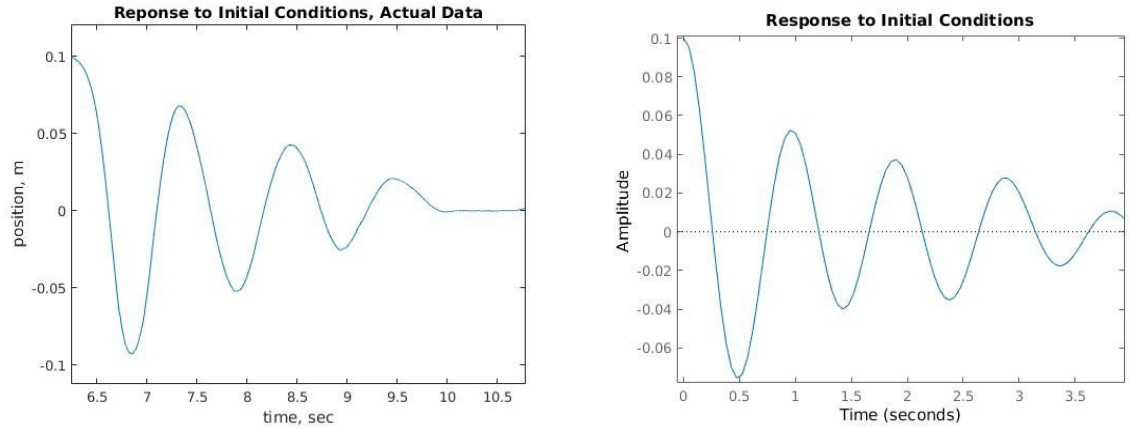


Figure 36: Initial response of Baxter (left) vs the actual model (right)

Results of Simulation for Single SEA with observers

Adding these observer poles much faster than the closed system poles shortens the time for the estimator error to converge. Given an initial estimator position input 5 cm away from the initial state position, the step response is shown.

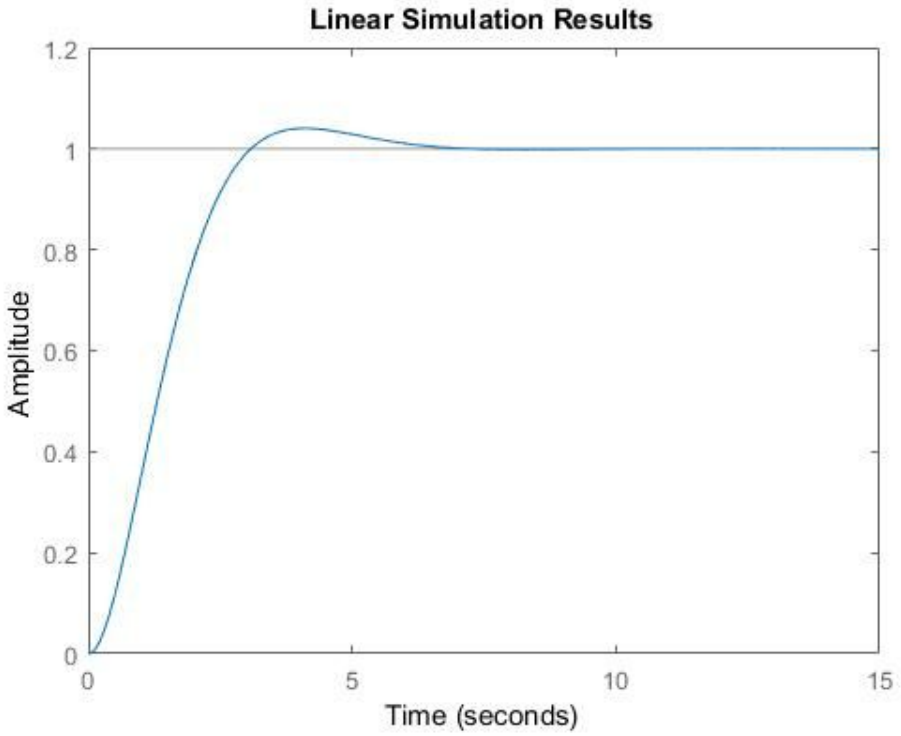


Figure 37: Step response of single SEA

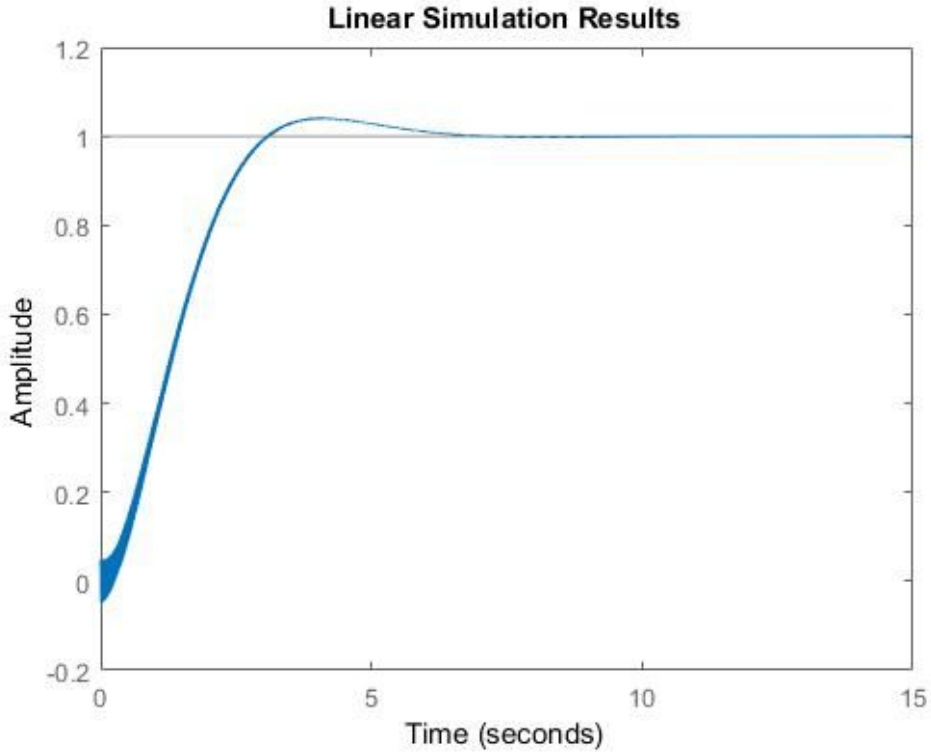


Figure 38: Observer response of single SEA

With an initial error, the observer state begins oscillatory but eventually converges to the actual state.

Results of Simulation for Dexter with observer and Kalman Filter

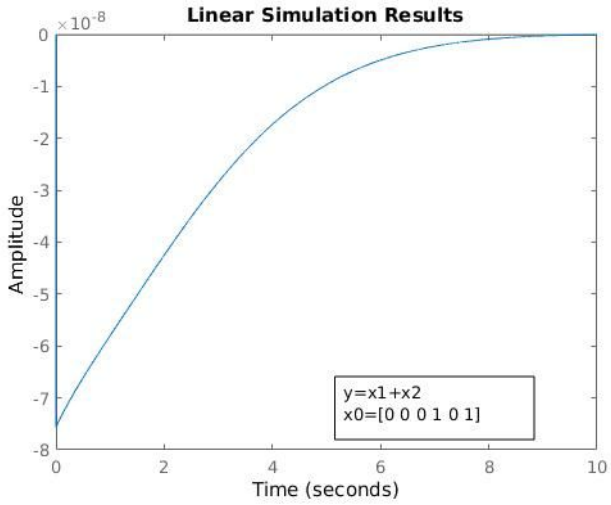


Figure 39: Simulation results of real system with initial error between real system and modeled Kalman system.

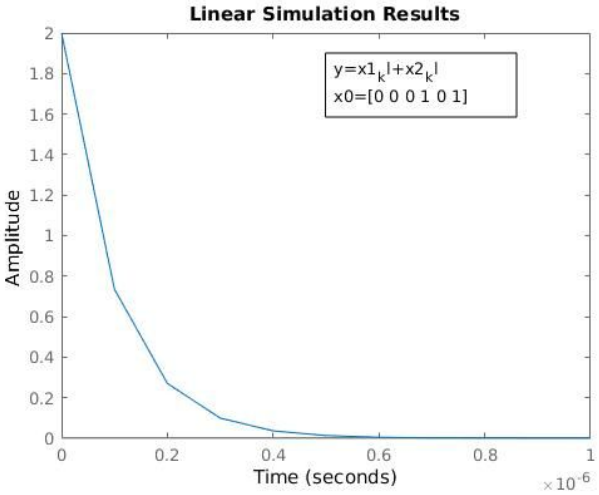


Figure 40: simulation of Kalman Filter with initial error between real system and modeled Kalman system.

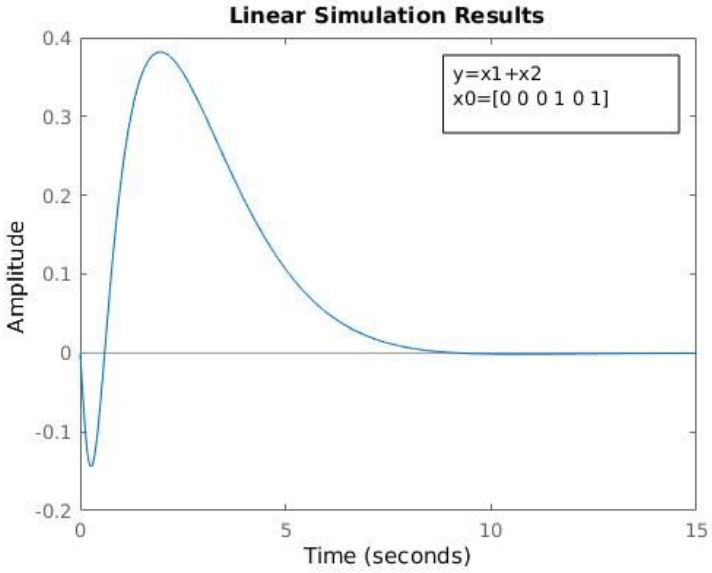


Figure 41: Simulation results of real system with initial error between real system and Observer model.

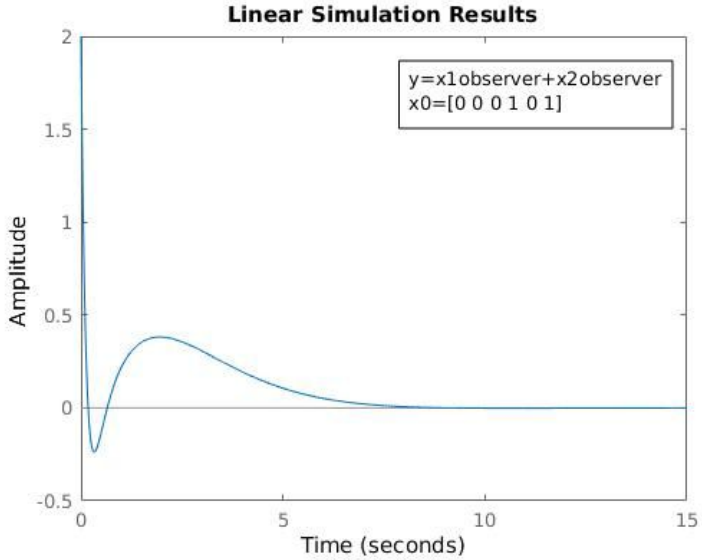


Figure 42: Simulation results of modeled system with initial error between real system and Observer model.

The optimal observer designed using Kalman Filter technique converges much faster than the observer designed using observer design methods of choosing poles that are 5 times left of the real poles of the system.

Result of Simulation for LQR design

A response to an initial condition can provide more insight into the tradeoffs expected for these control schemes. An initial **x** position of 0.3 meters was selected. In terms of Baxter, this response would represent his arm contracting to equilibrium from a position extended 0.3 meters.

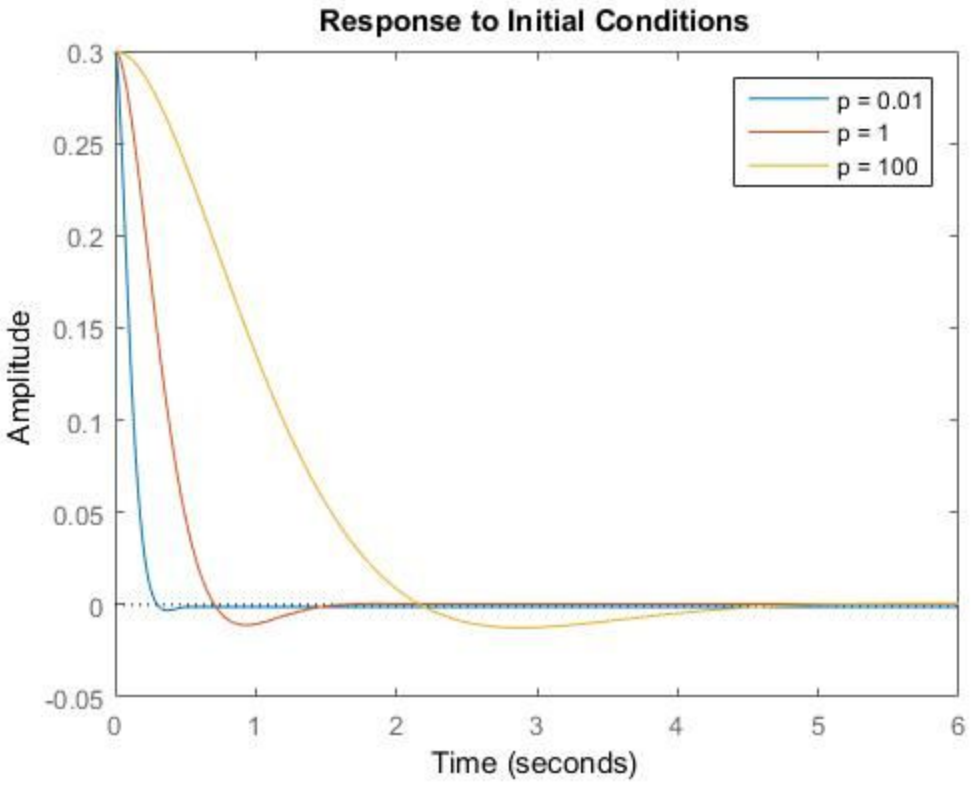


Figure 43: Initial condition response with varying p scalar (cheap to expensive)

The differences in performance are clearly illustrated by this response to initial conditions. Of course, cheap control (p = 0.01) has the best performance, with a faster rise time and shorter settling time. Depending on the specific performance criteria for the system, even expensive control (p = 100) may be considered a decent response.

Control effort should be examined to determine how the actuator torques are affected by cheap to expensive control.

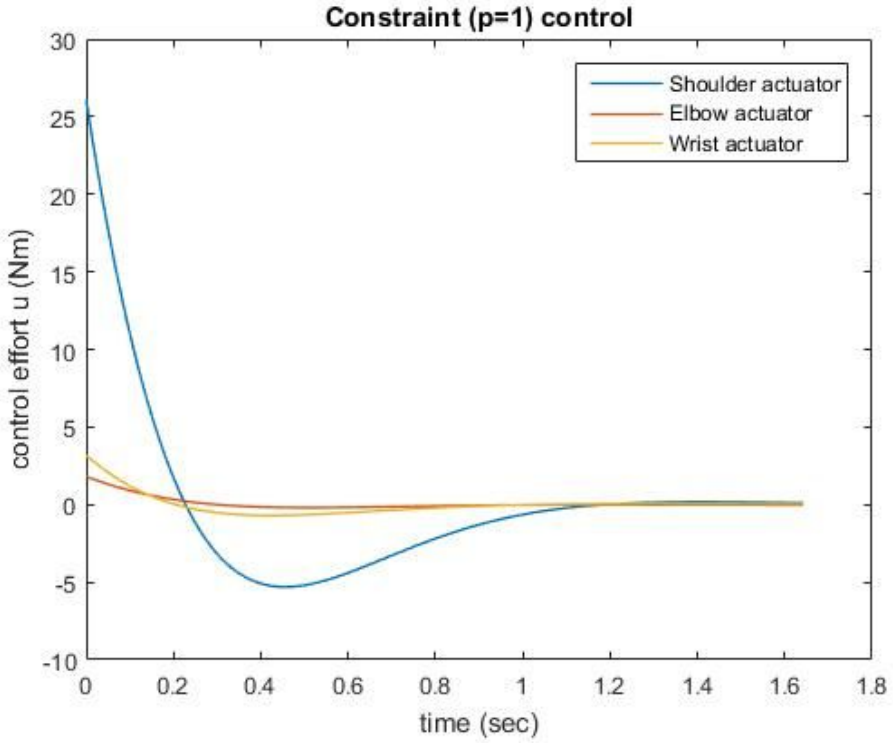


Figure 44: Control effort from initial conditions with p = 1

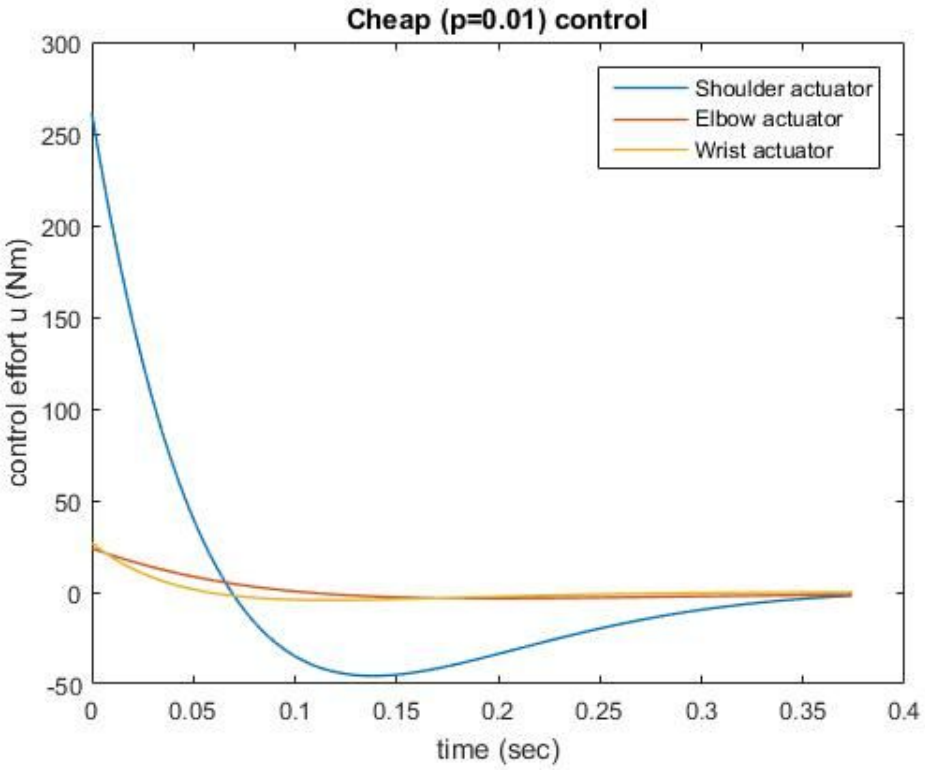


Figure 45: Control effort from initial conditions with p = 0.01

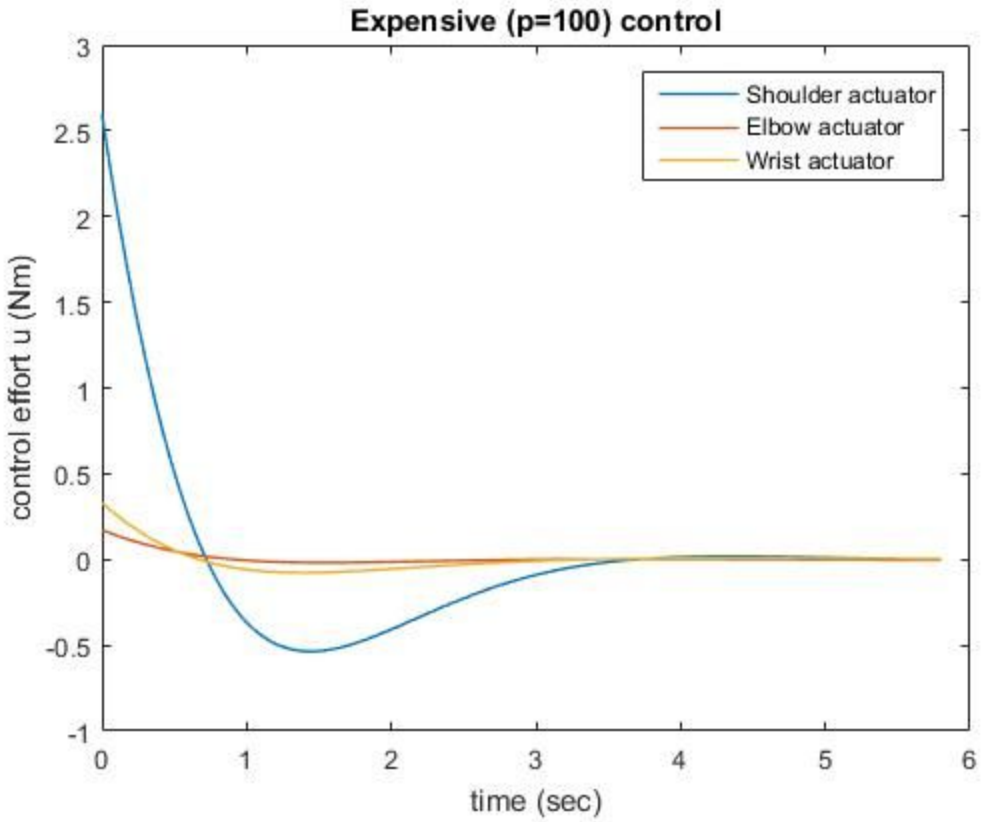


Figure 46: Control effort from initial conditions with p = 100

Dexter: LQR results

The three LQR designs were each test from initial deviation of 1.254 from steady state. x1_0=1, x2_0=.254.

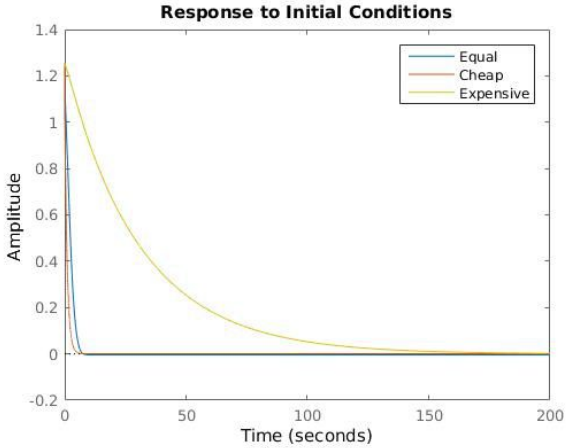


Figure 47: Closed Loop Responses for initial condition: x0=[1,0,.254]

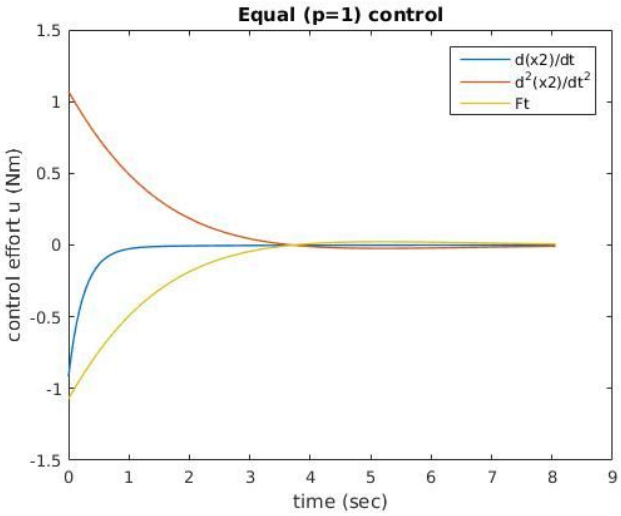


Figure 48: Effort plot for equal controller

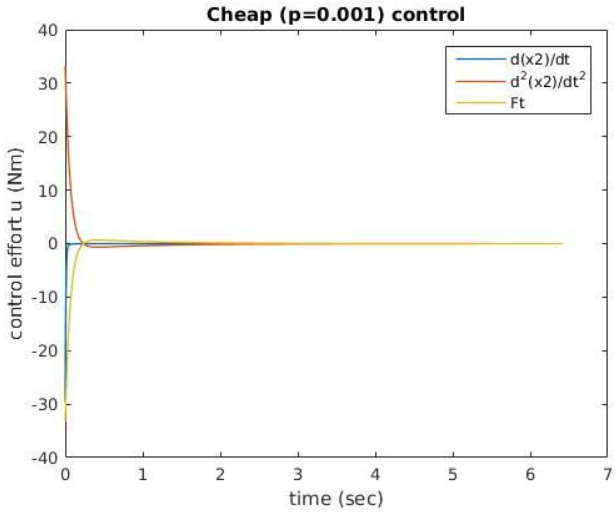


Figure 49: Effort plot for cheap controller

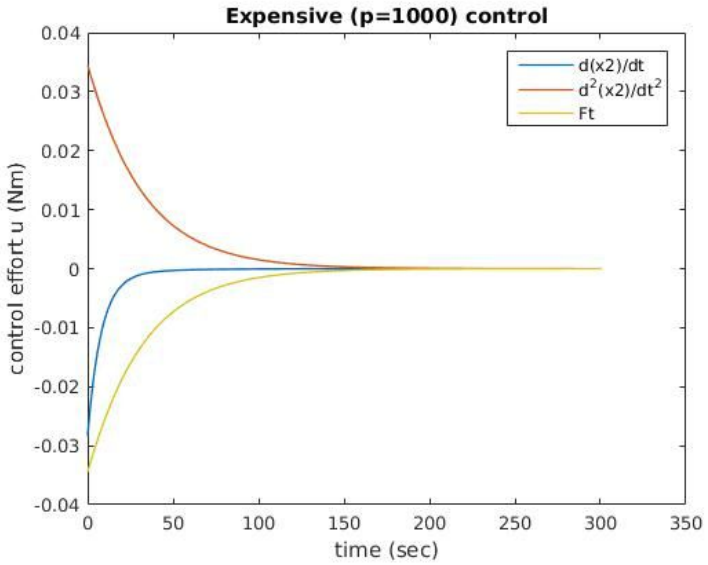


Figure 50: Effort plot for expensive controller

From the above plots of control effort, there is an obvious tradeoff of performance for control effort. To achieve the better system response from cheap control, Baxter and Dexter would require stronger actuators because the control requirements are above max inputs. If a more expensive controller is used such that it falls below the max input restrictions, performance and speed is necessarily sacrificed. Thus, tuning **p** according to desired performance goals could try cheaper control with consideration of the maximum torque output by the motors. Reasonable prediction of environmental or task expectations may assist in selecting these performance criteria.

V. Conclusions and Recommendations

Given the mechanical constraints of Baxter's and Dexter's actuators (such as maximum force, torque, and stroke length) and state constraints, LQR control design is a good method to consider such limitations and achieve an acceptable performance. For an arbitrary multi-degree of freedom robot to replace decommissioning tasks, selection of actuators and geometry will greatly influence the cost, and this process can be assisted by this design approach in order to optimize for cost and functionality. Careful tuning of the Q and R weighting matrices can optimize performance for the maximum properties and environmental expectations of the system. In the future, incorporating impedance into the cost function for LQR design may help designers of the robot optimize stiffness and damping according to the mechanical limitations of their hardware and environment.

The influence of delay is a significant inhibitor in terms of achieving certain impedance characteristics. As well as input/output delays may be characterized, the implication of a delay is that the system needs time to respond to any input. With respect to peak force due to impact on Dexter in the presence of delay, a peak force reduction is only possible with enough mechanical compliance, such as with a spring tip on the end effector. This provides a buffer for the controller to react to any intentional or unforeseen collisions before the entire force of impact has been imparted onto the vehicle. This is particularly valuable for autonomous decommissioning, since it provides a level of safety for the vehicle against any potential damage from the shock of impact. Given the drive to remove support vessels as much as possible from the process, impact damage to an autonomous decommissioning vehicle would be especially detrimental without the presence of a nearby support vessel to service the vehicle.

Together, the work done with both Baxter and Dexter have helped show that both a free-floating system and a backdrivable system using impedance control could provide a method through which an AUV's interaction with the environment can be controlled. It allows for a versatile multi-degree-of-freedom arm to express a complementary versatile range of interaction characteristics to fit a variety of scenarios, such a scrubbing biofouling, testing valves, or even mitigating the effects of an unexpected collision. The use of impedance control in autonomous underwater decommissioning of offshore structures offers a way to bridge the gap of capabilities that both divers and ROVs possess into one consolidated autonomous system, and opens the door to their potential replacement in the future.

List of Project Tasks

Lucille Hosford: Open and closed loop model of Baxter, comparison of actual behavior to simulation, and open loop model and discussion of Series Elastic Actuator dynamics

Brian Wilcox: LQR Design and discussion for Baxter. Observer design for single SEA.

Manjinder Singh: Open Loop model, LQR Design, Observer, and Kalman Filter design for Dexter

Christian Welch: Open Loop model and closed Loop model of Dexter, experimental comparison of actual system to system model, actuator analysis of thrusters and linear actuator

References

[1] Williamson, (1995) *Series Elastic Actuators*

[2] Observer diagram image
http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlStateSpace

[3] Colgate, James Edward, and Neville Hogan. “Robust control of dynamically interacting systems.” International journal of Control 48.1 (1988): 65-88.

[4] Fadali, M. Sami, Dr. “Discrete-Time State-Space Equations.” 22 May 2015. PDF file.

[5] Techet, A. H., Prof. “13.012 Hydrodynamics for Ocean Engineers.” File last modified on 30 Nov. 2004. PDF file.

[6] Chevron Energy Technology Company Subsea Group. Subsea Acoustic Positioning and Surveys Manual. Rept. no. 1400. Houston: n.p., 2008. Print.

Appendix

Brian and Lucille's Matlab Code:

1. **State Space Model of a Torque Controlled, Planar Robot With 3 DOFs and Series Elastic Actuators**

```
clear all,clc, close all


%%

%Model neglecting Series Elastic Actuators
%Open Loop
CIM =[11.3011 1.1257 2.3917;
   1.1257 6.1354 0.2495;
   2.3917 0.2495 1.1338];

%Damping. For now, set to zero.
% Damping=[0.1 0 0;
%     0 0.1 0;
%     0 0 0.1];

Damping=[0 0 0;
   0 0 0;
   0 0 0];

jj =[-0.3199 -0.0883 -0.2295;
   0.3004 -0.3639 0.0017;
   0 0 1.0000];


A1=[zeros(3) eye(3);
   zeros(3) -inv(CIM)*Damping];

B1=[zeros(3);
   inv(CIM)*inv(jj')];

C1=[1 0 0 0 0 0];
```

```
D1=[0];

%%
%LQR design for Baxter


l1=0.438809; %m link length 1
l2=0.37442; %m link length 2
l3=0.229525; %m link length 3




Q = eye(6);
Q(1,1) = 1/(l1/2+l2/2+l3/2)^2;
Q(2,2)=1e-2;
Q(3,3)=1e-2;
Q(4,4)=1e-2;
Q(5,5)=1e-2;
Q(6,6)=1e-2;



% Q = C1'*C1;



p1 = 1e-2;  %cheap
p2=1; %equal weight
p3 = 1e2; %expensive

R = eye(3);

R(1,1) = 1/(50^2);
R(2,2) = 1/(15^2);
R(3,3) = 1/(15^2);

R1 =p1*R;
R2 = p2*R;
```

```
R3 = p3*R;

[K1,S1,e1] = lqr(A1,B1,Q,R1);
[K2,S2,e2] = lqr(A1,B1,Q,R2);
[K3,S3,e3] = lqr(A1,B1,Q,R3);

Ac1 = A1-B1*K1;
Ac2 = A1-B1*K2;
Ac3 = A1-B1*K3;


sys = ss(A1,B1,C1,D1);
sys1 = ss(Ac1,B1,C1,D1);
sys2 = ss(Ac2,B1,C1,D1);
sys3 = ss(Ac3,B1,C1,D1);

%%

%Observer design for single SEA

Ksea=843;  %Spring stiffness
m1=0.6;    %mass, motor side
m2=0.6*0.4; %mass, link side
D=0.1;   %Damping, link side
B=0.1;  %Damping, motor side

Asea=[0 1 0 0;
   -Ksea/m2 -D/m2 Ksea/m2 0;
    0 0 0 1;
    Ksea/m1 0 -Ksea/m1 -B/m1];

Bsea=[0;
    0;
    0;
    1/m1];


Csea=[1 0 0 0];
%C1=[-Ksea -D Ksea 0];
```

```
Dsea=[0];

Qs=Csea'*Csea;

Rs=[1/1];

[K,S,e]=lqr(Asea,Bsea,Qs,Rs);

Ac=Asea-Bsea*K;

syssea=ss(Asea,Bsea,Csea,Dsea);
syssc=ss(Ac,Bsea,Csea,Dsea);
op = 5*pole(syssc);

L = place(Asea',Csea',op')';

% At = [ Asea-Bsea*K            Bsea*K
%        zeros(size(Asea))    Asea-L*Csea ];
%
% Bt = [    Bsea
%        zeros(size(Bsea)) ];
%
% Ct = [ Csea    zeros(size(Csea)) ];

At = [ Asea          -Bsea*K
      L*Csea    Asea-Bsea*K-L*Csea ];

Bt = [    Bsea
      Bsea ];

Ct = [ Csea    zeros(size(Csea)) ];
Ct2 = [zeros(size(Csea)) Csea ];
Ct3 = [-Ksea -D Ksea 0 zeros(size(Csea))];

syst = ss(At,Bt,Ct,Dsea);
syst2 = ss(At,Bt,Ct2,Dsea);
% syst3 = ss(At,Bt,Ct3,Dsea);
% [b,a] = ss2tf(At,Bt,Ct,Dsea);
```

```
%%

%Plots

%LQR Plots

% figure
% pzmap(sys)
% zpk(sys)
x0=[0.3 0 0 0 0 0];
% figure
% initial(sys1,sys2, sys3,x0)
% figure
% step(sys1,sys2,sys3)
% figure
% pzmap(sys1)
% zpk(sys1)
% figure
% pzmap(sys2)
% zpk(sys2)
% figure
% pzmap(sys3)
% zpk(sys3)
% figure
% bode(sys1.sys2,sys3)
% figure

[y1,t1,x1] = initial(sys1,x0) ;
% initial(sys1,x0)
% figure
[y2,t2,x2] = initial(sys2,x0) ;
% initial(sys2,x0)
% figure
[y3,t3,x3] = initial(sys3,x0) ;
% initial(sys3,x0)
u1 = [-K1*x1']' ;
u2 = [-K2*x2']' ;
```

```
u3 = [-K3*x3']' ;


J1 = x0*S1*x0' ;
J2 = x0*S2*x0' ;
J3 = x0*S3*x0' ;

%

% Control Effort plots

figure
plot(t1,u1,'LineWidth',1)
xlabel('time (sec)')
ylabel('control effort u (Nm)')
title('Cheap (p=0.01) control')
legend('Shoulder actuator', 'Elbow actuator', 'Wrist actuator')
figure
plot(t2,u2, 'LineWidth',1)
xlabel('time (sec)')
ylabel('control effort u (Nm)')
title('Constraint (p=1) control')
legend('Shoulder actuator', 'Elbow actuator', 'Wrist actuator')
figure
plot(t3,u3, 'LineWidth',1)
xlabel('time (sec)')
ylabel('control effort u (Nm)')
title('Expensive (p=100) control')
legend('Shoulder actuator', 'Elbow actuator', 'Wrist actuator')

%%
%Observer Plots

figure
pzmap(syssea)
zpk(syssea)

figure
pzmap(syssc)
```

```
zpk(syssc)

figure
pzmap(syst)
zpk(syst)

x0 = [0.05 0 0 0];  % initial condition for estimator
% x0 = [0 0 0 0];  % initial condition for estimator
t = 0:0.003:15;


figure
lsim(syst,ones(size(t)),t,[0 0 0 0 x0]); % simulated step input of 1
figure
lsim(syst2,ones(size(t)),t,[0 0 0 0 x0]);
figure
% bode(syssea,syst)
```

2. Single SEA model

```
clear all,close all,clc
%Open Loop

Ksea=843;  %Spring stiffness
m1=0.6;    %mass, motor side
m2=0.6*0.4; %mass, link side
D=0.1;   %Damping, link side
B=0.1;  %Damping, motor side

A1=[0 1 0 0;
   -Ksea/m2 -D/m2 Ksea/m2 0;
    0 0 0 1;
    Ksea/m1 0 -Ksea/m1 -B/m1];

B1=[0;
    0;
    0;
    1/m1];


C1=[1 0 0 0];
%C1=[-Ksea -D Ksea 0];

D1=[0];

sys=ss(A1,B1,C1,D1);

RC=rank(ctrb(A1,B1)); %Controllability of SEA
RO=rank(obsv(A1,C1));   %Observability of SEA




%system is fully controllable, but not observable looking at the link side
%velocity
Q=C1'*C1;

R=[1];
```

```
[K,S,e]=lqr(A1,B1,Q,R);

Ac=A1-B1*K;
sysc=ss(Ac,B1,C1,D1);
figure
pzmap(sysc)
figure
bode(sys)
grid minor

figure
pzmap(sys)

tr=zpk(sys);

figure
step(sys)

figure

rlocus(sys)

% x0=[1 0 1 0];
% figure
% initial(sys,x0)

%%
%Closed Loop
Kd=10;
Bd=-0.05;

Ac=[0 1 0 0;
   -Ksea/m2 -D/m2 Ksea/m2 0;
   0 0 0 1;
   Ksea/m1-Kd/m1 -Bd/m1 -Ksea/m1 -B/m1];

Bc=[0;
   0;
```

```
    0;
    Kd/m1];
%The poles in the rhp are coming from the damping term!?!?!

Cc=[0 1 0 0];

Dc=[0];

sys2=ss(Ac,Bc,C1,D1)

figure
bode(sys2)
grid minor
figure
pzmap(sys2)
tr2=zpk(sys2);

x0=[1 0 1 0];
figure
initial(sys2,x0)
```

Chris's Matlab Code:

1. **Linear Actuator Bode Plot**

```
close all
clear all
load actuator_calib.mat

% Linear Actuator Conversion Constants
%pot_meter_cal = 0.046164139161463; %m/V
%pot_speed_cal = 1.470509064524557e+02; %cmd/(V/s)
t_step = 0.1;

% Define the frequency range to test the arm
%omega = 10.^(0.1:0.01:2);
omega = [10.^0.1]; % for testing
mag  = zeros(1,1);
phase = zeros(1,1);
phase_lag = zeros(1,1);
amplitude_ratio = zeros(1,1);

%% Perform frequency response

for j=1:length(omega)

   % Initialize MOOS
   mexmoos('CLOSE');
   pause(1);
   disp('Initializing MOOS...')
   mexmoos('init','SERVERHOST','localhost','SERVERPORT','9000');
   pause(1);
   disp('Complete!')

%% Zero Linear Actuators

% Register for string potentiometer voltage and load cell force
mexmoos('REGISTER','ROBOTEQ_AIN1_VOLTS',0);

% Initialize variables
```

```
x = zeros(1,1);

% Define center point
center = 2.25;

% Initialize index variables
pos_idx = 1;
%
% Centering Linear Actuators
disp('Centering linear actuators...')
zeroed = 0;



while (zeroed==0)
   msgs=mexmoos('FETCH');
   mexmoos('NOTIFY','DESIRED_POWER_CH1',0);

   if (~isempty(msgs))
      for k=1:length(msgs)
         if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS'))
            x(pos_idx)=msgs(k).DBL;
            pos_idx = pos_idx + 1;
         end
      end
   end

   if x(end) > center && x(end)~=0
      disp('Too far...')
      while (zeroed == 0)
         msgs=mexmoos('FETCH');
         mexmoos('NOTIFY','DESIRED_POWER_CH1',-50); % Pulls inward
         if (~isempty(msgs))
            for k=1:length(msgs)
               if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS'))
                  x(pos_idx)=msgs(k).DBL;
                  pos_idx = pos_idx + 1;
               end
            end
         end
```

```
        if x(end)<= center
            zeroed = 1;
            mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
        end

        pause(0.02);
    end

elseif x(end) < center  && x(end)~=0
    disp('Too close...')
    while (zeroed == 0)
        msgs=mexmoos('FETCH');
        mexmoos('NOTIFY','DESIRED_POWER_CH1',50); % Pushes outward
        if (~isempty(msgs))
            for k=1:length(msgs)
                if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS'))
                    x(pos_idx)=msgs(k).DBL;
                    pos_idx = pos_idx + 1;
                end
            end
        end
        if x(end)>= center
            zeroed = 1;
            mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
        end
        pause(0.02);
    end
end

pause(0.02);

end
disp('Zeroed!')
%

%% Find initial position and force readings:
x = zeros(1,1);
t = zeros(1,1);
```

```
pos_idx = 1;

% Centering Linear Actuators
disp('Finding initial values...')
initialized = 0;
tic
while (initialized==0)
   msgs=mexmoos('FETCH');

   if (~isempty(msgs))
      for k=1:length(msgs)
         if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS'))
            x(pos_idx)=msgs(k).DBL;
            t(pos_idx)=toc;
            pos_idx = pos_idx + 1;
         end
      end
   end

   pause(0.01)

   if toc > 3
      initialized = 1;
   end
end

x0 = mean(x);
disp('Initialized!')

   %% Analyze Linear Actuators

   % Initialize MOOS
   mexmoos('CLOSE');
   pause(1);
   disp('Initializing MOOS...')
   mexmoos('init','SERVERHOST','localhost','SERVERPORT','9000');
   pause(1);
   disp('Complete!')
```

```
% Register for string potentiometer voltage and load cell force
mexmoos('REGISTER','ROBOTEQ_AIN1_VOLTS',0);

% Initialize variables
cmd = zeros(1,1);
command = zeros(1,1);
t = zeros(1,1);
x = x0;
xlp = x0;
X = [x0,x0];
Xlp = x0*ones(1,2);
v = zeros(1,1);
Vlp = zeros(1,2);
vlp = zeros(1,1);
xt = zeros(1,1);
XT = zeros(1,2);
com_pos = zeros(1,1);
delta_pos = zeros(1,1);
cmd_actual = zeros(1,1);


% Initialize index variables
pos_idx = 1;

% Trying to smooth this damn data...
x_mean = zeros(1,1);
f_mean = zeros(1,1);
v_mean = zeros(1,1);
fdot_mean = zeros(1,1);
X_state = zeros(2,1);
Y_state = zeros(2,1);
ct = zeros(1,1);
com_idx = 1;
pos_toc = 0;
force_toc = 0;


% Anayze Linear Actuators
disp('Running analysis...')
```

```
exit = 0;
tic;
while (exit==0)
   msgs=mexmoos('FETCH');

   if (~isempty(msgs))
      for k=1:length(msgs)
         if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS') && toc-pos_toc >= 0.01)
            pos_toc = toc;
            cmd = cos((pos_toc)*omega(j))*100;
            if cmd<0
               cmd_adj = P_lhp(1).*cmd.^6+P_lhp(2).*cmd.^5+P_lhp(3).*cmd.^4+...
                  P_lhp(4).*cmd.^3+P_lhp(5).*cmd.^2+P_lhp(6).*cmd.^1+...
                  P_lhp(7).*cmd.^0;
            elseif cmd>0
               cmd_adj = P_rhp(1).*cmd.^6+P_rhp(2).*cmd.^5+P_rhp(3).*cmd.^4+...
               P_rhp(4).*cmd.^3+P_rhp(5).*cmd.^2+P_rhp(6).*cmd.^1+...
               P_rhp(7).*cmd.^0;
            else cmd_adj = cmd;
            end
            mexmoos('NOTIFY','DESIRED_POWER_CH1',cmd_adj);
            command(pos_idx) = cmd;
            com_pos(pos_idx) = sin((pos_toc)*omega(j))*100/omega(j);
            xt(pos_idx)=toc;
            XT(2)=xt(pos_idx);
            x(pos_idx)=msgs(k).DBL;
            X(2) = x(pos_idx);
            if pos_idx==2
               v(1:pos_idx) = (X(2)-X(1))/(XT(2)-XT(1));
            end
            if pos_idx>=2
               if abs((X(2)-X(1))/(XT(2)-XT(1)))> 10+abs(v(pos_idx-1))
                  v(pos_idx)=v(pos_idx-1);
               else
                  v(pos_idx) = (X(2)-X(1))/(XT(2)-XT(1));
               end

            end
            if pos_idx==3;
```

```
            v(1:2) = v(3);
        end

        XT(1)=XT(2);
        X(1)=X(2);
        pos_idx = pos_idx + 1;
      end
    end
  end


  if (toc > 15*2*pi/omega(j))
    mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
    exit = 1;
  end

  if x(end)>4.25 || x(end)<0.25
    mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
    exit = 1;
  end

end
disp('Complete!')

mexmoos('CLOSE');

%% Frequency Analysis

% Correct position drift
p=polyfit(xt,x,2);
plot(xt,x,xt,p(1)*xt.^2+p(2)*xt+p(1));
x_nodrift = x-(p(1)*xt.^2+p(2)*xt+p(1));
plot(xt,x_nodrift)



figure(2)
subplot(2,1,1)
```

```
plot(xt,-com_pos*pot_meter_cal*100/pot_speed_bode,xt,-(x-mean(x))*pot_meter_cal*10
0)
   xlabel('Time (s)')
   ylabel('Position (cm)')
   grid on
   legend('Command (Zero-Mean)','Response (Zero-Mean)')
   title('Position Drift of Linear Actuator')
   subplot(2,1,2)

plot(xt,-com_pos*pot_meter_cal*100/pot_speed_bode,xt,-(x_nodrift-mean(x_nodrift))*po
t_meter_cal*100)
   xlabel('Time (s)')
   ylabel('Position (cm)')
   grid on
   legend('Command (Zero-Mean)','Response (Zero-Mean)')
   title('Corrected Position Drift of Linear Actuator')


   %
   % Prepare data for FFT
   window_xt=sin(pi*xt/xt(end));
   comFFT=padarray(((com_pos-mean(com_pos))/pot_speed_bode).*window_xt,[0
length(com_pos)]);
   potFFT=padarray(((x_nodrift-mean(x_nodrift))).*window_xt,[0 length(x)]);

   npts = length(comFFT);
   NFFT = 2^nextpow2(npts);
   Fs = npts/(xt(end)-xt(1));

   f = Fs/2*linspace(0,1,NFFT/2+1);

   % take the FFT
   X=fft(comFFT,NFFT)/npts;
   Y=fft(potFFT,NFFT)/npts;


   % Calculate the numberof unique points
```

```
figure(3)
subplot(211);
plot(f,2*abs(X(1:NFFT/2+1)));
title('X(f) : Magnitude response');
ylabel('|X(f)|')
subplot(212)
plot(f,2*abs(Y(1:NFFT/2+1)));
title('Y(f) : Magnitude response')
xlabel('Frequency (Hz)');
ylabel('|Y(f)|')

figure(4)
subplot(211)
plot(f,angle(X(1:NFFT/2+1)));
title('X(f) : Phase response');
ylabel('Phase (rad)');
subplot(212)
plot(f,angle(Y(1:NFFT/2+1)));
title('Y(f) : Phase response');
xlabel('Frequency (Hz)');
ylabel('Phase (rad)');

% Determine the max value and max point.
% This is where the sinusoidal
% is located. See Figure 2.
%{
[~, idx_x] = max(abs(X));
[~, idx_y] = max(abs(Y));
X(idx_x)=0;
Y(idx_y)=0;
%}
[mag_x, idx_x] = max(abs(X));
mag_y = abs(Y(idx_x));
% determine the phase difference
% at the maximum point.
px = angle(X(idx_x));
py = angle(Y(idx_x));
phase_lag(j) = py - px
amplitude_ratio(j) = rms((x_nodrift-mean(x_nodrift)))/rms(com_pos/pot_speed_bode)
```

```
    % determine the amplitude scaling
    %amplitude_ratio(j) = mag_y/mag_x
   %}
end

for l=1:length(phase_lag)
    if phase_lag(l)<-180
        phase_lag(l) = phase_lag(l)+2*pi;
    end
    if phase_lag(l)>0
        phase_lag(l) = phase_lag(l)-2*pi;
    end
end

nyquist_freq = 2*pi/(mean(diff(xt)));

% Make Bode Plot
figure(5)

subplot(2,1,1)
loglog(omega(1:length(amplitude_ratio)),amplitude_ratio)
ylim([10^-1 10^1])
xlim([10^0 10^2])
ylabel('Magnitude')
xlabel('Frequency [rad/s]')
title('Linear Actuator Position Response')
set(gca,'fontsize', 14)
grid on

subplot(2,1,2)
semilogx(omega(1:length(amplitude_ratio)),(phase_lag*180/pi))
ylim([-360 0])
xlim([10^0 10^2])
ylabel('Phase [degrees]')
xlabel('Frequency [rad/s]')
%text(1.25,-225,['NOTE: Nyquist freq. = ~',num2str(round(nyquist_freq)),'
rad/s'],'FontSize',12)
set(gca,'fontsize', 14)
grid on
```

2. **Thruster Bode Plot**

```
close all
clear all
load thruster_calib.mat

% Define the frequency range to test the arm
%omega = 10.^(0.5:0.01:2);
omega = 0.1*2*pi; % for testing
mag  = zeros(1,1);
phase = zeros(1,1);
phase_lag = zeros(1,1);
amplitude_ratio = zeros(1,1);

%% Perform frequency response

for j=1:length(omega)

    % Initialize MOOS
    mexmoos('CLOSE');
    pause(1);
    disp('Initializing MOOS...')
    mexmoos('init','SERVERHOST','localhost','SERVERPORT','9000');
    pause(1);
    disp('Complete!')

    %% Find initial force readings:
    mexmoos('REGISTER','Fz',0);

    f = zeros(1,1);
    f0 = zeros(1,1);
    ft = zeros(1,1);

    % Initialize index variables
    force_idx = 1;
    force_toc = 0;
    loop_toc = 0;

    % Centering Linear Actuators
```

```
disp('Finding initial values...')
initialized = 0;
tic

while (initialized==0)
   msgs=mexmoos('FETCH');

   if (~isempty(msgs))
      for k=1:length(msgs)
         if(strcmp(msgs(k).KEY,'Fz') && toc-force_toc >= 0.005)
            force_toc = toc;
            ft(force_idx)=toc;
            f(force_idx)=msgs(k).DBL;
            force_idx = force_idx + 1;
         end
      end
   end


   if toc > 10
      initialized = 1;
   end
end

f0 = mean(f);
f_rate = mean(diff(ft));
disp('Initialized!')
pause(1)
%

%% Analyze Linear Actuators

% Initialize MOOS
mexmoos('CLOSE');
pause(1);
disp('Initializing MOOS...')
mexmoos('init','SERVERHOST','localhost','SERVERPORT','9000');
pause(1);
disp('Complete!')
```

```
disp('Ready?')
pause(3)


%Register variables for thruster voltage and current values
mexmoos('REGISTER','Fz',0);
mexmoos('REGISTER','ROBOTEQ_CH2_CURRENT',0);

% Initialize variables for joint values
f = zeros(1,1);
ft = zeros(1,1);
q = zeros(1,1);
qt = zeros(1,1);
fl = zeros(1,1);
force_idx = 1;
current_idx = 1;


msgs=mexmoos('FETCH');

% Test Thrusters
disp('Running Force Feedback...')
exit = 0;
tic;
cmd = zeros(1,1);

while (exit==0)
   msgs=mexmoos('FETCH');


   if (toc <1)
      mexmoos('NOTIFY','DESIRED_POWER_CH2',0);
   end




   if (~isempty(msgs) && toc > 1)
      for k=1:length(msgs)
```

```
            if(strcmp(msgs(k).KEY,'Fz'))
               i = -sin((toc-1)*omega(j))*10;
               %
               if i<0
                  command =
(-P_lhp(2)-sqrt((P_lhp(2)).^2-4.*P_lhp(1).*(P_lhp(3)+i))./(2.*P_lhp(1));
               elseif i>0
                  command =
(-P_rhp(2)-sqrt((P_rhp(2)).^2-4.*P_rhp(1).*(P_rhp(3)+i))./(2.*P_rhp(1));
               else command = i;
               end
               %}
               mexmoos('NOTIFY','DESIRED_POWER_CH2',command);
               cmd(force_idx) = -i;
               f(force_idx)=msgs(k).DBL-f0;
               ft(force_idx)=toc-1;
               force_idx = force_idx + 1;
            elseif(strcmp(msgs(k).KEY,'ROBOTEQ_CH2_CURRENT'))
               q(current_idx)=msgs(k).DBL;
               qt(current_idx)=toc-1;
               current_idx = current_idx + 1;
            end
         end
      end


      if (toc > 15*2*pi/omega(j))
         mexmoos('NOTIFY','DESIRED_POWER_CH2',0);
         exit = 1;
      end

      pause(0.01);

   end
   disp('Complete!')

   mexmoos('CLOSE');

   %% Frequency Analysis
```

```
force=f;
figure(2)
plot(ft,cmd,ft,force)
xlabel('Time (s)')
ylabel('Force (N)')
grid on
legend('Command (Zero-Mean)','Response (Zero-Mean)')
title('Time Series of Thruster Force')



%
% Prepare data for FFT
window_ft=sin(pi*ft/ft(end));
comFFT=padarray((cmd-mean(cmd)).*window_ft,[0 length(cmd)]);
potFFT=padarray((force-mean(force)).*window_ft,[0 length(f)]);

npts = length(comFFT);
NFFT = 2^nextpow2(npts);
Fs = npts/(ft(end)-ft(1));

f = Fs/2*linspace(0,1,NFFT/2+1);

% take the FFT
X=fft(comFFT,NFFT)/npts;
Y=fft(potFFT,NFFT)/npts;


% Calculate the numberof unique points

figure(3)
subplot(211);
plot(f,2*abs(X(1:NFFT/2+1)));
title('X(f) : Magnitude response');
ylabel('|X(f)|')
subplot(212)
plot(f,2*abs(Y(1:NFFT/2+1)));
title('Y(f) : Magnitude response')
```

```
    xlabel('Frequency (Hz)');
    ylabel('|Y(f)|')

    figure(4)
    subplot(211)
    plot(f,angle(X(1:NFFT/2+1)));
    title('X(f) : Phase response');
    ylabel('Phase (rad)');
    subplot(212)
    plot(f,angle(Y(1:NFFT/2+1)));
    title('Y(f) : Phase response');
    xlabel('Frequency (Hz)');
    ylabel('Phase (rad)');

    % Determine the max value and max point.
    % This is where the sinusoidal
    % is located. See Figure 2.
    %{
    [~, idx_x] = max(abs(X));
    [~, idx_y] = max(abs(Y));
    X(idx_x)=0;
    Y(idx_y)=0;
    %}
    [mag_x, idx_x] = max(abs(X));
    mag_y = abs(Y(idx_x));
    % determine the phase difference
    % at the maximum point.
    px = angle(X(idx_x));
    py = angle(Y(idx_x));
    phase_lag(j) = py - px
    amplitude_ratio(j) = rms((force-mean(force)))/rms(cmd)
    % determine the amplitude scaling
    %amplitude_ratio(j) = mag_y/mag_x
   %}
end

for l=1:length(phase_lag)
    if phase_lag(l)<-180
        phase_lag(l) = phase_lag(l)+2*pi;
```

```
    end
    if phase_lag(l)>0
        phase_lag(l) = phase_lag(l)-2*pi;
    end
end

nyquist_freq = 2*pi/(mean(diff(ft)));

% Make Bode Plot
figure(5)

subplot(2,1,1)
loglog(omega(1:length(amplitude_ratio)),amplitude_ratio)
ylim([10^-1 10^1])
xlim([10^0 10^2])
ylabel('Magnitude')
xlabel('Frequency [rad/s]')
title('Thruster Force Response')
set(gca,'fontsize', 14)
grid on

subplot(2,1,2)
semilogx(omega(1:length(amplitude_ratio)),(phase_lag*180/pi))
ylim([-360 0])
xlim([10^0 10^2])
ylabel('Phase [degrees]')
xlabel('Frequency [rad/s]')
%text(1.25,-225,['NOTE: Nyquist freq. = ~',num2str(round(nyquist_freq)),'
rad/s'],'FontSize',12)
set(gca,'fontsize', 14)
grid on
```

3. **Impedance Controller**

```
%% This controller manages the endpoint impedance of Dexter ASV
%Revision History
% 12/6/15 4:52 PM - Reworked to reflect actuations needed for single
% mass-spring-dashpot system

close all
clear all
format long
load input_calibration.mat

% Define constants
op_freq = 1*2*pi; %natural frequency of system
zeta = 0.0443;
xa_max = 0.2032; % 8-inch stroke length of linear actuator
ft_max = 10; % maximum thrust of thrusters

x1_scale = 0.0820;% m/V
v1_scale = 1.8080;% thruster command to raft speed cmd/(V/s)
xa_scale = 0.0462;% m/V
va_scale = 14.7051;% linear actuator command to linear actuator speed cmd/(V/s)


%% Define Model Parameters

% Actual System
m = 0; % kg - mass of vehicle
b = 0; % kg/s - damping of vehicle


% Ideal System
mi = 500; %50; % kg - ideal mass
ki = mi*op_freq^2; % N/m - ideal stiffness
bi = (2*sqrt(ki*mi))*zeta;%25; % kg/s - ideal damping


% Calculate natural frequency and damping ratio
w0=sqrt(ki/mi); %undamped natural frequency
%zeta = bi/(2*sqrt(ki*mi));%damping ratio
```

```
wd = sqrt(ki/mi)*sqrt(1-zeta^2); %frequency at which damped sys oscillates

%% Create State Space Model of System

A = [0 1; -ki/mi -bi/mi];
B = [0; 1/mi];
C = [1 0; 0 1];
D = [0; 0];

T = 0.135;
s = tf('s');
sys_delay = exp(-s*T);

sys = ss(A,B,C,D,'statename',{'x','x_dot'},'inputname',{'fl'},'outputname',{'x_dot'});

sys_model = sys*sys_delay;

%% Create a Discete Time Version of the Controller

t_step = 0.02 ; % controller time step
ss_d = c2d(ss(A,B,C,D),t_step);


%% Initialize controller variables

% conroller variables
ft_cmd = zeros(1,1); % thruster command
xa_cmd = zeros(1,1); % actuator command
x1_c = 2.5; % raft center point (V)
xa_c = 3; % actuator center point (V)
x1_idx = 1; % Initialize index variables
xa_idx = 1; % Initialize index variables
fl_idx = 1; % Initialize index variables
x1_0 = zeros(1,1);
xa_0 = zeros(1,1);
fl_0 = zeros(1,1);
ft_bias = -65; % Force bias commeand to keep raft in neutral position
roboteq_toc = zeros(1,1);
crio_toc = zeros(1,1);
```

```
loop_t = zeros(1,1);
loop_toc = zeros(1,1);
loop_idx = 1;
X_state = [0 0]';
Y_state = zeros(size(ss_d.c,2),1);


% inputs
ft = zeros(1,1); % thruster force
fl = zeros(1,1); % load cell force
xref = -.50; % reference position
xdotref = zeros(1,1); % reference velocity
xa = zeros(1,1); % actuator position
xadotdot = zeros(1,1); % actuator acceleration

%% Initialize MOOS and Arduino

mexmoos('CLOSE');
pause(1);
disp('Initializing MOOS...')
mexmoos('init','SERVERHOST','localhost','SERVERPORT','9000');
pause(1);
disp('Complete!')

disp('Connecting to Arduino...')
a = arduino('/dev/ttyACM0','uno');
%a = arduino('/dev/tty.usbserial-A800eUKe','uno');
disp('Connected!')

%% Initialize actuators and sensors and variables

% Register for linear actuator string potentiometer voltage and load cell force
mexmoos('REGISTER','ROBOTEQ_AIN1_VOLTS',0);
mexmoos('REGISTER','Fz',0);


% Center Linear Actuator
disp('Centering linear actuators...')
zeroed = 0;
```

```
while (zeroed==0)
    msgs=mexmoos('FETCH');
    mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
    %disp(xa(end));

    if (~isempty(msgs))
        for k=1:length(msgs)
            if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS'))
                xa(xa_idx)=msgs(k).DBL;
                xa_idx = xa_idx + 1;
            end
        end
    end

    if xa(end) > xa_c && xa(end)~=0
        disp('Too far...')
        while (zeroed == 0)
            msgs=mexmoos('FETCH');
            mexmoos('NOTIFY','DESIRED_POWER_CH1',-50); % Pulls inward
            if (~isempty(msgs))
                for k=1:length(msgs)
                    if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS'))
                        xa(xa_idx)=msgs(k).DBL;
                        xa_idx = xa_idx + 1;
                    end
                end
            end
            if xa(end)<= xa_c
                zeroed = 1;
                mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
            end

            pause(0.02);
        end

    elseif xa(end) < xa_c  %&& x2(end)~=0
        disp('Too close...')
        while (zeroed == 0)
```

```
        msgs=mexmoos('FETCH');
        mexmoos('NOTIFY','DESIRED_POWER_CH1',50); %Pushes outward
        if (~isempty(msgs))
           for k=1:length(msgs)
              if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS'))
                 xa(xa_idx)=msgs(k).DBL;
                 xa_idx = xa_idx + 1;
              end
           end
        end
        if xa(end)>= xa_c
           zeroed = 1;
           mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
        end
        pause(0.02);
     end
  end

  pause(0.02);

end
disp('Linear actuator centered!')

pause(1)

% Center Vehicle
disp('Centering vehicle...')
zeroed = 0;

while (zeroed==0)
  msgs=mexmoos('FETCH');
  mexmoos('NOTIFY','DESIRED_POWER_CH2',ft_bias);

  x1(x1_idx)=readVoltage(a,0);
  x1_idx = x1_idx + 1;

  if x1(end) > x1_c && x1(end)~=0
     disp('Too far...')
     while (zeroed == 0)
```

```
        msgs=mexmoos('FETCH');
        mexmoos('NOTIFY','DESIRED_POWER_CH2',ft_bias+15);
        x1(x1_idx)=readVoltage(a,0);
        x1_idx = x1_idx + 1;
        if x1(end)<= x1_c
            zeroed = 1;
            mexmoos('NOTIFY','DESIRED_POWER_CH2',ft_bias);
        end

        pause(0.02);
    end

elseif x1(end) < x1_c  && x1(end)~=0
    disp('Too close...')
    while (zeroed == 0)
        msgs=mexmoos('FETCH');
        mexmoos('NOTIFY','DESIRED_POWER_CH2',ft_bias-5);
        x1(x1_idx)=readVoltage(a,0);
        x1_idx = x1_idx + 1;
        if x1(end)>= x1_c
            zeroed = 1;
            mexmoos('NOTIFY','DESIRED_POWER_CH2',ft_bias);
        end
        pause(0.02);
    end
end
end
disp('Vehicle centered!')

% Initialize Variables
disp('Initializing variables...')
zeroed = 0;
x1 = zeros(1,1);
xa = zeros(1,1);
fl = zeros(1,1);
x1_idx = 1; % Initialize index variables
xa_idx = 1; % Initialize index variables
fl_idx = 1; % Initialize index variables
tic;
```

```
while (toc < 10)
   msgs=mexmoos('FETCH');
   mexmoos('NOTIFY','DESIRED_POWER_CH2',ft_bias);

   x1(x1_idx)=readVoltage(a,0);
   x1_idx = x1_idx + 1;

   if (~isempty(msgs))
      for k=1:length(msgs)
         if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS') && toc-roboteq_toc >= 0.01)
            roboteq_toc = toc;
            xa(xa_idx)=msgs(k).DBL;
            xa_idx = xa_idx + 1;

         elseif(strcmp(msgs(k).KEY,'Fz') && toc-crio_toc >= 0.005)
            crio_toc = toc;
            fl(fl_idx)=msgs(k).DBL;
            fl_idx = fl_idx + 1;
         end
      end
   end


   x1_0 = mean(x1)*x1_scale;
   xa_0 = mean(xa)*xa_scale;
   fl_0 = mean(fl);

end

disp('Complete!')
disp('Now wait...')
pause(10)


%% Run the Controller

% Analyze Linear Actuators
disp('Running analysis...')
```

```
% RESET VARIABLES
x1_idx = 1; % Initialize index variables
xa_idx = 1; % Initialize index variables
fl_idx = 1; % Initialize index variables
x1 = zeros(1,1);
xa = zeros(1,1);
fl = zeros(1,1);
roboteq_toc = zeros(1,1);
xa_t = zeros(1,1);
crio_toc = zeros(1,1);
fl_t = zeros(1,1);
stop = 0;

tic
while (stop==0)
   msgs=mexmoos('FETCH');
   mexmoos('NOTIFY','DESIRED_POWER_CH2',100);
   %mexmoos('NOTIFY','DESIRED_POWER_CH1',
100*sin(w0*toc)*exp(-zeta*w0*toc));
   %

   if (~isempty(msgs))
      for k=1:length(msgs)
         if(strcmp(msgs(k).KEY,'ROBOTEQ_AIN1_VOLTS') && toc-roboteq_toc >= 0.01)
            roboteq_toc = toc;
            xa_t(xa_idx) = roboteq_toc;
            xa(xa_idx)=msgs(k).DBL;
            xa_idx = xa_idx + 1;


         elseif(strcmp(msgs(k).KEY,'Fz') && toc-crio_toc >= 0.005)
            crio_toc = toc;
            fl_t(fl_idx) = crio_toc;
            fl(fl_idx)=msgs(k).DBL;
            fl_idx = fl_idx + 1;
         end
      end
   end
```

```
   if toc - loop_toc >= t_step
      loop_toc = toc;
      loop_t(loop_idx)=loop_toc;
      %x1(x1_idx)=readVoltage(a,0);
      %x1_idx = x1_idx + 1;
      %
      U_state(:,loop_idx) = [fl(end)-fl_0];
      %X_state(:,loop_idx+1) = ss_d.a*[xa(end)*xa_scale-xa_0; X_state(2,loop_idx)] +
ss_d.b*U_state(:,loop_idx);
      X_state(:,loop_idx+1) = ss_d.a*X_state(:,loop_idx) + ss_d.b*U_state(:,loop_idx);
      Y_state(:,loop_idx+1) = ss_d.c*X_state(:,loop_idx);

      xadot_cmd =
Y_state(2,loop_idx+1)*va_scale/xa_scale-5000*((xa_scale*xa(end)-xa_0)-X_state(1,loo
p_idx));
      %xadot_cmd = max(Y_state(2,:))*va_scale/xa_scale;
      %
      if xadot_cmd<0
         xadot_cmd =
neg_xa_cmd(1).*xadot_cmd.^6+neg_xa_cmd(2).*xadot_cmd.^5+neg_xa_cmd(3).*xadot
_cmd.^4+...

neg_xa_cmd(4).*xadot_cmd.^3+neg_xa_cmd(5).*xadot_cmd.^2+neg_xa_cmd(6).*xadot
_cmd.^1+...
            neg_xa_cmd(7).*xadot_cmd.^0;
      elseif xadot_cmd>0
         xadot_cmd =
pos_xa_cmd(1).*xadot_cmd.^6+pos_xa_cmd(2).*xadot_cmd.^5+pos_xa_cmd(3).*xadot
_cmd.^4+...

pos_xa_cmd(4).*xadot_cmd.^3+pos_xa_cmd(5).*xadot_cmd.^2+pos_xa_cmd(6).*xadot
_cmd.^1+...
            pos_xa_cmd(7).*xadot_cmd.^0;
      else xadot_cmd = xadot_cmd;
      end
      %}
      display(xadot_cmd);
      %mexmoos('NOTIFY','DESIRED_POWER_CH1', xadot_cmd);
```

```
        mexmoos('NOTIFY','DESIRED_POWER_CH1', xadot_cmd);
        %display(5000*((xa_scale*xa(end)-xa_0)-X_state(1,loop_idx)));
        %mexmoos('NOTIFY','DESIRED_POWER_CH1',
100*sin(w0*toc)*exp(-zeta*w0*toc));
        %}

        loop_idx = loop_idx +1;
    end
     %}

    if toc > 15
        stop = 1;
        display(mean(diff(loop_t)))
    end

    %{
    if xa(end)>4.75 || xa(end)<.25 || x1(end)>4.75 || x1(end)<.25
        mexmoos('NOTIFY','DESIRED_POWER_CH1',0);
        mexmoos('NOTIFY','DESIRED_POWER_CH2',0);
        stop = 1;
    end
    %}
end
disp('Complete!')


mexmoos('CLOSE');

%% Plot some results
figure(1)
plot(xa_t-.05,xa_scale*(xa)-xa_0);%+0.2-(0.2/10)*xa_t)
hold on
plot(loop_t+0.135,X_state(1,1:end-1),'LineWidth',3)
hold off
xlabel('time (s)')
ylabel('position (m)')
legend('data','model')
title('Linear Actuator Position Step Response')
grid on
```

```
xlim([0 4])
```