Lab 11 - Preliminary Sea Trials

2.S01 Introduction to Autonomous Underwater Vehicles

Spring 2026

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

1 Objectives **4**

2 Updating software **4**
2.1 Merging the branch 2026_2s01 into your own branch 4
2.2 Updating software on the vehicle 6

3 Calibrating the AHRS **6**
3.1 Calibration 6
3.2 Heading verification (instructor checkoff) 6

4 Understanding indicator light patterns **7**

5 Ballasting **8**
5.1 Vacuum check (self checkoff) 8
5.2 In-water ballasting (instructor checkoff) 8

6 Pre-launch system checks **8**
6.1 Verifying sensors and actuators . . . 8
6.2 Verify vacuum (self checkoff) . . . 9
6.3 Verify the vehicle axes system (instructor checkoff) . . . 9

7 First in-water deployment **12**
7.1 Simulating the mission (self checkoff) . . . 13
7.2 Runtime launch (instructor checkoff) . . . 14
7.3 Post-mission data analysis . . . 15

8 Second in-water deployment **15**

9 Tuning the low-level PID control system **16**

1 Objectives

- Learning how to calibrate a micro-electro mechanical systems (MEMS) inertial measurement unit (IMU).
- Learning how to ballast an AUV.
- Learning how to conduct pre-launch systems checks.
- Conducting in-water deployment.
- Learning how to tune the PID gains of an AUV.

2 Updating software

Before in-water experiments, it is essential to ensure that the vehicle is running the latest software.

2.1 Merging the branch 2026_2s01 into your own branch

At this point, you have created a git branch in missions-StackUxV for your own vehicle. However, class software updates are typically made to the 2026_2s01 branch. Therefore, you will need to merge the 2026_2s01 branch into your own branch using your topside computer. Then, push the updated branch to the GitHub remote server so that the latest version of your branch (e.g., the bee branch in Supun's case) can also be pulled onto your Raspberry Pi and PocketBeagle.

Before merging the 2026_2s01 branch into your own branch, it is good practice to check the status of your branch and ensure that there are no uncommitted local changes on your topside laptop:

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

nothing to commit, working tree clean
```

In my case, the bee branch does not have any modified or uncommitted files, nor any untracked files, so I am ready to merge. However, if you have untracked files or code modifications that are not committed, you may choose to either discard these changes using the $ git reset --hard command, or commit them (see the *Low-level Control Systems of AUVs* lab for more information).

Before we merge, let's make sure the updates to the 2026_2s01 branch have been downloaded to your topside computer. To do this, switch to that branch using the $ git checkout 2026_2s01 command:

```
$ git checkout 2026_2s01
Switched to branch '2026_2s01'
Your branch is up to date with 'origin/2026_2s01'.
```

Pull the latest version of 2026_2s01 branch from the git remote server on to your topside laptop using $ git pull origin 2026_2s01 command:

```
$ git pull origin 2026_2s01
Warning: the ECDSA host key for 'github.com' differs from the key for the IP address '140.82.112.4'
Offending key for IP in /home/sb-topside-15/.ssh/known_hosts:4
Matching host key in /home/sb-topside-15/.ssh/known_hosts:60
Are you sure you want to continue connecting (yes/no)? yes
From github.com:supun-randeni/missions-StackUxV
 * branch            2026_2s01      -> FETCH_HEAD
Already up to date.
```

Let's switch back to your own branch (in Supun's case, the branch named `bee`) using `$ git checkout <your-branch-name>` command.

```
$ git checkout <your-branch-name>
Switched to branch '<your-branch-name>'
Your branch is up to date with 'origin/<your-branch-name>'.
```

Finally we can merge the `2026_2s01` branch into your own branch using `$ git merge 2026_2s01` command:

```
$ git merge 2026_2s01
```

A terminal window may open with the nano text editor, prompting you to enter a comment about the merge. You can enter a comment and exit by pressing `Ctrl+X`. If both you and the instructors have modified the same file (which should not be the case here, but will likely occur in future merges), you will receive a warning about a merge conflict. In such cases, both parties need to discuss and resolve the conflict. If you encounter a merge conflict, please contact the instructors.

By merging the `2026_2s01` branch into your own branch, you have introduced changes to your branch on the topside computer. These are referred to as local changes. Now, you need to push them to the Git remote server so that the latest version of your branch (e.g., the `bee` branch in my case) can be downloaded onto your Raspberry Pi and PocketBeagle:

```
$ git push origin <your-branch-name>
Warning: the ECDSA host key for 'github.com' differs from the key for the IP address '140.82.113.4'
Offending key for IP in /home/sb-topside-15/.ssh/known_hosts:9
Matching host key in /home/sb-topside-15/.ssh/known_hosts:60
Are you sure you want to continue connecting (yes/no)? yes
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 282 bytes | 282.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:supun-randeni/missions-StackUxV.git
   15e9fd6..fed544d  <your-branch-name> -> <your-branch-name>
```

2.2 Updating software on the vehicle

Now you can update missions-StackUxV on your vehicle embedded computers. You can use the update_missions_stackuxv script on the Raspberry Pi to update and rebuild software on both Raspberry Pi and PocketBeagle at once.

```
$ update_missions_stackuxv
```

3 Calibrating the AHRS

The magnetometers in the Attitude and Heading Reference System (AHRS) of the AUV can be affected by magnetic interference from other components in the vehicle. To obtain an accurate heading estimate, the impact of such disturbances must be mitigated. Internal, non-time-varying magnetic disturbances can be corrected using a hard and soft iron (HSI) calibration. Therefore, once the AUV is fully assembled, its AHRS must be re-calibrated.

3.1 Calibration

For optimal performance, the calibration should be conducted in an area with minimal magnetic disturbances. You can use the calibrate_xsens_mti3 program on the PocketBeagle to perform the calibration.

```
$ calibrate_xsens_mti3
```

You will be asked several questions both before and after the calibration. For example:

```
Press Y if you would like to log the data from your calibration dance . Ctrl-c to exit:
```

You do not necessarily need to log the calibration data to a file; hence you may enter N.

Once the calibration process begins, you will need to carry the AUV and rotate it in circles (i.e., changing its heading angle) while also varying its pitch and roll, reaching at least 30 degrees. Calibration data will be recorded for 60 seconds. Ensure the calibration is successful. Afterward, you will be prompted to enter the latitude and longitude of the test site for magnetic declination correction. The latitude and longitude of the MIT Sailing Pavilion are as follows:

- Latitude: 42.3584
- Longitude: -71.0874

3.2 Heading verification (instructor checkoff)

Once the calibration is complete, you can run the test_xsens_mti3 program to verify the heading measurement from the sensor. This program will output the current roll, pitch, and heading angles. When the vehicle's nose is pointed towards north, the sensor should read approximately 0 degrees.

4 Understanding indicator light patterns

The indicator LEDs on the AUV's mast blink in different patterns to indicate various states of the vehicle. It is important to understand what these patterns mean and what the vehicle is trying to communicate.

If your vehicle does not flash the mast LEDs, the daemon service, which should automatically run the `mitfs_safety_daemon` program as a background process on boot, may need to be enabled. In that case, please consult an instructor.

1. Main pressure hull has no vacuum:
   - **Red:** solid with a quick 100 millisecond blink, once every 3 seconds
   - **Green:** off
   - **White:** off
2. Main pressure hull vacuum is leaking:
   - **Red:** 1-second blink; i.e. 1 second on, 1 second off
   - **Green:** off
   - **White:** off
3. Main pressure hull vacuum is good. A mission is not launched. Vehicle is idle:
   - **Red:** 200-millisecond blink; i.e. 200 millisecond on, 200 millisecond off
   - **Green:** 200-millisecond blink; i.e. 200 millisecond on, 200 millisecond off
   - **White:** 200-millisecond blink; i.e. 200 millisecond on, 200 millisecond off
4. Mission has been launched, time to put the vehicle in the water:
   - **Red:** off
   - **Green:** off
   - **White:** 1-second blink; i.e. 1 second on, 1 second off
5. Mission has been launched, time to put the vehicle in the water. It is 10 seconds before the propeller starts spinning:
   - **Red:** off
   - **Green:** off
   - **White:** starts to blink faster
6. Mission is active:
   - **Red:** 1-second blink; i.e. 1 second on, 1 second off
   - **Green:** 1-second blink; i.e. 1 second on, 1 second off
   - **White:** solid

7. Mission is paused. Do not approach the AUV as it may resume the mission (or approach with caution):
    - **Red:** solid
    - **Green:** 1-second blink; i.e. 1 second on, 1 second off
    - **White:** 1-second blink; i.e. 1 second on, 1 second off
8. Mission is complete. You can recover the vehicle:
    - **Red:** 1-second blink; i.e. 1 second on, 1 second off
    - **Green:** solid
    - **White:** 1-second blink; i.e. 1 second on, 1 second off

5 Ballasting

Once the vehicle is fully assembled and the tether is attached, you need to (re)ballast the vehicle to ensure that the hydrostatic condition of the AUV is appropriate.

5.1 Vacuum check (self checkoff)

First, let's double-check that the vacuum is holding. You can run the `check_vacuum` program on the PocketBeagle to verify the vacuum level:

```
$ check_vacuum
```

5.2 In-water ballasting (instructor checkoff)

If the vacuum is holding, ballast the vehicle so that it is slightly positively buoyant, with level trim and list angles. Make sure the rope is tied to the dock (or held securely by your partner) while ballasting to avoid accidentally sinking the vehicle or pushing it under the dock.

6 Pre-launch system checks

The following pre-launch system checks should be completed before deploying the vehicle:

6.1 Verifying sensors and actuators

6.1.1 Test depth sensor (self checkoff)

Test if the depth sensor is working properly:

```
$ test_br_depth
```

6.1.2 Test AHRS (self checkoff)

Check if the AHRS is working properly, and providing you with sensible roll, pitch and heading angle measurements:

```
$ test_xsens_mti3
```

6.1.3 Test GPS (self checkoff)

Test if the GPS is working. You can unit test the GPS by using the linux minicom tool (https://linux.die.net/man/1/minicom). Minicom will display the NMEA strings that the GPS outputs.

```
$ minicom -D /dev/ttyO0 -b 9600 -w
```

To exit from minicom, press ctrl+A and then x.

6.1.4 Test control surfaces (self checkoff)

Test the control surfaces; and visually inspect if they are in good working order:

```
$ test_control_surfaces
```

6.1.5 Test thrusters (self checkoff)

Test the main thruster:

```
$ test_br_thruster
```

6.2 Verify vacuum (self checkoff)

Verify the vacuum level and make sure that there are no leaks:

```
$ check_vacuum
```

6.3 Verify the vehicle axes system (instructor checkoff)

When a new SeaBeaver AUV is built, it is important to verify that the servos driving the control surfaces are configured correctly; for example, ensuring that the control surfaces move in the correct direction.

This is achieved by running a bench testing mission called `bench_test`. First, let's inspect what this mission/cruise does. Every mission configured using the `configure_cruise.sh` script has two

important files associated with it located in the missions-StackUxV/cruise/<cruise-name> directory. For example, the bench_test cruise has the following two associated files:

1. missions-StackUxV/cruise/bench_test/cruise.def
2. missions-StackUxV/cruise/bench_test/cruise_plugs/backseat_plugs/pMITFS_MissionScript.plug

You can open these files using your preferred text editor to inspect them. The most important parameters in the cruise.def file are as follows:

```
// Mission related configurations ------------------------------------------------------
#define MISSION_START_TIME                10
#define MISSION_END_TIME                 300
```

These parameters define the mission's start and end times in seconds. The most important parameters in the pMITFS_MissionScript.plug file are as follows:

```
// Configuring the mission profile
add_mission: DEPLOY_MODE=CONSTANT_COURSE, mission_start_second=10, heading=150, speed=0.0, depth=1.5, duration_mi
add_mission: deploy_mode=constant_course, mission_start_minute=1,  heading=150,speed=0.0, depth=1.5, duration_mir
add_mission: deploy_mode=constant_course, mission_start_minute=2,  heading=150,speed=0.0, depth=1.5, duration_mir
```

Each add_mission: line in the file defines a portion of the overall mission—specifying parameters such as the mission type, start time, etc. In the bench_test mission, there are three constant_course runs. A constant course mission maintains a fixed heading, speed, and depth for a specified duration (or until the next mission begins). In this case, all three runs use the same heading, speed, and depth values. As a result, the vehicle will attempt to maintain a 150-degree heading, 1.5 m depth, and zero speed until the mission end time (i.e., 300 seconds), which is ideal for verifying whether the servos are operating in the correct axes system.

6.3.1 Configuring the cruise

Let's configure the architecture, vehicle and cruise. On the Raspberry Pi:

```
$ cd missions-StackUxV/launch_scripts
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh <your-vehicle-name>
$ ./configure_cruise.sh bench_test
```

Please note that you need to configure the vehicle (i.e. configure_vehicle.sh) with the name of your AUV.

6.3.2 Running the mission and inspecting the vehicle axes system

Now you are ready to launch the bench_test mission. You can launch it with the launch_runtime.sh script:

```
$ ./launch_runtime.sh
```

The mission should start in 10 seconds. The AUV will attempt to dive and steer towards a heading of 150 degrees. Rotate the vehicle and observe whether the control surface angle changes make sense; that is, whether they are attempting to correct the course appropriately. If the responses do not make sense, you may need to adjust the axis configuration of the control surfaces in your vehicle definition file:

```
// Actuators axis convention
#define TOP_RUD_POSITIVE_INCREASE_HEADING true
#define STBD_ELV_POSITIVE_NOSE_UP         false
#define PORT_ELV_POSITIVE_NOSE_UP         true
```

Your AUV's axis system must be verified and approved by an instructor before proceeding to the next step.

6.3.3 Adjusting PID gains

If the axes of the control surfaces are correct, but their response when you turn the AUV seems unreasonable, you may need to adjust the PID gains. It is recommended to begin with simple PID settings; for example:

```
// ********************************************************************************
// 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
#define PROP_MODE_ROLL_KI                 0
#define PROP_MODE_ROLL_KD                 0
#define PROP_MODE_ROLL_MAX_INTEGRAL       10

#define PROP_MODE_PITCH_KP                0.5
#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                0.2
#define PROP_MODE_DEPTH_KD                0
#define PROP_MODE_DEPTH_MAX_INTEGRAL      6

#define PROP_MODE_HEADING_KP              0.3
#define PROP_MODE_HEADING_KI              0
#define PROP_MODE_HEADING_KD              0
#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
```

Please note that if you decide to change the PID gains, you should change them on your topside laptop, commit and push the changes to the github remote server, and pull them to the Raspberry Pi and PocketBeagle.

7 First in-water deployment

The first sea trial of the vehicle is a critical step in the AUV development process. It is common to repeat the initial mission several times to debug and resolve any issues the vehicle may have. To ensure safety, we typically begin with a very short and simple mission for the initial in-water deployment, named `a_confidence`. Like any other mission (e.g., the `bench_test` mission), `a_confidence` also has two associated cruise definition files:

1. `missions-StackUxV/cruise/a_confidence/cruise.def`
2. `missions-StackUxV/cruise/a_confidence/pMITFS_MissionScript.plug`

You can use your preferred text editor to examine these files. In particular, take a look at the start and end times defined in the `cruise.def` file:

```
// Mission related configurations ------------------------------------------------------------
#define MISSION_START_TIME                60
#define MISSION_END_TIME                 120
```

As you can see, the propeller will start spinning 60 seconds after the mission is launched, and the mission will end at 120 seconds, providing one minute of flight time. The mission parameters are defined in the pMITFS_MissionScript.plug file:

```
// Configuring the mission profile
add_mission: DEPLOY_MODE=CONSTANT_COURSE, mission_start_second=60, heading=150, speed=0.8, depth=1.5, duration_mi
```

As you can see, the mission profile contains only one constant course segment, which commands the vehicle to maintain a heading of 150 degrees, a depth of 1.5 meters, and a speed of 0.8 m/s.

7.1 Simulating the mission (self checkoff)

It is always recommended to simulate the mission before running it in the water, using both software-in-the-loop and hardware-in-the-loop methods. At this stage, you should already be familiar with how to run these simulations

7.1.1 Software-in-the-loop simulations

On the topside computer, you can use the following commands to configure and simulate the mission:

```
$ cd missions-StackUxV/launch_scripts
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh <your-vehicle-name>
$ ./configure_cruise.sh a_confidence
$ ./launch_simulation.sh
```

Please note that you need to configure the vehicle (i.e. configure_vehicle.sh) with the name of your AUV.

7.1.2 Hardware-in-the-loop simulations

As you know at this stage, running a mission in hardware-in-the-loop simulations is a two step process:

1. Configuring and running the topside community.
2. Configuring and running the vehicle communities.

Following commands can be used on the topside computer to configure and run the topside community:

```
$ cd missions-StackUxV/launch_scripts
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh <your-vehicle-name>
$ ./configure_cruise.sh a_confidence
$ ./launch_topside.sh
```

Following commands can be used on the Raspberry to configure and run the vehicle backseat and frontseat communities:

```
$ cd missions-StackUxV/launch_scripts
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh <your-vehicle-name>
$ ./configure_cruise.sh a_confidence
$ ./launch_hitl.sh --topside=<topside-ip-address>
```

When you are running hardware-in-the-loop simulations on the actual AUV, please do not run the thruster for more than 20 seconds continuously. Prolonged operation in air can lead to overheating of the thruster.

Based on the direction in which air is pushed by the propeller, you can determine whether the propeller's rotation direction — clockwise or counter-clockwise — is configured correctly. If the propeller is not rotating in the correct direction, please notify an instructor.

7.2 Runtime launch (instructor checkoff)

When you are satisfied with the mission and the vehicle's performance, and you are confident in its expected behavior, you can coordinate with the instructors to conduct the first in-water experiment.

Configure the vehicle using following commands on the Raspberry Pi:

```
$ cd missions-StackUxV/launch_scripts
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh <your-vehicle-name>
$ ./configure_cruise.sh a_confidence
```

Please note that you need to configure the vehicle (i.e. `configure_vehicle.sh`) with the name of your AUV.

When you are ready to deploy the vehicle, ensure that a rope is attached and the river is clear of boat traffic, then launch the mission using the following command:

```
$ ./launch_runtime.sh
```

Now you have 60 seconds to place the vehicle in the water. The white LED indicator will start to blink faster 10 seconds before the propeller start to spin.

Please treat the thruster as if it is live at all times. Do NOT place your fingers near the propeller!!!

7.2.1 Recovering the vehicle

When the vehicle completed the mission (i.e. 60 seconds after the propeller started spinning in this case), the AUV will surface, and you may recover it.

7.2.2 Killing the mission

Once you bring the AUV back to the lab (i.e., close to the router), it will automatically reconnect to the SeaBeaverNetwork Wi-Fi network. The terminal window on your topside laptop will resume, and you can enter `ktstack` to terminate the mission on both the Raspberry Pi and PocketBeagle.

7.3 Post-mission data analysis

Now, we can download the vehicle log files from the Raspberry Pi to your topside laptop using the `scp` command, and visualize/analyze the data with the `alogview` tool. First, create a directory on your topside laptop to store your logs. For example, I have created a folder named `bee_logs`, and within it, a sub-folder labeled with today's date:

```
$ cd ~
$ mkdir <your-vehicle-name>_logs
$ cd <your-vehicle-name>_logs
$ mkdir 20260507
$ cd 20260507
```

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

```
$ scp -r seabeaver-raspi@<your-vehicle-IP>:/home/seabeaver-raspi/missions-StackUxV/logs/LOG* .
```

Now you can visualize these logs using `alogview`, as you have done in previous labs. One interesting measurement that you might want to observe is the variation of water temperature at different depths and locations in the river. This is logged under `EXTERNAL_TEMPERATURE` MOOS variable.

8 Second in-water deployment

When you are confident in your vehicle, you can attempt a more ambitious mission named `b_zigzag`. This is a 7-minute-long mission during which the vehicle performs several different constant course runs. You can review the mission profile by inspecting the `missions-StackUxV/cruise/b_zigzag/cruise.def` and `missions-StackUxV/cruise/pMITFS_MissionScript.plug` files.

It is recommended to simulate the mission using both software-in-the-loop and hardware-in-the-loop methods before launching it in the water. You can follow the steps outlined in Section 7.1 to do this. Once you are satisfied with the mission and have a clear understanding of what to expect, coordinate with the instructors to carry out the `b_zigzag` mission. Configure the vehicle using the following commands on the Raspberry Pi:

```
$ cd missions-StackUxV/launch_scripts
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh <your-vehicle-name>
$ ./configure_cruise.sh b_zigzag
```

When you are ready to deploy the vehicle, and the river is clear of boat traffic, launch the mission using the following command:

```
$ ./launch_runtime.sh
```

Once you recovered the vehicle, kill the mission, download the logs, and analyze the data.

9 Tuning the low-level PID control system

At this stage, you may choose to further tune the PID gains based on the observed control performance. Make the changes on your topside laptop, then commit and push them to the GitHub remote server. Finally, pull the updates to both the Raspberry Pi and PocketBeagle.