Autonomous Underwater Vehicle Final Project: ROB 599

Christian Foreman
University of Michigan
cjforema@umich.edu

Nithish Kumar
University of Michigan
nithishk@umich.edu

Max Rucker
University of Michigan
mruck@umich.edu

Jack Wischmeyer
University of Michigan
jmwisch@umich.edu

I. Introduction

Our group decided to model an autonomous underwater vehicle (AUV) for this project. For our model, we utilize the AUV II created by the Naval Postgraduate School [1]. This is a submersible that consists of two rudders on the back, multiple controllable fins, and a hull with a programmable computer and battery allowing for autonomous control of the vehicle. In this report, we will give an overview of our current work with modeling the motion of the AUV with linearized equations across different subsystems for control.

II. Background

A. Reference Frames

- The global(Earth) fixed frame has its x and y axes in a horizontal plane, and a z-axis (positive downward). This is an inertial frame.
- The body fixed frame has its origin at the center of mass of the AUV. The x-axis points forward, the z-axis points downward, and the y-axis completes the right-hand rule.

B. Assumptions

Assumptions were taken from [1], [2], and Professor Girard's Slides:

1) The AUV II is neutrally buoyant.
2) The AUV is symmetrically loaded in the transverse direction and the vertical center of buoyancy is midway between the top and bottom of the AUV II.
3) The counter-rotating propellers produce no yaw moment.
4) The products of inertia about the body system are zero because the AUV II possesses two axes of symmetry.
5) The AUV II acceleration and deceleration rates are small enough so that propeller slip can be neglected.
6) The effect of cross-coupled hydrodynamic coefficients can be neglected in most cases, again because of the AUV II geometric symmetry.
7) Gravity is constant and uniform.
8) Current is constant and in the lateral x, y plane. We may investigate a variable current in the future.
9) The mass matrix is nominally constant and invertible.

C. Variables and Parameters

There are a ton of different variables and parameters to describe the motion of the AUV. Descriptions of these variables can be found in the appendix as well as [1]–[3].

III. Equations of Motion

Based on our assumptions and [1, 2], we get the following equations of motion for our AUV:

Heave Equation of Motion

$$\begin{aligned} m\dot{w} - mx_G\dot{q} - Z_{\dot{q}}\dot{q} - Z_{\dot{w}}\dot{w} &= muq - mvp \\ -mx_Gpr + mz_G(p^2+q^2) &+ Z_quq + Z_wuw \\ +u^2(Z_{\delta s}\delta_s + Z_{\delta b}\delta_b) - \int_{x_{\text{tail}}}^{x_{\text{nose}}} &[Q(x)]\frac{(w-xq)}{U_{cf}(x)}dx \end{aligned} \tag{1}$$

Surge Equation of Motion

$$\begin{aligned} m\dot{u} + mz_G\dot{q} - X_{\dot{u}}\dot{u} &= mvr - mwq + mx_Gq^2 \\ +mx_Gr^2 - mz_Gpr + X_{rr}r^2 &+ X_{vv}v^2 + u^2X_{\text{prop}} \\ +u^2(X_{\delta s\delta s}\delta_s^2 + X_{\delta b\delta b}\delta_b^2 + X_{\delta_{rb}\delta_{rb}}\delta_{rb}^2 &+ X_{\delta_{rs}\delta_{rs}}\delta_{rs}^2) \end{aligned} \tag{2}$$

Sway Equation of Motion

$$\begin{aligned} m\dot{v} + mx_G\dot{r} - mz_G\dot{p} - Y_{\dot{r}}\dot{r} - Y_{\dot{v}}\dot{v} &= \\ mwp - mur - mx_Gpq - mz_Gqr &\\ +Y_rur + Y_vuv + u^2(Y_{\delta_{rb}}\delta_{rb} + Y_{\delta_{rs}}\delta_{rs}) &\\ -\int_{x_{\text{tail}}}^{x_{\text{nose}}}[Q(x)]\frac{(v+xr)}{U_{cf}(x)}dx & \end{aligned} \tag{3}$$

Roll Equation of Motion

$$\begin{aligned} I_X\dot{p} - mz_G\dot{v} - K_{\dot{p}}\dot{p} &= \\ (I_Y - I_Z)qr + mz_Gur - mz_Gwp &+ \\ K_pup - (z_GW - z_B)cos\theta sin\phi & \end{aligned} \tag{4}$$

Pitch Equation of Motion

$$\begin{aligned} I_y\dot{q} - mx_G\dot{w} + mz_G\dot{u} - M_{\dot{q}}\dot{q} - M_{\dot{w}}\dot{w} &= \\ (I_z - I_x)pr - mx_Guq + mx_Gvp &\\ +mz_Gvr - mz_Gwq + M_quq + M_wuw &\\ +u^2(M_{\delta s}\delta_s + M_{\delta b}\delta_b) - (z_GW - z_BB)sin\theta &\\ -\int_{x_{\text{tail}}}^{x_{\text{nose}}}[Q(x)]\frac{(w-xq)}{U_{cf}(x)}xdx & \end{aligned} \tag{5}$$

YAW EQUATION OF MOTION

$$\begin{aligned} I_z\dot{r} - mx_G\dot{v} - N_{\dot{r}}\dot{r} - N_{\dot{v}}\dot{v} = \\ (I_x - I_y)pq - mx_Gur + mx_Gwp + N_rur + N_vuv \\ +u^2(N_{\delta_{rb}}\delta_{rb} + N_{\delta_{rs}}\delta_{rs}) + (x_GW - x_BB)cos\theta sin\phi \\ +u^2N_{\text{prop}} - \int_{x_{\text{tail}}}^{x_{\text{nose}}}[Q(x)]\frac{(w-xq)}{U_{cf}(x)}xdx \end{aligned} \quad (6)$$

Where:

$$Q(x) = C_{Dy}h(x)(v+xr)^2 + C_{Dz}b(x)(v+xr)^2$$
$$U_{cf}(x) = [(v+xr)^2 + (w-xq)^2]^(1/2)$$

IV. LINEARIZATION

We split up the linearization of the AUV into separate functions of speed, diving, and steering control. This splits the system into noninteracting (lightly interacting) subsystems which makes the system easier to analyze. The following briefly describes what states each system controls:

- Speed: $[u(t)]$
- Steering: $[v(t), r(t), \psi(t)]$
- Diving: $[q(t), \theta(t), Z(t)]$

The following sections describe the linearized systems for each of these controls taken from [2]. These are linearized along a nominal flight path. Since we are not yet on the control of our systems, we will assume no/constant control for now.

A. Speed

The speed of the AUV controls its longitudinal motion. From [2] we get that:

$$\dot{u}(t) = -\alpha u(t)|u(t)| + (\alpha\beta)n(t)|n(t)| \quad (7)$$

Where:

$$\alpha = \frac{\rho L^2 C_d}{2m + \rho L^3 X_{\dot{u}}}, \quad \beta = \frac{u_0^2}{n_0^2}$$

With $u_0 = 1.832\frac{m}{s}$ being the nominal velocity of the AUV and $n_0 = 52.359\frac{rads}{s}$ being the nominal propeller speed of the AUV. Because we are not using the the designed controller from [2], we ignore the $n(t)|n(t)$ component of the linearization.

We attempt to linearize $\dot{u}(t)$ and get the following:

$$\dot{u}(t) = -2\alpha u(t) + 2\alpha\beta n(t) \quad (8)$$

With this linearization we can model the speed of our AUV. Figure 1 shows the function of $u(t)$ with different initial speeds. As time goes on, we can see the speed converge to zero for varying initial speeds.

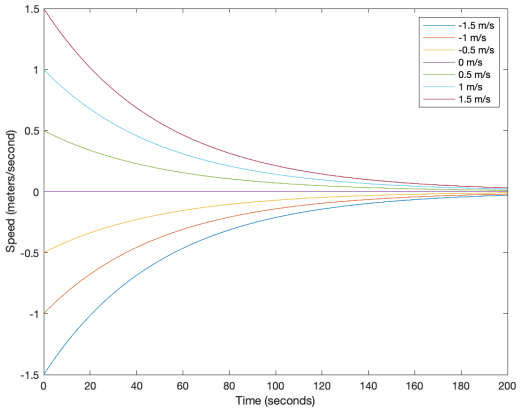

Fig. 1. Speed $u(t)$ with different initial speeds

B. Vertical Motion

From [2], the linearized diving system dynamics are given by the following system of equations:

$$\begin{bmatrix}\dot{q}(t)\\ \dot{\theta}(t)\\ \dot{Z}(t)\end{bmatrix} = \begin{bmatrix}-0.7 & -0.3 & 0\\ 1 & 0 & 0\\ 0 & -u_0 & 0\end{bmatrix}\begin{bmatrix}q(t)\\ \theta(t)\\ Z(t)\end{bmatrix} + \begin{bmatrix}0.035\\ 0\\ 0\end{bmatrix}\delta_s(t) \quad (9)$$

In [2], they use $\delta_s(t)$ as the control input. Since we are currently not implementing control for our system we can ignore $\delta_s(t)$. With this linearization, we can model the diving motion of the AUV. We show the depth trajectories in Figure 2 with constant speed $u_0 = 1.832\frac{m}{s}$. In every initial condition, we set the AUV's depth to 1 meter and vary the initial pitch (θ) and angular velocity's pitch (q). We can see that, with no changes in the pitch, the depth stays constant. If we vary the pitch, the AUV will either move up or down depending on the magnitude of the pitch or the initial angular velocity. Over time, the pitch angle levels out to 0.

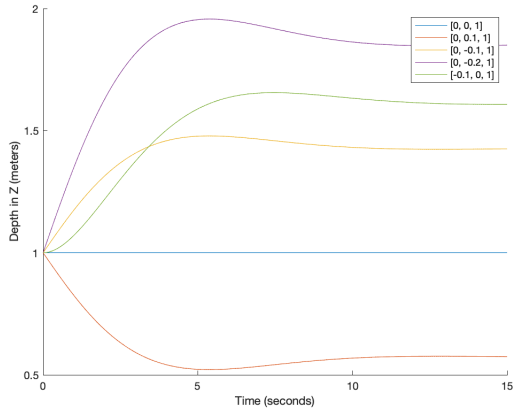

Fig. 2. Depth trajectories with different initial conditions $[q, \Theta, Z]$.

In Figure 3, we simulate the pitch angle θ of the AUV. For this case, we don't modify the Z value in the initial condition, as the depth does not impact the pitch. For the cases where we modify the initial θ value, we see that the pitch goes to zero and slightly overshoots it before stabilizing at zero. When we

modify the initial angular velocity in pitch (q), we see that the pitch angle increases, reaches a maximum as the angular velocity in pitch becomes 0, and then the pitch converges back to a value of zero.

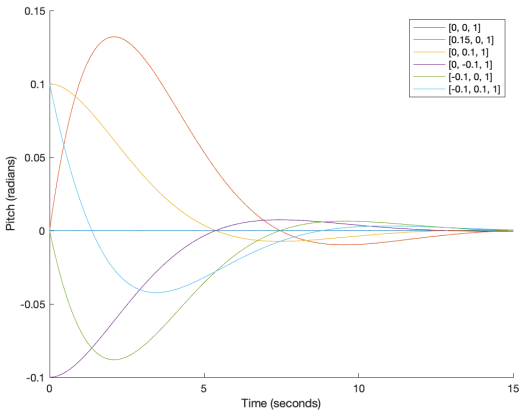

Fig. 3. Pitch trajectories with different initial conditions $[q, \Theta, Z]$.

In Figure 4, we are measuring the angular velocity of the pitch q of the AUV. The depth of our AUV does not affect the change in the angular velocity of the pitch. The main two initial conditions being changed are the angular velocity of the pitch and the pitch itself. With these two initial conditions being set in various ways, we can see that the values are consistently attracted to zero and slightly overshoot the value, but then slowly stabilize to a value of zero.

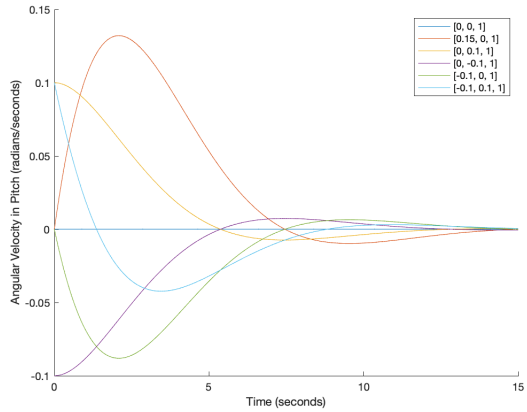

Fig. 4. Angular velocity in pitch with different initial conditions $[q, \Theta, Z]$.

C. Lateral Motion

The lateral motion of the AUV is based on the steering control. The position of the vehicle can be determined by:

$$\begin{aligned} \dot{X}(t) &= u(t)cos\psi(t) - v(t)sin\psi(t) + u_{cx} \\ \dot{Y}(t) &= u(t)sin\psi(t) + v(t)cos\psi(t) + u_{cy} \end{aligned} \tag{10}$$

We can express the linearization of the lateral motion with:

$$\begin{bmatrix} \dot{v}(t) \\ \dot{r}(t) \\ \dot{\psi}(t) \end{bmatrix} = A_2 \begin{bmatrix} v(t) \\ r(t) \\ \psi(t) \end{bmatrix} + b_2\delta_r(t) \tag{11}$$

Where:

$$A_2 = \begin{bmatrix} \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}^{-1} \times \begin{bmatrix} Y_1 & Y_2 \\ N_1 & N_2 \end{bmatrix} & \begin{matrix} 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{bmatrix} \tag{12}$$

and

$$b_2 = \begin{bmatrix} \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}^{-1} \times \begin{bmatrix} Y_3 \\ N_3 \end{bmatrix} \\ 0 \end{bmatrix} \tag{13}$$

Where the values of m_1, m_2, m_3, m_4 are provided in [3] and the values of $Y_1, Y_2, Y_3, N_1, N_2, N_3$ are found in [2].

With this linearization done, we can view trajectories based on different initial conditions. Again, since we are not paying attention to the controller from [2], we ignore $\delta_r(t)$. In Figure 5, we are modeling the yaw with different initial conditions. From these initial conditions, the angular velocity of yaw changes directly acts on the rate of change in yaw. We can also notice that sway velocity has an inverse relationship with yaw, causing a decrease in yaw as the sway velocity increases.

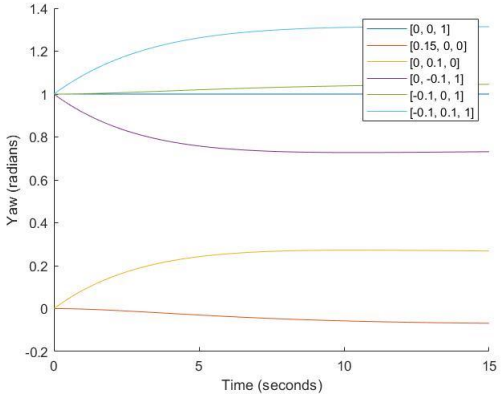

Fig. 5. Yaw with different initial conditions $[v, r, \Psi]$.

In Figure 6, we are measuring the angular velocity of the yaw of the AUV. Here, we can see that angular velocity only depends on the initial conditions set for the angular velocity and slowly returns to a value of 0 after a bit of time.

In Figure 7, we are measuring the Sway velocity of the AUV. From looking at this figure, we can see that the Sway velocity is slightly affected by the Yaw angular velocity but eventually steadies back to a value of 0 after a bit of time.

in Figure 8, we plot X and Y based on different values of a constant ψ with an initial speed of u_0. We can see that, based on the initial heading of yaw (ψ), we get a corresponding trajectory in that direction.

V. Stability

To find the stability of our separate subsystems, we need to calculate the eigenvalues of each equation. This can easily be done by using the "eig()" function in Matlab and inputting the A matrix of each equation. We explain the stability of each subsystem in the following.

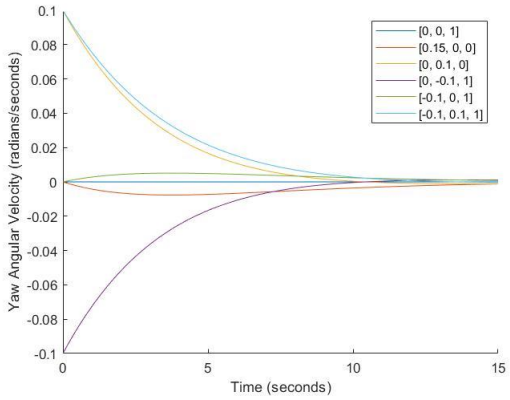

Fig. 6. Yaw with different initial conditions $[v, r, \Psi]$.

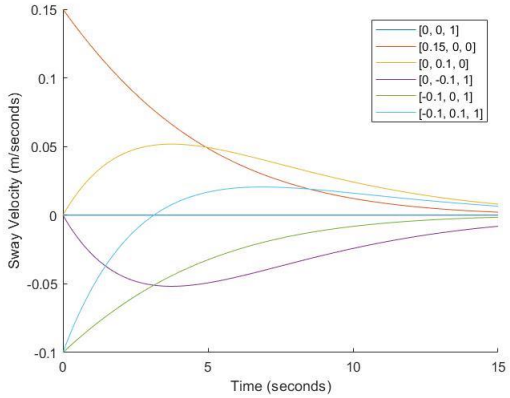

Fig. 7. Yaw with different initial conditions $[v, r, \Psi]$.

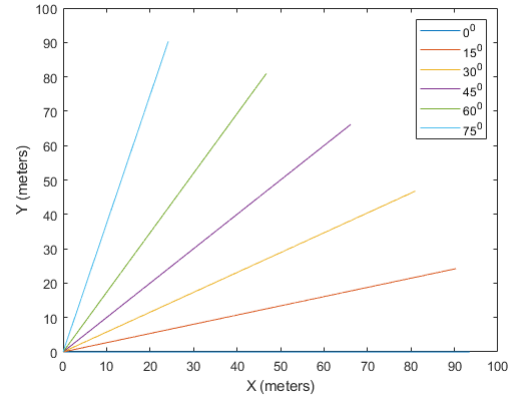

Fig. 8. X and Y Trajectories based on different values of ψ

A. Speed

The eigenvalue of $\dot{u}(t)$ gives us a value of $\lambda = -0.0195$. This corresponds to the speed being stable as the eigenvalue is negative. This eigenvalue gives us a time constant of $\tau = 51.2820s$.

B. Vertical Motion

The eigenvalues from the A matrix for vertical motion are $\lambda_{1,2} = -0.35 \pm 0.4213i, \lambda_3 = 0$. In this case, since the real component of the eigenvalues are negative, and there is an imaginary component, this system is stable and has damped oscillations. The natural frequency of this system is $\omega_0 = 0.548\frac{rads}{s}$ with a damping ratio of $\zeta = 0.639$.

C. Lateral Motion

The eigenvalues from the A matrix for the lateral motion are $\lambda_{1,2} = -0.2499 \pm 0.0927i, \lambda_3 = 0$. Since the eigenvalues have negative real components along with an imaginary value, the system is stable and has damped oscillations. The natural frequency of this system is $\omega_0 = 0.266\frac{rads}{s}$ with a damping ratio of $\zeta = 0.938$.

VI. PID Control

A. Controllability

The first step in designing a controller for our system is to determine if the state-space equations we have are controllable or not. To determine this, we used the *ctrb()* function in Matlab. This function returns the number of controllable variables within our state-space model given the A and B matrices for each system. Using this, we determined that every variable within our three models of speed, vertical motion, and lateral motion are controllable.

One thing of note is how our b matrices for the vertical and lateral motion work. In these matrices, we can see that there are variables we directly control with our control input $\delta(t)$ while other variables get canceled out by zeros in the b matrix. For example in the b matrix for vertical motion, the control part of the equation is only directly affecting $q(t)$ with a factor of 0.035 while the other variables have a 0. With this, we are only controlling the angular velocity of the pitch. While this is true, controlling the angular velocity of pitch allows us to indirectly control pitch angle, which then indirectly controls depth.

B. Designing the PID Controllers

Since we are splitting up our AUV system into three parts, we decided to create three PID loops to control each part: speed, lateral, and vertical motion. The errors for these three PID loops are defined as follows:

- Speed PID controller error: the differece between the target speed and the current speed.
- Lateral PID controller error: the difference between the desired yaw heading (yaw heading to point the sub from the current XY the target XY) and the current yaw heading.

- Vertical PID controller error: the difference between the target pitch and the current pitch.

C. Tuning

The PID tuning process is relatively simple. We started out by setting the P, I, and D terms all to zero. From here, we tuned just the P term to get a P controller. This roughly defines the response time of our system, however, just P control normally results in steady state error. To help with this steady state error, we then tune the I term. This term is normally relatively small compared to P, so we chose smaller values until we got relatively good steady state error. With just PI control, all of our controllers work pretty well. In Figure 9, you can see the different outcomes for our Pitch control when tuning our PI controllers with different K_p and K_i values.

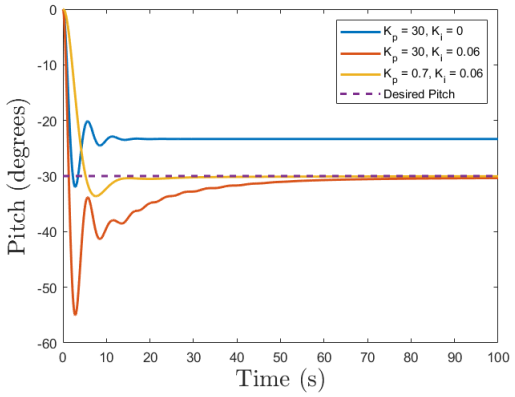

Fig. 9. Pitch PI Controller with Different K_p and K_i Values

We also explored how adding the D term to our control would help (Figure 10. However, since our PI controller was already solid, we found that adding the D term did not improve any of our controllers. Therefore, we solely stuck with PI controllers.

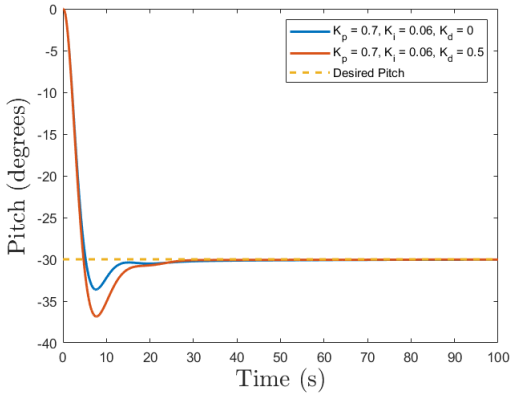

Fig. 10. Pitch PID Controller with Different K_d Values

D. Final Controllers

After tuning all of our controllers, we can define them in the form of PID controllers: $K_p e(t) + K_i \int_t^{t_0} e(t)\,dt + K_d \frac{de(t)}{dt}$

Speed Controller:

$$n(t) = 5000e(t) + 10 \int_t^{t_0} e(t)\,dt \tag{14}$$

Lateral Controller:

$$\delta_s(t) = 0.5e(t) \tag{15}$$

Vertical Controller:

$$\delta_r(t) = 0.7e(t) + 0.06 \int_t^{t_0} e(t)\,dt \tag{16}$$

We found that these values resulted in good response times and low steady state error. We noticed that in order for us to see good speed control, we needed to set the K_p to a very high value. This is an area we should further explore to make sure it is realistic. Also, for our lateral controler, we set the K_i, to zero, meaning there is no integration term. We found that, for even small values of K_i, the AUV would perform very poorly when there was ocean currents. Thus, we decided to remove the integration term for Lateral Control.

VII. Waypoint Navigation

A. Control Logic

With our tuned PID controllers we are able to implement a simple control loop to navigate to various waypoints. The overall logic of our controller looks like:

1) Determine what waypoint to navigate to.
2) Navigate to the desired depth (Z value) while maintaining the nominal speed and turning the AUV towards the desired X/Y.
3) Once the desired depth is within the threshold distance, focus on turning the AUV towards the waypoint in the XY Plane. While turning the AUV, slow down its speed to 0.
4) Once the yaw heading is within an angle threshold, speed up the AUV until it hits the waypoint.
5) Check if the position is within the threshold distance of the waypoint, if so, select a new waypoint and run these steps again. If not within the threshold, run steps 2-5 again.

In short, we first control the AUV to reach the target depth, rotate the AUV's yaw to point towards the XY lateral component, and then speed up until we hit the waypoint.

B. Selecting Waypoints

Our AUV is approximately 5.3 meters long, and it is meant for travelling large distances in the ocean, mostly in the XY plane. Thus, we chose are waypoints to be on the scale of 0 to 1000 meters and our depth to be 0 to 100 meters. To keep things consistent for this report, we will be using the order of waypoints from Table I to demonstrate our controller. Also, the AUV starts at 0 in the X and Y, with a depth of 10 (0, 0, 10). We find this list of waypoints to be a good example of how our controller acts in many situations.

Waypoint #	X	Y	Z
1	50	0	40
2	90	30	35
3	20	20	50
4	40	-20	30
5	20	0	5
6	100	50	15
7	120	30	40

TABLE I
LIST OF WAYPOINTS

One area of concern when selecting waypoints was the disregard for the maximum allowable pitch angle of the vehicle. It was observed that sequential waypoints with a dramatic change in depth would prompt the PID controller to completely flip the AUV past the target pitch angle. To remedy this, a conditional statement was added into the PID controller to manually restrict the AUV's pitch angle to a certain maximum, chosen as 30 degrees from the horizontal. With this change, the AUV model was able to transition much more smoothly and continuously between waypoints.

C. Reaching Waypoints

The condition for reaching a waypoint for our system is rather simple. We define a distance threshold, and continue navigating towards the current waypoint until our position is within the distance threshold. Initially, we chose a distance threshold with a value of 0.25 meters. However, we found that a low threshold like this would result in loopy and almost figure-8-type artifacts when the AUV was close to the goal waypoint (see Figure 11). This behavior is expected, as, since the AUV is rather large, it can be tough to make very accurate navigation. From this, we decided to increase our threshold to 3 meters. For the scale of our system, a threshold of 3 meters is still rather small, and this helps reduce the loopy artifacts seen in our navigation, see Figure 12. The speed to reach all the waypoints is also significantly reduced by a more generous distance threshold. It only takes 292 seconds for the AUV to navigate to all the waypoints with a threshold of 3 meter while it takes 1250 seconds to reach all the waypoints with a threshold of 0.25 meters.

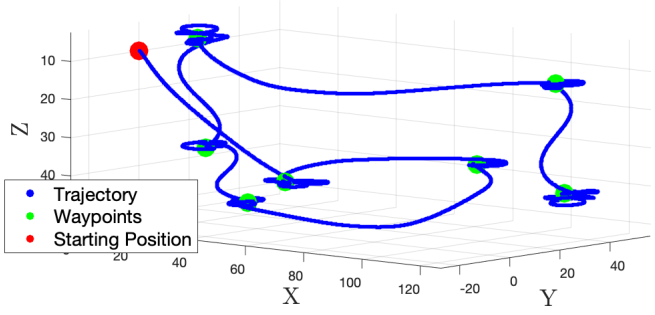

Fig. 11. Trajectory for Waypoint Navigation with Threshold = 0.25m

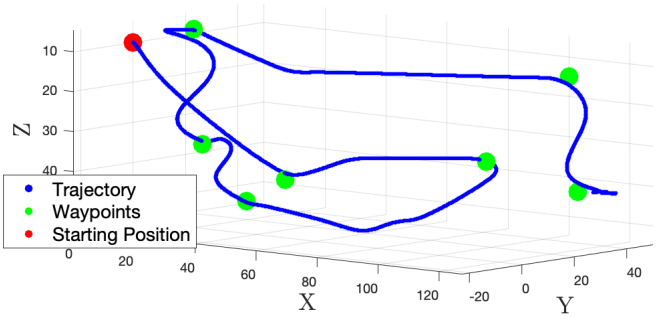

Fig. 12. Trajectory for Waypoint Navigation with Threshold = 3.0m

D. Performance With Ocean Current

So far, all of our work has been with no current or any other disturbances. Here, we briefly show how our controller performs for constant ocean currents. Figure 13 shows how our trajectory differes with and without currents. In this case, we apply a current of [0.6 0.3 0], which means there is a constant push of 0.6 meters/second on the X, 0.3 meters/second on the Y, and 0.0 meters/second on the Z of the AUV. Based on this example, we can see that our controller still performs well and reaches all the waypoints. One cool behavior we noticed is that, if the AUV is struggling to reach a waypoint with the current, it will align itself in the opposite direction of the current and then move forwards until it hits the waypoint.

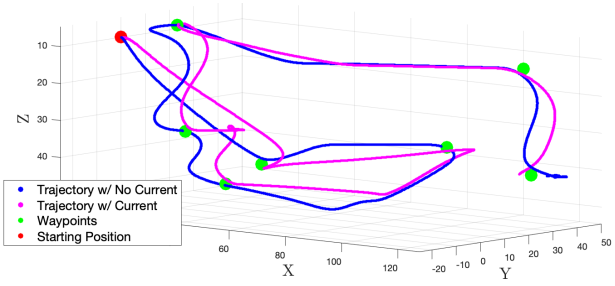

Fig. 13. Trajectories with and without Ocean Currents

VIII. REAL-TIME SIMULATION OF AUV IN 3D

The PID control loop provides the 12 outputs defined in [2], being the three displacements and three rotations in the moving frame of the vehicle and the world's fixed frame. These outputs were used to develop a three-dimensional simulation of the AUV traversing between waypoints based on the order in which they were indexed.

A simple model of the AUV was designed in Solidworks and converted into a set of three-dimensional coordinates through Matlab's importGeometry function. These data points composed a simple three-dimensional shell resembling the AUV's geometry.

For convenience, the volumetric centroid of the AUV model was selected to be the origin of the moving frame. The vertex data points were oriented around the model's centroid by subtracting the average X, Y, and Z values from the respective

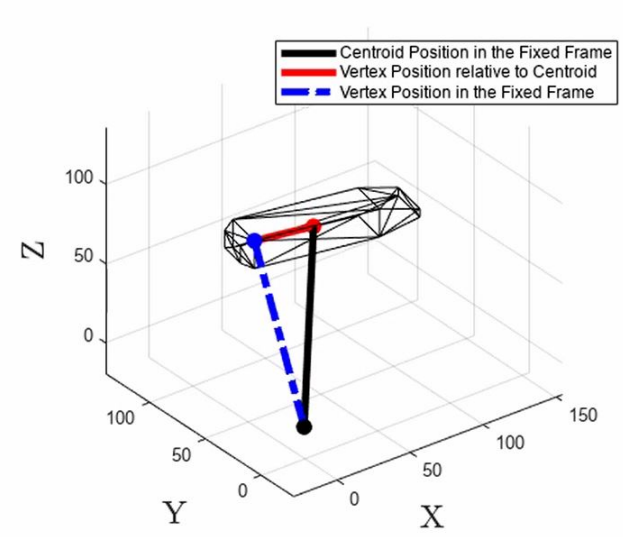

Fig. 14. Geometric Vertex Coordinates relative to the Fixed/Moving Frames

dimension for each vertex. Equation 17 illustrates the process for computing the augmented vertex coordinates for the i-th element in a set of n-total vertices:

$$\forall\, i, 1 \leq i \leq n, \exists\, V_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} - \begin{bmatrix} mean(x_1, x_2, ..., x_n) \\ mean(y_1, y_2, ..., y_n) \\ mean(z_1, z_2, ..., z_n) \end{bmatrix} \quad (17)$$

By defining the vertex data points with respect to the model's centroid, any given rotations would revolve the AUV model about it's own centroid, now the origin of the moving frame. Animating the rotations of the vehicle model required each vertex to be multiplied by the three rotation matrices defined in Equations 18, 19, and 20. However, roll was neglected in the model, so the rotation matrix A_x reduces to the identity in Equation 18 below:

$$A_x(\phi(t)) = A_x(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(0) & -sin(0) \\ 0 & sin(0) & cos(0) \end{bmatrix} = I \quad (18)$$

$$A_y(\theta(t)) = \begin{bmatrix} cos(\theta(t)) & 0 & sin(\theta(t)) \\ 0 & 1 & 0 \\ -sin(\theta(t)) & 0 & cos(\theta(t)) \end{bmatrix} \quad (19)$$

$$A_z(\psi(t)) = \begin{bmatrix} cos(\psi(t)) & -sin(\psi(t)) & 0 \\ sin(\psi(t)) & cos(\psi(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (20)$$

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}_F = A_z(\psi(t))A_y(\theta(t))I \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}_M + \begin{bmatrix} d_x(t) \\ d_y(t) \\ d_z(t) \end{bmatrix} \quad (21)$$

The pitch (θ) and yaw (ψ) angles, used to compute the rotation matrices A_y and A_z, are time-dependent, same as the terms d_x, d_y, and d_z, representing the position of the AUV centroid in the fixed frame. All five terms exist as predetermined inputs, computed and indexed over a time interval by separate ode45 functions. The outputs x_i, y_i, and z_i, with subscript F, represent the augmented coordinates of the i-th numbered vertex in the AUV's skeleton, with respect to the fixed frame. Being time-dependent, the outputs of Equation 21 enable the visual movement and rotations of the AUV model in the simulated environment. The alphaShape function was used to create the visual representation of the geometric data points in the simulation.

IX. Marsupial Planning

With waypoint navigation and a simple simulation implemented, we explored the idea of controlling multiple AUVs. The idea here is that there is a larger carrier AUV which can deploy mini vehicle AUVs to explore areas. We drew inspiration from [5] which performs marsupial planning in contested environments. We also based our work off of [6] which explores carrier-vehicle systems with TSP and [7] which provides an implementation for multiple TSP. In this report, we refer to the larger carrier/mother AUV as the carrier AUV, and the smaller explorer/daughter AUVs as vehicle AUVs.

A. Assumptions

There are a few assumptions we made in implementing our marsupial system.

1) The carrier AUV and vehicle AUVs all share the same dynamics as described so far in this paper.
2) The carrier AUV can deploy and recall the vehicle AUVs instantly.
3) Current is non existent for multi-AUV planning.

B. Multi-Traveling Salesman Problem

The key challenge in implementing multi-AUV planning was determining which waypoints each vehicle AUV should navigate to in order to efficiently search the space. We explored Dijkstra's Algorithm and many implementations of the Multiple Travelling Salesman Problem (MTSP). We decided to use a variant of MTSP from [7]. This MTSP uses a genetic algorithm with a multi-chromosome representation to find near-optimal paths for minimum salesman subject to constraints (minimum distance, cities per salesman, etc.). Figure 15 is an example of what the example paths look like after running MTSP on 20 waypoints for four salesman (AUVs). We use these paths with our AUV dynamics to navigate to the waypoints.

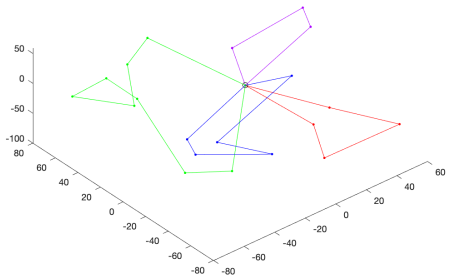

Fig. 15. Planned paths for 4 AUVs to reach 20 waypoints

C. Problem Setup

Here is a brief description of how we demonstrate our multi-AUV planning. The main carrier AUV follows the waypoints as described in section VII (see figure 16). At each one of these green waypoints, the carrier AUV does a rough scan of its environment, and detects potential areas of interest (cyan points in figure 17). The carrier AUV then deploys these mini vehicle AUVs to explore the areas of interest (cyan points). When the vehicle AUVs reach these areas of interest (cyan points) they are able to do a more in-depth scan and detect points of interest (red points). These red points then need to be searched by the vehicle AUVs in a future pass. In total, there is one carrier AUV with four vehicle AUVs.

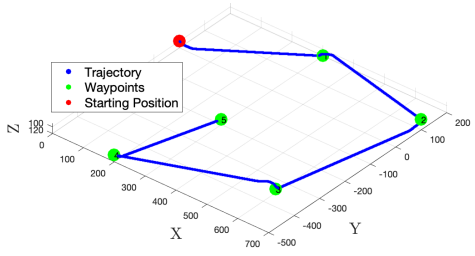

Fig. 16. Path of the large carrier AUV

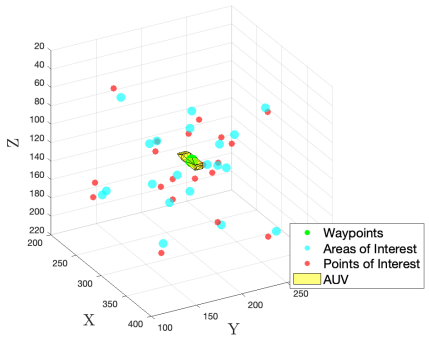

Fig. 17. Areas/Points of Interest Relative to the Carrier AUV Vehicle Deployment Point (Green Waypoint)

D. Results

Here we will go over how well our multi-AUV planning works in simulation. Figure 18 shows the vehicle trajectories for traversing the points of interest. We can see that the trajectories are well spaced out and they result in all the vehicle AUVs returning back to the carrier AUV within a similar timeframe.

Note: For better simulation results, refer to the demo videos found in the Video Demos section.

X. Path Planning

We also explored the application of exploring sunken underwater ships. With this, we determined that implementing an object avoidance pathing algorithm was essential to the success of our AUV to move throughout the sunken ship.

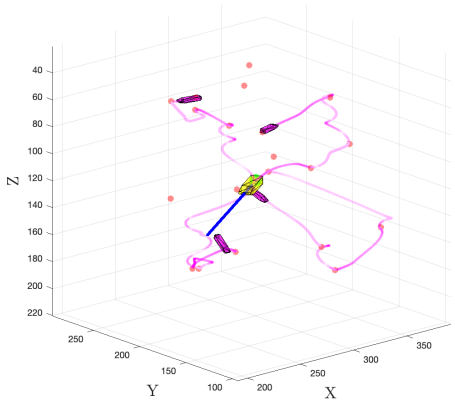

Fig. 18. Temp

A. Assumptions

For this section, we assumed that we had perfect knowledge of the map our AUV was traversing through. This keeps our problem focused solely working on the path-planning aspect of moving through a sunken shipwreck versus a vision challenge of mapping our surroundings. This is something we can explore in the future for our research.

B. Occupancy Map

To create a map for our AUV to traverse through, we use a 3D occupancy map to model the area in which our model will be moving through. An occupancy map handles all the (X, Y, Z) coordinates and their occupancy status. For an occupancy map, a cell can be either unoccupied, unknown, or a free space. For our model, we are only holding onto the values of whether the space is unoccupied or free since we are not mapping our space but rather using a predetermined map.

For our map, we used the Matlab function *occupancyMap3D()* to create our occupancy map. We then set a specific resolution for our map to change how much size each cell would occupy. We determined a cell size of 3.33 meters per cell was the most efficient sizing of cells for our use case giving us a resolution of 0.3. We then initialize a map size of $150\ meters * resolution = 45$ cells to leave us with a 90x90x90 grid of free space with each cell being 3.33 meters. We can then add obstacles by setting a group of cells to be occupied in the occupancy map.

After having the occupancy grid finished with all the obstacles inserted, we can then inflate each obstacle to the radius of our AUV. This is done to make sure that our AUV does not path too close to the obstacles included in the map. We also give our map a safety radius of 3 meters to give some leeway for our AUV to path around obstacles.

Using this occupancy map, we can pass that to our A* path planning algorithm along with a list of desired waypoints to reach. When passing the waypoints into our A* algorithm, we only pass two waypoints at a time since A* only considers a start node and a goal pose. We use a loop to pass each

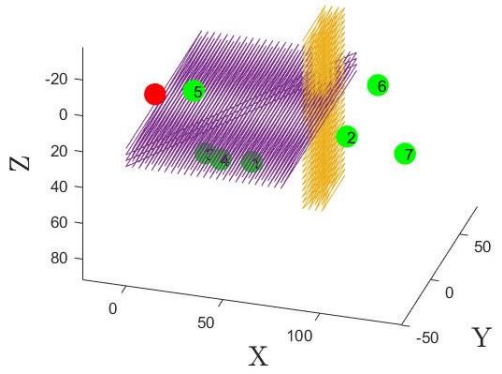

Fig. 19. Example Occupancy Map and Desired Waypoints

waypoint in and just combine all the paths found by the A* algorithm to get one full path that reaches each waypoint.

C. A* Path Planning

The A* pathing algorithm handles the majority of the work when pathing to each waypoint. It takes in a start node and a goal node and uses two heuristics to calculate the "costs" of each cell as it searches for a valid path towards the goal. We focus on three costs, the g cost, h cost, and f cost. The g cost is the distance traveled from the start node to the desired node. The h cost is the distance from the desired node to the goal node. The f cost is the sum of these two costs.

The A* algorithm uses an open list and a closed list which are dictionaries to hold a list of cells. The open list contains cells that have been found, but not searched yet, and the closed list contains a list of cells that have been investigated. When starting the algorithm, we start with the closed list empty and the open list with the start node. To find each list, we iterate through the open list and pop the node in the list with the lowest f cost. With this node we check if it has the same coordinates as the goal node, and will end the search if it is. If not, The algorithm then expands from that current node to the adjacent surrounding nodes if they occupy a free space in the occupancy map. As we look at each other expanded node, we record their coordinates, parent node, g cost, h cost, and f cost. If a child node is in either the open list we update the values in the lists. This signifies we have found a better path utilizing the node than previously. If the node is not in either list, we then add it to the open list.

By going through this loop, our A* algorithm expands from the start node until it either reaches the goal node or checks every node until the open list is empty. If the open list becomes empty, that means no path exists between the start node and the goal node. If a goal node is found, we exit the loop and extract a path using the parents of each node. We start at the goal node and keep track of each coordinate reached until we arrive at the start node. This gives us a list of coordinates the A* algorithm took to find the goal node from the start node. After this, we prune the path by comparing the direction each step of the path takes. If they are the same, we remove in-between nodes that are included in our path array. This gives us a path that only has the key coordinates where we are changing our heading. We then return this whole path to our AUV controller to follow each point in the entire path.

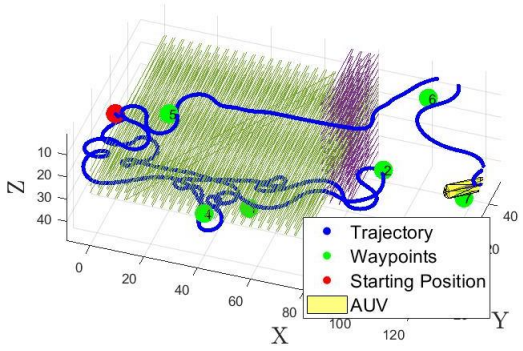

Fig. 20. Example Path Planning with Obstacle Avoidance

D. Findings

From our A* path planning, we noticed that it would cause our AUV model to move slowly in some areas where it had to turn a lot. This may be due to the fact that if there are small changes in the points in the path the AUV PID controller struggles to move efficiently through these small changes. This may be fixed by tuning the integral term for the velocity controller or changing the threshold of how close the AUV has to be to a point to considered it reached.

XI. Video Demos

For a real time simulation of our simple waypoint navigation see the video here. A demo of our multi-AUV planning can be found here. A demo of our A* path planning can be found here.

XII. Conclusion

We linearized the equations for our control and implemented PID controllers to control our AUV. We demonstrated that we can use our controllers to navigate basic waypoints with and without constant current. We also explored more complicated planning including marsupial planning and obstacle avoidance and showed how our AUV performs. There are many adaptations of the current model which may be explored in future work. As stated previously, the roll angle of the vehicle was neglected to mimic the conditions of the model developed in [2]. It would be desirable to explore the effects of roll on the response of the PID controller if it were included in the vehicle response. In addition, the AUV model could be given more autonomous capabilities to enable a firmer decision making process beyond the naviagtion of waypoints in a predetermined order.

REFERENCES

[1] Warner, David C. "Design, Simulation and Experimental Verification of a Computer Model and Enhanced Position Estimator for the NPS AUV II," in Naval Postgraduate School, 1991.

[2] A. J. Healey and D. Lienard, "Multivariable Sliding-Mode Control for Autonomous Driving and Steering of Unmanned Underwater Vehicles," IEEE J. Oceanic Eng., vol. 18, NO. 3, 1993.

[3] D. Lienard, "Autopilot Design for Autonomous Underwater Vehicles Based on Sliding Mode Control," in Naval Postgraduate School, 1990.

[4] Van den Bremer, T. S., and Breivik, "Stokes Drift," in Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol 376, issue 2111, 2018.

[5] R. Tian, H. Chen, G. Frey, B. Zu, A. Girard and I. Kolmanovsky, "Path planning for information collection in contested environments using marsupial systems," 2017 International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 2017, pp. 706-715, doi: 10.1109/ICUAS.2017.7991386.

[6] Garone, Emanuele & Naldi, Roberto & Casavola, Alessandro. (2011). A Traveling Salesman Problem for a Class of Carrier-Vehicle Systems. Journal of Guidance, Control, and Dynamics.

[7] András Király, János Abonyi, Redesign of the supply of mobile mechanics based on a novel genetic optimization algorithm using Google Maps API, Engineering Applications of Artificial Intelligence, Volume 38, 2015, Pages 122-130. Matlab Code: here.

APPENDIX

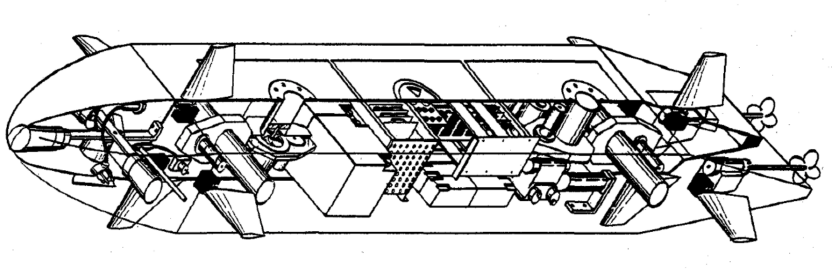

Fig. 21. Sketch of the NPS II Vehicle

Variable	Description
x,y,z	Distance along the principle axes
u,v,w	Velocity Components of body axis system relative to fluid along body axes
p,q,r	Angular velocity components of body relative to inertial reference system along body axes
X, Y, Z	Hydrodynamic force components along body axes
K,M,N	Hydrodynamic moment components along body axes
Ψ, θ, ϕ	Yaw, pitch, and roll angles (Euler angles)
m	Mass of AUV
W	Weight of AUV
v	Displacement volume of the AUV
B	Buoyancy force acting on the AUV
x_G, y_G, z_G	Coordinates of the Center of Gravity in the body axis system. These depend on the mass distribution of the vehicle
x_B, y_B, z_B	Coordinates of the Center of Gravity in the body axis system. These depend on the mass distribution of the vehicle
I_x, I_y, I_z	Moments of inertia about the body system axes
I_{xy}, I_{xz}, I_{yz}	Products of inertia about the body system axes
ρ	Mass density water
l	Reference length used to nondimensionalize the hydrodynamic coefficients
$b(x), h(x)$	Width and height of the AUV in the xy and xz planes, respectively, measured in the body axis system.
x_{nose}, x_{tail}	Coordinates of the vehicle nose and tail as measured in body axis system
$U_{cf}(x)$	Total crossflow velocity on AUV at position x
δ_{rb}, δ_{rs}	Bow and Stern rudder deflection angles in radians
C_{Dy}, C_{Dz}	Drag coefficients along the y and z axes of the body system axes