Full Autonomy of Hovering AUV

Easton Potokar

April 2022

1 Introduction

For my project, I implemented full autonomous navigation on a hovering autonomous underwater vehicle (HAUV) including state estimation, low-level control, trajectory following, and path-planning. In the following sections, I cover how each of these were implemented including specifics on the invariant extended Kalman filter (InEKF), dynamics for use in a linear quadratic regulator (LQR), and implementation of the rapidly-exploring random trees (RRT) path planner. These steps are visualized in Fig. 1.

This is all done on simulation in HoloOcean [2], which features all necessary common underwater sensors, an easy-to-use python interface, and accurate dynamics. Code for this entire project can be found at `https://github.com/contagon/AUVControl`.

2 State Estimation

For state estimation, I used the InEKF to track the position, velocity, and orientation in the global frame along with both the angular velocity and acceleration biases. These can be represented respectively by $R_b^g, v_{gb}^g, p_{gb}^g$, but for conciseness I abbreviate these as R, v, p. I do this using an IMU for the prediction step, and a doppler velocity log (DVL), pressure sensor, magnetometer, and GPS for the update step.

Previous literature has been done deriving these models [3] and is not the main purpose of this project, so I will just briefly cover it here. I model our state X as an element of $SE_2(3)$, a 5×5 matrix, as follows

$$X \triangleq \begin{bmatrix} R & v & p \\ 0_{1\times 3} & 1 & 0 \\ 0_{1\times 3} & 0 & 1 \end{bmatrix}. \tag{1}$$

Our process models involves integrating the IMU measurements as follows,

$$\hat{R} = \hat{R}\exp((\tilde{\omega} - \hat{b}^{\omega})\Delta t) \tag{2}$$

$$\hat{v} = \hat{v} + \hat{R}(\tilde{a} - \hat{b}^{a})\Delta t + g\Delta t \tag{3}$$

$$\hat{p} = \hat{p} + \hat{v}\Delta t + \frac{1}{2}\hat{R}(\tilde{a} - \hat{b}^{a})\Delta t^2 + \frac{1}{2}g\Delta t^2 \tag{4}$$

$$\hat{b}^{\omega} = \hat{b}^{\omega}, \quad \hat{b}^{a} = \hat{b}^{a} \tag{5}$$

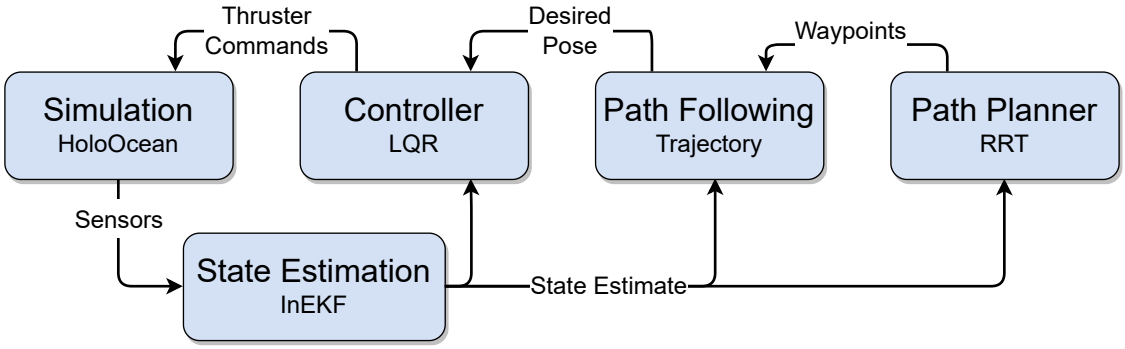


Figure 1: Flow chart of full stack implementation. HoloOcean outputs IMU, DVL, GPS, Magnetometer and pressure sensor data. This is then used by the invariant extended Kalman Filter to estimate our state. A path through the environment is planned using RRT, which is then fed to a simple time-based trajectory generator, and desired poses are then fed into an LQR controller.

where $\hat{b}^{\omega}$ and $\hat{b}^{a}$ are the angular velocity and acceleration biases, respectively.

Our measurement models for the DVL, pressure sensor, GPS, and magnetometer, are respectively

$$\tilde{z}^{DVL} = R^T v + w^v, \qquad \mathrm{w}^v \sim \mathcal{N}(0_3, M^v) \tag{6}$$

$$\tilde{z}^{PRES} = p^z + \mathrm{w}^z, \qquad \mathrm{w}^z \sim \mathcal{N}(0_1, \sigma_z) \tag{7}$$

$$\tilde{z}^{GPS} = p + \mathrm{w}^p, \qquad \mathrm{w}^p \sim \mathcal{N}(0_3, M^p) \tag{8}$$

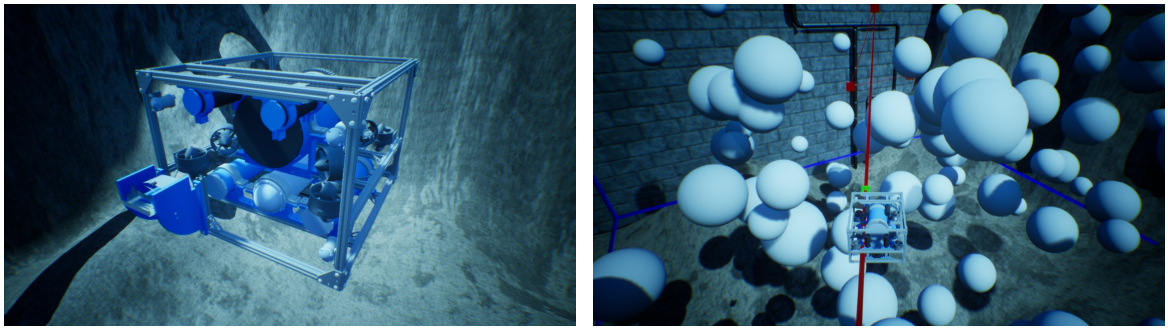
$$\tilde{z}^{MAG} = R^T m + \mathrm{w}^c, \qquad \mathrm{w}^c \sim \mathcal{N}(0, M^c) \tag{9}$$

These all fit well into the InEKF model, as does the IMU process model, and have a fairly straightforward implementation using the InEKF library [1]. I have the IMU sampled at 100Hz, DVL at 5Hz, pressure sensor at 50Hz, magnetometer at 50Hz, and GPS at 2Hz if within 3 meters of the surface.

3 Control

To accurately control the AUV, I first derive its dynamics. This took me a good portion of time, but, fortunately, is very similar to the dynamics of a quadrotor with some adjustments. The HAUV has 8 thrusters, 4 vertical ones and 4 horizontal ones at 45° angles for complete motion control. Six of these can be seen in Fig. 2(a). First, I convert our 8 thruster commands into x, y, z forces and torques, as follows,

$$W^b = \begin{bmatrix} F^b \\ \tau^b \end{bmatrix} = Mf = \begin{bmatrix} d_1 & \cdots & d_8 \\ p_1 \times d_1 & \cdots & p_8 \times d_8 \end{bmatrix} f \tag{10}$$



(a) (b)

Figure 2: (a) HAUV, with 6 of 8 thrusters visible. (b) The agent moving following a RRT trajectory through the environment with sphere objects. The RRT waypoints are shown as red dots, the trajectory as a red line, and the current commanded pose as the green dot.

where each d_i is the direction of the thruster, and p_i is the position vector from the COM to the thruster in the body frame. The d_i are given by

$$d_1 = d_2 = d_3 = d_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad d_5 = d_7 = \begin{bmatrix} \sqrt{2} \\ \sqrt{2} \\ 0 \end{bmatrix}, \quad d_6 = d_8 = \begin{bmatrix} \sqrt{2} \\ -\sqrt{2} \\ 0 \end{bmatrix}, \tag{11}$$

and the p_i are vehicle specific and not included here for conciseness. This matrix is 6×8 so I invert it using the right Moore-Penrose pseudo inverse,

$$f = M^{\dagger} W^b = M^T (M M^T)^{-1} W^b. \tag{12}$$

Using this F^b, τ^b, our dynamics are as follows with position and velocity in the global frame, and ω in the body frame,

$$\dot{p} = v \tag{13}$$

$$\dot{v} = \frac{1}{m} R_b^g F^b + g e_3 - \frac{\rho g}{m} V e_3 - D v \tag{14}$$

$$\dot{\Theta} = S(\Theta)\omega \tag{15}$$

$$\dot{\omega} = J^{-1}\Big(-\omega \times (J\omega) + \tau^b - \tau_{BY}^b \Big) - E\omega \tag{16}$$

where g is gravity, $e_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, ρ the density of water, V the volume of the vehicle, D, E damping matrices, J the inertia matrix, Θ the standard Euler angles, and τ_{BY}^b is the buoyant torque in the body frame given by,

$$V \rho g (R_g^b e_3 \times p_{COB}^b) \tag{17}$$

with p_{COB}^b being the position vector of the center of buoyancy from the center of mass in the body frame. I make a few assumptions to simplify these models into a linear form.

First, I assume that $m/\rho = V$, implying that the vehicle is perfectly buoyant. This is often done using flotation devices and is often close to true, and results in the buoyancy force and gravity canceling out.

Next, since pitch and roll angles are generally close to 0, I assume $S(\theta) = I$. Since the HAUV is close to square, I assume $J = cI$ for some $c > 0$, implying $\omega \times J\omega = 0$.

Finally, I feedback linearize, letting $\tilde{F} = R_b^g F^b$ and $\tilde{\tau} = \tau^b - \tau_{BY}^b$. This results in the linear model

$$\begin{bmatrix} \dot{p} \\ v \\ \Theta \\ \omega \end{bmatrix} = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & -D & 0 & 0 \\ 0 & 0 & 0 & I \\ 0 & 0 & 0 & -E \end{bmatrix} \begin{bmatrix} p \\ v \\ \Theta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{1}{m} I & 0 \\ 0 & 0 \\ 0 & J^{-1} \end{bmatrix} \begin{bmatrix} \tilde{F} \\ \tilde{\tau} \end{bmatrix} \triangleq Ax + Bu. \tag{18}$$

This system is fully controllable and is controlled using LQR. I tuned the gains accordingly to get good performance, but I'm sure there's plenty of room for improvement still.

4 Trajectory

Since our state is fully controllable - including both linear and angular velocity - I use a simple trajectory following scheme, where the agent is sent to a desired state given by the time of simulation. Desired position and orientation (via Euler angles) are determined this way, and both velocities are determined through a simple numerical derivative.

5 Path Planning

In our environment I randomly initialized 100 spheres with sizes from 2 to 5 meters and make a path across the environment. I used RRT with a step size of 5 meters to generate potential paths in 3D, then chose the path with the shortest length to execute. After choosing a path, the waypoints are iteratively stepped through to smooth the path by removing unnecessary waypoints. An example of a resulting trajectory can be seen in Fig. 2(b).

6 Conclusion

The full stack appears to work well, even when running on the estimated values from the InEKF. The odometry does drift and could be improved with some sort of SLAM implementation or from surfacing more often to gather GPS data.

Room for improvement includes an integrator in the LQR controller, use of the invariant error in the LQR controller, and implementation of an RRT* (or other) closer to optimal path planners.

References

[1] Easton Potokar. InEKF. `https://bitbucket.org/frostlab/inekf`, 2020.

[2] Easton Potokar, Spencer Ashford, and Joshua Mangelson. HoloOcean: An underwater robotics simulator. In *Proc. IEEE Int. Conf. Robot. and Automation*, 2022. `https://bitbucket.org/frostlab/holoocean`.

[3] Easton R Potokar, Kalin Norman, and Joshua G Mangelson. Invariant extended kalman filtering for underwater navigation. *IEEE Robotics and Automation Letters*, 6(3):5792–5799, 2021.