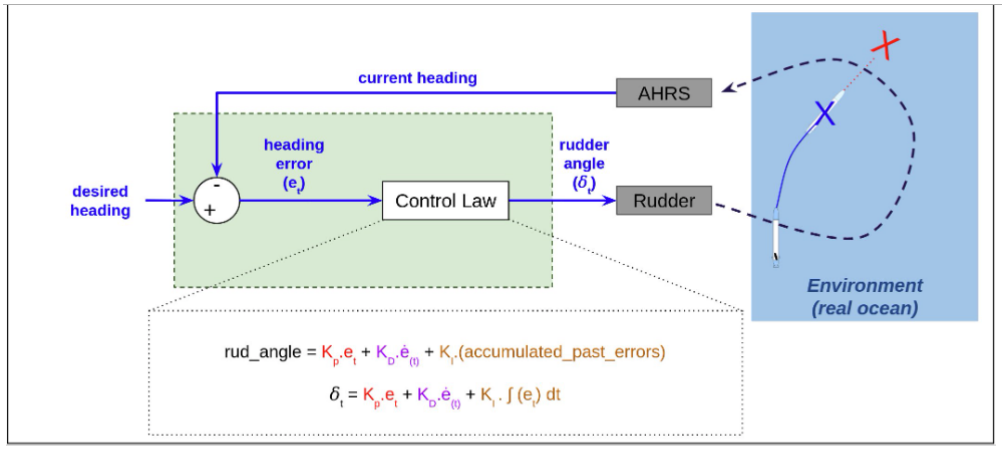
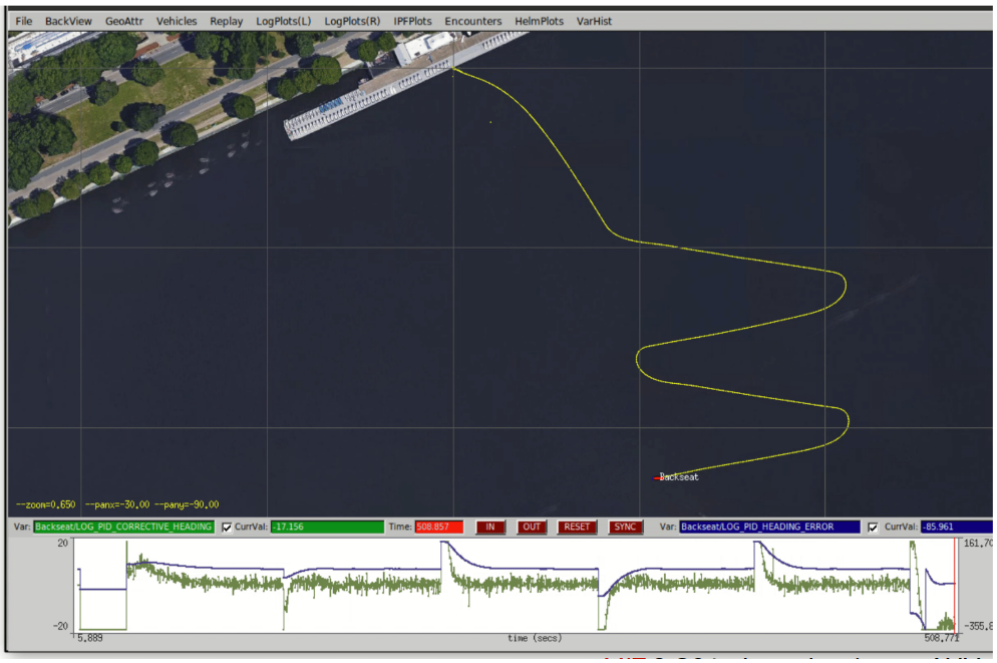
Lab 7b - Low-level Control Systems of AUVs

2.S01 Introduction to Autonomous Underwater Vehicles





Spring 2026

Supun Randeni, supun@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Objectives

- Learning how to create a new vehicle definition file for your own AUV.
- Learning how to push and pull code changes into and from github remote server.
- Understanding the effects of PID gains on the control performance of an AUV.
- Understanding how to tune PID gains of an AUV using trial-and-error method.
- Learning how to conduct post-mission analysis of vehicle data.

2 Updating your computers to the latest software

The instructors frequently update `StackUxV`, `StackUxV-drivers` and `VECTORS` software packages, as well as the `missions-StackUxV` repository. Therefore, you should always begin each lab by updating these software on the topside laptop, Raspberry Pi, and PocketBeagle computers.

2.1 Updating `StackUxV`, `StackUxV-drivers` and `VECTORS`

As we discussed in Lab 5, we can update `StackUxV`, `StackUxV-drivers`, and `VECTORS` by updating the `StackUxV-DEBIAN-PKG-ARCHIVE` repository and re-running the `install_packages.sh` script. Let's start with the topside computer; `cd` into the `StackUxV-DEBIAN-PKG-ARCHIVE` directory on the topside computer:

```
$ cd ~/StackUxV-DEBIAN-PKG-ARCHIVE/
```

First, update the `StackUxV-DEBIAN-PKG-ARCHIVE` Git repository to obtain the latest packages by using the `git pull origin main` command. This will pull the latest version of the repository from the Git server (that is, `origin`) on the `main` branch:

```
$ git pull origin main
From github.com:supun-randeni/StackUxV-DEBIAN-PKG-ARCHIVE
 * branch            main       -> FETCH_HEAD
Already up to date.
```

Now you can use the `install_packages.sh` script to re-install the latest versions of `StackUxV`, `StackUxV-drivers` and `VECTORS`.

Now you can repeat the same steps to update `StackUxV`, `StackUxV-drivers` and `VECTORS` on the Raspberry Pi and PocketBeagle in your training-kit.

2.2 Updating `missions-StackUxV`

As we discussed, `missions-StackUxV` repository contains only shell scripts and MOOS configuration files, so there is no need to build the code. You only need to update the repository by running the `git pull` command. Let's start with the topside computer:

```
$ cd ~/missions-StackUxV/
$ git pull origin 2026_2s01
```

Now let's update missions-StackUxV on the embedded computers. As we discussed in the last lab, we created the update_missions_stackuxv program to quickly update the current branch of missions-StackUxV on both the Raspberry Pi and the PocketBeagle at the same time. When you run this program on the Raspberry Pi, it first updates the Raspberry Pi and then remotely executes the update commands on the PocketBeagle by sending commands over SSH. Please make sure to run this program on the Raspberry Pi, not on the PocketBeagle.

```
$ update_missions_stackuxv
```

3 Configuring your own vehicle

During the first part of today's lab, you will create a vehicle configuration file for your own AUV in missions-StackUxV. To minimize human error, we highly recommend avoiding direct software edits on embedded computers. The preferred workflow is to make edits on your laptop (i.e., the topside computer), commit and push the changes to the remote Git server at https://github.com/supun-randeni/missions-StackUxV, and then pull those changes onto the embedded computers.

3.1 Creating a git branch for your vehicle

Once you have updated StackUxV, StackUxV-drivers, and missions-StackUxV on the topside computer, change the directory to missions-StackUxV. Before editing any files, we ask that you create a new Git branch named after your AUV (i.e. the name of your training-kit; mine is called *Bee*). This allows you to make modifications without interfering with others' workflows:

```
$ cd ~/missions-StackUxV/
$ git pull
$ git checkout -b bee 2026_2s01
Switched to a new branch 'bee'
```

The command git checkout -b bee 2026_2s01 creates a new Git branch called bee, based on the existing branch 2026_2s01. This means the software in the bee branch is currently identical to that in 2026_2s01, since you have not made any code changes yet. Right now, this new branch exists only on your topside laptop. Next, let's push (or upload) this new branch to the remote Git server (called origin) so that it becomes available to everyone to pull (or download), including your embedded computers:

```
$ git push -u origin bee
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'bee' on GitHub by visiting:
remote:      https://github.mit.edu/supun-randeni/missions-StackUxV/pull/new/bee
remote:
To github.mit.edu:supun-randeni/missions-StackUxV.git
 * [new branch]      bee -> bee
Branch 'bee' set up to track remote branch 'bee' from 'origin'.
```

We strongly recommend you to read on git version control on your own time (https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control).

3.2 Creating a vehicle definition file for your vehicle

Now that you have your own Git branch for missions-StackUxV, you can modify it as needed without interfering with the workflow of other missions-StackUxV users (*for example, the AUVs Ivy and Kay are currently being deployed 10-20 miles offshore of Hawaii as part of an experiment – it is important that we do not edit files that they rely on to avoid disrupting their operations*). Let's start by creating a definition file for your own AUV. Vehicle definition files are located in the missions-StackUxV/vehicle directory. Change your directory to missions-StackUxV/vehicle and explore the files in this directory:

```
$ cd ~/missions-StackUxV/vehicle/
$ ls -F
buoy_one/    buoy_two/  current_vehicle@    fay/         ham/  jan/  sb2_alpha/
buoy_three/  cap/       ematt_prototype_1/  glider_one/  ivy/  kay/  template_uuv/
```

There is a directory for each vehicle that is currently configured. At a minimum, you will see the directories cap and ivy. Change your directory to one of these (e.g., cap) and explore the files inside it:

```
$ cd cap
$ ls -F
sensor_config.txt  vehicle.def  vehicle_plugs/
```

As you can see, each AUV has two files and one directory: the vehicle.def file, sensor_config.txt file, and vehicle_plugs/ directory. Open these files and examine their contents. Try to recognize or guess what each of the parameters represents.

The contents of the vehicle_plugs/ directory are beyond the scope of 2.S01, but it is still helpful to understand its general purpose. This directory provides a "drag-and-drop" structure for defining any additional MOOS applications associated with a particular vehicle, organized by MOOS community. The vehicle_plugs/ directory contains several subdirectories, including backseat_plugs, frontseat_plugs, hydroman_plugs, topside_plugs, virtual_ocean_plugs, and cloak_plugs. Each of these subdirectories may contain configuration files for extra applications relevant to that community. If a configuration file is present in one of these subdirectories, the corresponding MOOS application will be included and run in that MOOS community.

To create a vehicle definition for a new AUV, you need to create a new directory named after that vehicle and populate it with a vehicle.def file, a sensor_config.txt file, and the vehicle_plugs/ directory structure. In practice, the easiest way to do this is to duplicate the existing cap directory, rename it with your vehicle's name, and then edit the relevant parameters in vehicle.def and sensor_config.txt.

```
$ cd ~/missions-StackUxV/vehicle/
$ ls -F
buoy_one/    buoy_two/  current_vehicle@    fay/         ham/  jan/  sb2_alpha/
buoy_three/  cap/       ematt_prototype_1/  glider_one/  ivy/  kay/  template_uuv/
$ cp -r cap/ bee
$ ls -F
bee/         buoy_two/         ematt_prototype_1/  ham/  kay/
buoy_one/    cap/              fay/                ivy/  sb2_alpha/
buoy_three/  current_vehicle@  glider_one/         jan/  template_uuv/
```

Now that you have created a new directory with your vehicle's name (e.g., `bee` in my case), you will need to update the `vehicle.def` and `sensor_config.txt` files inside it. Start by opening the `bee/vehicle.def` file in your preferred text editor, and update the following values as needed:

3.2.1 Changing the vehicle name

Change the vehicle name: `#define VEHICLE CAP` to `#define VEHICLE BEE`

3.2.2 Changing the vehicle ID

Your vehicle ID is the last two digits of your IP addrress; for example,if you IP address is `192.168.0.51`, your vehicle ID is `51`.

Change the vehicle ID: `#define VEHICLE_ID 2` to `#define VEHICLE_ID 51`

3.2.3 Changing the network port table

Following network ports should change for your vehicle, corresponding to your vehicle ID:

- `MOOS_PORT_COMBINED` = 91`ID`
- `MOOS_PORT_BACKSEAT` = 92`ID`
- `MOOS_PORT_FRONTSEAT` = 93`ID`
- `MOOS_PORT_HYDROMAN` = 94`ID`
- `INTERFACE_PORT_HYDROMAN` = 96`ID`

For example, if the last two digits of the vehicle IP address is 51, the ID should be replaced with 51. I.e., change: `#define MOOS_PORT_COMBINED 9152` to `#define MOOS_PORT_COMBINED 9151`.

3.3 Running a software-in-the-loop simulation to test the newly configured vehicle definition

Once you created the definition files for your vehicle, let's test it by running a software-in-the-loop simulation on the topside:

1. Configure the architecture to `seabeaver_iii`.

2. Configure the vehicle to your AUV (i.e. the definition that you just created; e.g. bee in my case).

3. Configure the cruise to `shipwreck_search`.

4. Launch the mission in simulation.

If you have created the vehicle definition without errors, you should see the same `shipwreck_search` mission on the `pMarineViewer` window that you saw during the last lab.

3.4 Pushing your software to the git remote server

Once you have successfully created the vehicle definition on your topside laptop, push (or upload) the new code to the Git remote server so that other collaborators (e.g., your lab partner, instructors) can also access it. Pushing the code to the remote server also allows you to pull the changes onto the embedded computers, eliminating the need to make the same modifications on the Raspberry Pi and PocketBeagle. We strongly recommend you take the time to read up on Git version control on your own (https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control).

If you use the command `git status` while in the `missions-StackUxV` directory or any of its subdirectories, you will see the new directory:

```
$ git status
On branch bee
Your branch is up to date with 'origin/bee'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        vehicle/bee/

nothing added to commit but untracked files present (use "git add" to track)
```

To commit a change, you first need to stage the file(s) you want to include in the commit. This is done by adding the modified file(s):

```
$ git add vehicle/bee
```

If you run `git status` now, you will see the files that have been staged for commit:

```
$ git status
On branch bee
Your branch is up to date with 'origin/bee'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   vehicle/bee/sensor_config.txt
        new file:   vehicle/bee/vehicle.def
        new file:   vehicle/bee/vehicle_plugs/backseat_plugs/suspended_app_list.plug
        new file:   vehicle/bee/vehicle_plugs/frontseat_plugs/suspended_app_list.plug
        new file:   vehicle/bee/vehicle_plugs/hydroman_plugs/suspended_app_list.plug
        new file:   vehicle/bee/vehicle_plugs/topside_plugs/suspended_app_list.plug
        new file:   vehicle/bee/vehicle_plugs/virtual_ocean_plugs/suspended_app_list.plug
```

Now that the changes are staged, you can commit them using the command `git commit -m "Adding the vehicle definition for the AUV Bee"`. The `-m` flag lets you include a short message with the commit, helping you and your collaborators understand the purpose of the change.

```
$ git commit -m "Adding the vehicle definition for the AUV Bee"
[bee 1525c35] Adding the vehicle definition for the AUV Bee
 2 files changed, 144 insertions(+)
 create mode 100644 vehicle/bee/sensor_config.txt
 create mode 100644 vehicle/bee/vehicle.def
```

You maybe asked to verify who you are, if you are pushing into this Git organization for the first time. Follow the instructions to provide your name and email address so that the Git organization knows who is committing these changes.

Once comitted, you can use `$git status` command again to double check if everything is as expected:

```
$ git status
On branch bee
Your branch is ahead of 'origin/bee' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Finally, you can push the commit(s) to the git remote server:

```
$ git push origin bee
Are you sure you want to continue connecting (yes/no)? yes
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.67 KiB | 1.67 MiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote: This repository moved. Please use the new location:
remote:   git@github.com:supun-randeni/missions-StackUxV.git
To github.com:supun-randeni/missions-seabeaver.git
   b8c66ed..1525c35  bee -> bee
```

3.5 Self checkoff assessment

1. Are you able to run the software-in-the-loop simulation using your own AUV's configuration?
2. When you run the software-in-the-loop simulation, does your AUV's name appear next to the vehicle icon instead of `CAP`?
3. Were you able to push your vehicle configuration to the github server?

4 Tuning the PID gains in software-in-the-loop simulations

During the lecture *"Low-level Control Systems of AUVs"*, we discussed the proportional-integral-derivative (PID) control algorithm and emphasized that tuning the P, I, and D gains is essential for achieving good control performance. In this part of the lab, you will tune the control gains for your vehicle by conducting software-in-the-loop simulations.

The PID tuning approach we'll use in this class is called the trial-and-error method. In this approach, we adjust the `Kp`, `Ki`, and `Kd` gains one at a time, while considering how each parameter influences the overall control performance. For example:

1. Conduct a PID tuning mission.
2. If you are running the experiment as a hardware-in-the-loop simulation or during an actual in-water deployment, download the log files to the topside computer.
3. Visualize the data using the `alogview` tool, and examine the vehicle's motion response.
4. Based on the observed behavior, determine which gains (`Kp`, `Ki`, or `Kd`) need adjustment, and update them in the vehicle definition file.
5. Run the PID tuning mission again using the updated gains.
6. Repeat the process until satisfactory control performance is achieved.

Later in this class, we will follow the same steps to tune the PID gains of your actual vehicle through in-water experiments. Therefore, it is useful to get familiar with the PID tuning process in software-in-the-loop simulations, where you can make mistakes and learn without damaging the actual vehicle.

4.1 Cleaning up past log files

Every time you launch a MOOS community, a MOOS log file is automatically generated on all computers, including the topside laptop, Raspberry Pi, and PocketBeagle. These log files are stored in the `missions-StackUxV/logs` directory on each computer. If a particular log file is important to you (e.g., log files from in-water experiments), you should copy it to a different location. Once you have saved the logs that are important, you can clear the `missions-StackUxV/logs` directory by running the `clean_logs.sh` script:

```
$ cd ~/missions-StackUxV/
$ ./clean_logs.sh
```

Do not forget that this directory exists on all three computers, and log files will be generated on each one (i.e., topside logs on the topside laptop, backseat logs on the Raspberry Pi, and frontseat logs on the PocketBeagle). Therefore, it is important to clean the logs on all three computers.

4.2 Configuring PID gains for your AUV

In the Section 3.2, you created a definition file for your own vehicle in `missions-StackUxV/vehicle` directory. For instance, for the AUV named `bee`, these definition files are in `bee/vehicle.def`. Inspect the `vehicle.def` file for your vehicle (e.g., `bee/vehicle.def`) using your preferred text editor. Around line 86, under the section *PID control related*, you will find the PID gain settings for your vehicle:

```
// ********************************************************************************************************
// PID control related
// ********************************************************************************************************
#define ENABLE_TRI_FIN_HEADING_CONTROL    true
#define ELEVATOR_HEADING_CONTROL_PERCENT  50

#define PROP_MODE_VEHICLE_PITCH_LIMIT     20

#define PROP_MODE_ROLL_KP                 0.1
#define PROP_MODE_ROLL_KI                 0
#define PROP_MODE_ROLL_KD                 0
#define PROP_MODE_ROLL_MAX_INTEGRAL       10

#define PROP_MODE_PITCH_KP                1.2
#define PROP_MODE_PITCH_KI                0
#define PROP_MODE_PITCH_KD                0
#define PROP_MODE_PITCH_MAX_INTEGRAL      5

#define PROP_MODE_DEPTH_KP                20
#define PROP_MODE_DEPTH_KI                1
#define PROP_MODE_DEPTH_KD                0.8
#define PROP_MODE_DEPTH_MAX_INTEGRAL      6

#define PROP_MODE_HEADING_KP              0.2
#define PROP_MODE_HEADING_KI              0.05
#define PROP_MODE_HEADING_KD              0.8
#define PROP_MODE_HEADING_MAX_INTEGRAL    10

#define PROP_MODE_SPEED_KP                20
#define PROP_MODE_SPEED_KI                0.5
#define PROP_MODE_SPEED_KD                0
#define PROP_MODE_SPEED_MAX_INTEGRAL      50

#define PROP_MODE_SPEED_CURVE             0.9:30 | 0.7:25 | 0.3:10 | 1.3:40
```

Let's go over what each of these parameters means and how they affect the control performance of the vehicle.

4.3 Tuning the heading control sub-system

The PID gains for the heading sub-system are defined under following section of the `vehicle.def` file:

```
#define PROP_MODE_HEADING_KP              0.2
#define PROP_MODE_HEADING_KI              0.05
#define PROP_MODE_HEADING_KD              0.8
#define PROP_MODE_HEADING_MAX_INTEGRAL    10
```

For example, the P gain for the heading sub-system (`PROP_MODE_HEADING_KP`) is currently set to 0.2. You can adjust this value by editing the `vehicle.def` file.

4.3.1 Observing the heading performance for the current PID gain setting

Before we begin tuning, let's first run a simulation and observe the control performance with the current PID gains. We have created a new cruise called `lab_7_control` specifically for PID tuning. This mission consists of five legs, each at a constant heading, performing a zig-zag pattern. Remember, you need to configure the architecture, vehicle, and cruise before running the mission (in my case, the AUV name is `bee`).

```
$ cd ~/missions-StackUxV/launch_scripts/
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh bee
$ ./configure_cruise.sh lab_7_control
```

Now you can launch the mission in the software-in-the-loop simulation (I am running the simulation at 5 times the real-time speed):

```
$ ./launch_simulation.sh 5
```

The `pMarineViewer` window will appear, and you will see the vehicles moving. Once the mission has ended, let's visualize the logs using `alogview`. Navigate to the directory where the logs are generated and list all the files:

```
$ cd ~/missions-StackUxV/logs/
$ ls
LOG_BEE_Backseat_22_4_2026_____14_00_08   LOG_BEE_VECTORS_22_4_2026_____14_00_08
LOG_BEE_Frontseat_22_4_2026_____14_00_07  LOG_VECTORS__hydro_summary_.txt
LOG_BEE_Topside_22_4_2026_____14_00_08
```

Since we run four MOOS communities in the topside computer during SITL simulations, you can see logs from all four (i.e. Backseat MOOS community, Frontseat MOOS community, Topside MOOS community and VECTORS MOOS community). For PID tuning, we are only interested in the Backseat MOOS community. In some cases, there might be logs from previous missions as well. Therefore we need to choose the correct log file. Usually it is the latest one (i.e. `LOG_BEE_Backseat_22_4_2026_____14_00_08`). Let's visualize it using `alogview`:

```
$ cd LOG_BEE_Backseat_22_4_2026_____14_00_08
$ alogview LOG_BEE_Backseat_22_4_2026_____14_00_08.alog
```

The `alogview` window shown in Figure 1 will pop up.

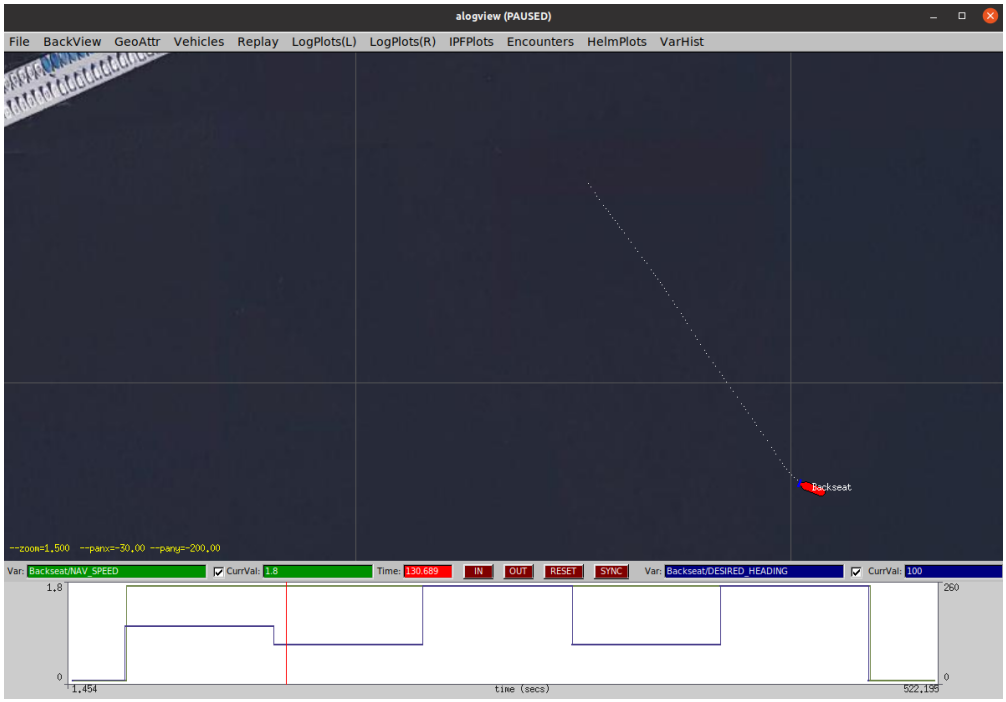


Figure 1: The alogview window

In alogview, you can plot time-series data (https://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=IvPTools.ALogView#sec_alogview). For example, you can plot the variation of DESIRED_HEADING and NAV_HEADING over time. To do this, load NAV_HEADING from the LogPlots(L) pull-down menu and DESIRED_HEADING from the same menu. You should see a response similar to Figure 2. Plot the following MOOS variables and analyze their responses:

- DESIRED_HEADING
- DESIRED_DEPTH
- DESIRED_SPEED
- LOG_PID_DESIRED_PITCH
- NAV_HEADING
- NAV_DEPTH
- NAV_SPEED
- NAV_PITCH
- LOG_PID_HEADING_ERROR
- LOG_PID_DEPTH_ERROR
- LOG_PID_PITCH_ERROR

- UPPER_RUDDER
- STBD_ELEVATOR
- PORT_ELEVATOR
- PROPELLER

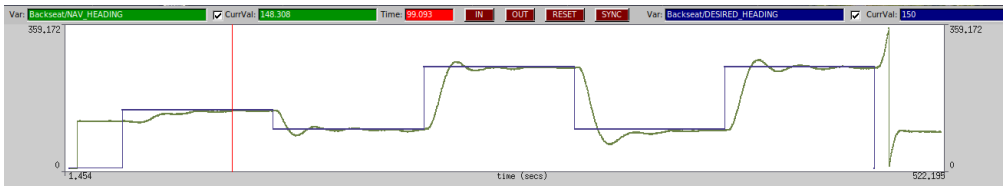


Figure 2: Time-series data of DESIRED_HEADING and NAV_HEADING

4.3.2 Changing heading PID gains

Now you can experiment with different Kp, Ki, and Kd gains for the heading sub-system and observe how the motion response varies. Try testing values such as 0.1, 0.2, 0.4, 0.8, and 1.6 for each gain. It is best the change one gain at a time so that you can systematically observe the variation in performance.

4.4 Tuning the depth and pitch control sub-systems

As we discussed during the *"Low-level Control Systems of AUVs"* lecture, depth control for under-actuated AUVs is achieved in two steps. As shown in Figure 3, (1) the depth control sub-system computes the desired pitch angle of the AUV to minimize the depth error, and (2) the pitch control sub-system computes the elevator angle to minimize the pitch error.

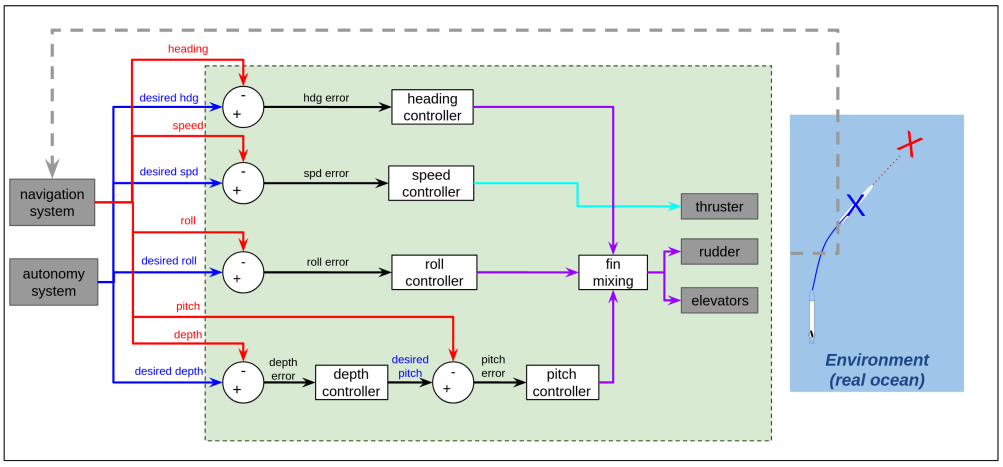


Figure 3: Low-level control sub-systems of SeaBeaver II AUVs

The PID gains for the depth and pitch sub-system are defined under following section of the `vehicle.def` file:

```
#define PROP_MODE_PITCH_KP                1.2
#define PROP_MODE_PITCH_KI                0
#define PROP_MODE_PITCH_KD                0
#define PROP_MODE_PITCH_MAX_INTEGRAL      5

#define PROP_MODE_DEPTH_KP                20
#define PROP_MODE_DEPTH_KI                1
#define PROP_MODE_DEPTH_KD                0.8
#define PROP_MODE_DEPTH_MAX_INTEGRAL      6
```

Following the same procedure described in Section 4.3, try adjusting the PID gains for both the depth and pitch sub-systems (one at a time), and observe the corresponding response.

5 Instructor Check-off Assessment

The lab assessments are structured to ensure that you stay on track to complete your AUV by the end of the semester for in-water trials. You are expected to demonstrate to one of the instructors that you can complete the following tasks. More importantly, please don't hesitate to communicate with the instructors if you encounter any issues.

5.1 Tuning the heading, depth and pitch PID control sub-systems

Tune the heading, depth, and pitch PID gains to achieve a stable control response for the vehicle. Ensure that the rudder and elevators are not oscillating at a very high frequency by plotting the time-series responses of the `UPPER_RUDDER`, `STBD_ELEVATOR`, and `PORT_ELEVATOR` MOOS variables. Once you've tuned the gains, post a screenshot of the `alogview` that illustrates the control response of your vehicle on Piazza.

5.2 Pushing your tuned PID gains to the github server

Once you are satisfied with the control performance, commit the code changes and push them to your branch on the GitHub remote server. If you do not remember how to do this, please follow the relevant steps in Section 3.4.

6 Extra Activities – Testing the tuned PID gains by conducting hardware in the loop simulations

Once you are satisfied with the control performance, conduct a hardware-in-the-loop simulation using these new PID gains. To transfer the updated gains to the Raspberry Pi and PocketBeagle, commit the changes and push them to your branch on the GitHub remote server (which you did in the last step).

Once you've pushed the updated gains to your branch on the GitHub remote server, you can pull them to the Raspberry Pi and PocketBeagle to test the new gains in hardware-in-the-loop

simulations.

6.1 Switching the missions-StackUxV Branch on the Raspberry Pi and PocketBeagle

The missions-StackUxV repositories on both the Raspberry Pi and PocketBeagle are still set to the 2026_2s01 branch. We need to switch each to your new branch and pull the updated code (i.e the newly tuned PID gains).

The missions-StackUxV repositories on the Raspberry Pi and PocketBeagle are not yet aware of your new branch. To make the new branch visible, first fetch all updates from the remote server using the command git fetch.

```
$ cd ~/missions-StackUxV
$ git fetch
$ git checkout bee
Branch 'bee' set up to track remote branch 'bee' from 'origin'.
Switched to a new branch 'bee'
$ git pull
```

Make sure to switch to your new branch on both the Raspberry Pi and PocketBeagle.

6.2 Quickly updating the software

Once you have switched to your new branch on both the Raspberry Pi and PocketBeagle, you can use the update_missions_stackuxv script to quickly update missions-StackUxV to the latest commit (i.e., equivalent to git pull) on both devices at once. Be sure to run this script on the Raspberry Pi. If run on the PocketBeagle, it will only update that device.

```
$ update_missions_stackuxv

Updating missions-StackUxV on SeaBeaver Raspberry Pi..

missions-StackUxV is available.
missions-StackUxV is in branch:  bee

Updating missions-StackUxV..
Already up to date.
Remotely triggering git update on the PocketBeagle..
Debian GNU/Linux 10

BeagleBoard.org Debian Buster IoT Image 2020-04-06

Support: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

Being launched remotely by the RasPi..

Updating missions-StackUxV on SeaBeaver PocketBeagle..

missions-StackUxV is available.
missions-StackUxV is in branch:  bee

Updating missions-StackUxV..
Already up to date.
Connection to 192.168.7.2 closed.
Remote git update on the PocketBeagle is complete.
```

6.3 Running a hardware-in-the-loop simulation

Once you have updated the embedded computers with your tuned PID gains, you can run a hardware-in-the-loop simulation to test the new gains. If you do not remember how to run hardware-in-the-loop simulations, please refer to the *Software and Hardware in the Loop Simulations* lab.

If you are running the hardware-in-the-loop simulations on the actual AUV (i.e. not the training-kit), please do not run the thruster for more than 30 seconds continuously. Prolonged operation in air can lead to overheating of the thruster.

6.4 Downloading logs and post-mission data visualization

Finally, you can download the log files (using `scp` command) from the Raspberry Pi to your topside laptop, and visualize them using `alogview` tool. Create a directory on your topside laptop where you want to store your logs. For instance, I have created a folder named `bee_logs`, and within it, a sub-folder with today's date:

```
$ cd ~
$ mkdir bee_logs
$ cd bee_logs
$ mkdir 20260422
$ cd 20260422
```

Now you can copy the logs to this folder using secure-copy:

```
$ scp -r seabeaver-raspi@10.42.0.5:/home/seabeaver-raspi/missions-StackUxV/logs/LOG* .
```

Now you can visualize these logs using alogview.