Submarine Mission Report

Claudio Vestini

Motivation This brief report will concern the B1 submarine coding practical, where we were tasked with designing the controller to guide a Submarine through its cave mission by tracking a given reference to avoid collisions with the cave boundaries. Preliminary steps taken before starting development:

1. Create and activate a virtual environment with the given requirements (numpy, matplotlib and pandas packages).
2. Fork project repository to my GitHub account.
3. Set up a .env file to add local packages onto Python PATH.
4. Make sure running files do not give any errors before branching off 'main'.

Mission Data The first task was to obtain mission data from the given .csv file. I achieved this through the following steps:

5. Create a new branch to modify the Mission class within.
6. Implement a new classmethod to extract each column of the mission.csv file into a separate variable, then return the data as an instance of the Mission class.
7. Test the new functionality by using the Trajectory class's plotting methods; then merge the branch back into 'main', and delete the unused branch.

Controller The design of the controller necessitated a thorough understanding of the Submarine and ClosedLoop classes, both of which were to be modified. For full analysis, check Apppendix A. I completed my implementation as follows:

8. Create a new branch to avoid pushing error prone code to 'main'.
9. Add a new method to the Submarine class to obtain the state space dynamics via matrices A, B, C and D.
10. Inside of a new module named control.py, create a new Controller class, initialised with the four matrices above. Create a subclass PDController, which inherits the dynamics, and possesses additional class variables K_P and K_D. For this subclass, develop a class method to compute the next control action $u[t]$ given observation $y[t]$ and reference $r[t]$.
11. Modify the ClosedLoop class to correctly call the controller within the simulate method.
12. Test the controller by using the given demo.ipynb file. Merge and delete the branch.

My decision to use a hierarchical class system proved effective as it kept the codebase modular. It also allows for future development of any other type of controller: I subsequently developed another subclass MPCController, which was better at tracking the reference but required more computational effort.

A System Equations

To implement the controller, I first analysed the given code to infer the system's dynamics. The submarine progresses at constant speed in the x direction, so needs only be controlled in the y direction (another hint to this is that we only have one set of reference values).

By inspection of the transition method, given drag D, velocity $\frac{dy}{dt}$, actuator gain K, input $u(t)$ and disturbance $d(t)$, the force in the vertical direction is given by:

$$F_y = -D \cdot \frac{dy}{dt} + K \cdot [u(t) + d(t)] \tag{1}$$

Combining (1) with Newton's second law we obtain acceleration:

$$\frac{d^2y}{dt^2} = -\frac{D}{m} \cdot \frac{dy}{dt} + \frac{K}{m} \cdot [u(t) + d(t)] \tag{2}$$

It is then very simple to obtain all matrices for the (continuous) state space dynamics of the Plant in canonic form:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + B[u(t) + d(t)] \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{3}$$

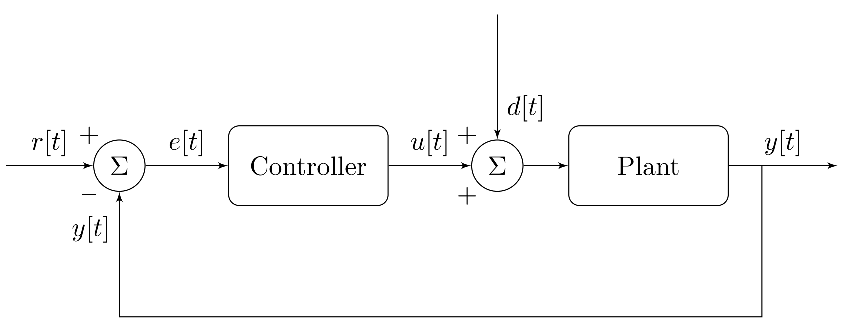
as:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{D}{m} \end{bmatrix}; \quad B = \begin{bmatrix} 0 \\ \frac{K}{m} \end{bmatrix}; \quad C = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad D = \begin{bmatrix} 0 \end{bmatrix} \tag{4}$$

Our system operates in discrete time, hence the PD Controller takes the standard form:

$$u[t] = K_P \cdot e[t] + K_D \cdot (e[t] - e[t-1]) \tag{5}$$

The system diagram in discrete time is then:



where the Plant is specified in equations (3) and (4) and the Controller is given by equation (5), with gains $K_P = 0.15$ and $K_D = 0.6$ (as per requirements).