Implementation of closed-loop control for UUV

Jin Rhee

18 October 2024

Background This report describes work done as the coding practical of B1 Scientific Computing in Michaelmas Term 2024. The motivation is to develop a controller for closed-loop control of an uncrewed underwater vehicle (UUV) given mission data and submarine dynamics.

Problem definition Two tasks are defined: 1. to create a function to import mission data from an external `.csv` file, and 2. to implement and test a PD feedback controller.

Key changes Changes made to files within `/uuv_mission` include: 1. implement `Mission.from_csv()` to import `.csv` mission data, 2. implement a PD (proportional-derivative) controller as the `Controller` class, 3. complete `ClosedLoop.simulate()`, and 4. add `Trajectory.get_traj_error()` which returns trajectory error to use in control gain tuning.

Changes to other files include a new bash script that conveniently imports Python modules, and a test unit within `notebooks/demo.ipynb` which tests and determines optimal control gains with the lowest trajectory error.

Design choices 1. The controller is implemented as a Class rather than a function such that future iterations that use different control schemes can be implemented as a method within the Class. It has three internal variables, with `prev_error` initially set to 0 at time zero and the respective gains being set to the suggested values given in the handout. Functions `set_gains()` and `set_prev_error()` allow the user to 'reset' a class instance for reuse rather than constructing a new instance. 2. The function `get_traj_error()` returns the root of squared error sums for a trajectory as to 'normalize' possibly negative error values. This is sampled and averaged over a number of iterations for one control gain pair to compare with another. 3. The `./install_requirements` script was created for the user to easily install Python dependencies. While somewhat redundant in its current state, other processes (such as virtual env setup) could be automated by this script in future iterations for further convenience.

Difficulties and future work One inconvenience when implementing the trajectory error function involves the error state, which is internally computed within a `ClosedLoop` instance at each timestep, which is inaccessible. Thus the error (reference - depth) value must be recomputed within `Trajectory.get_traj_error()` at each timestep, representing an inefficiency. Future work could solve this by restructuring the code. Other work could involve improving the setup script, implementing other modes of control (PID, MPC, etc.) within the `Controller` class, implementing other heuristic tuning methods (i.e. Ziegler-Nichols method), and implementing such tuning methods as a separate function or class.