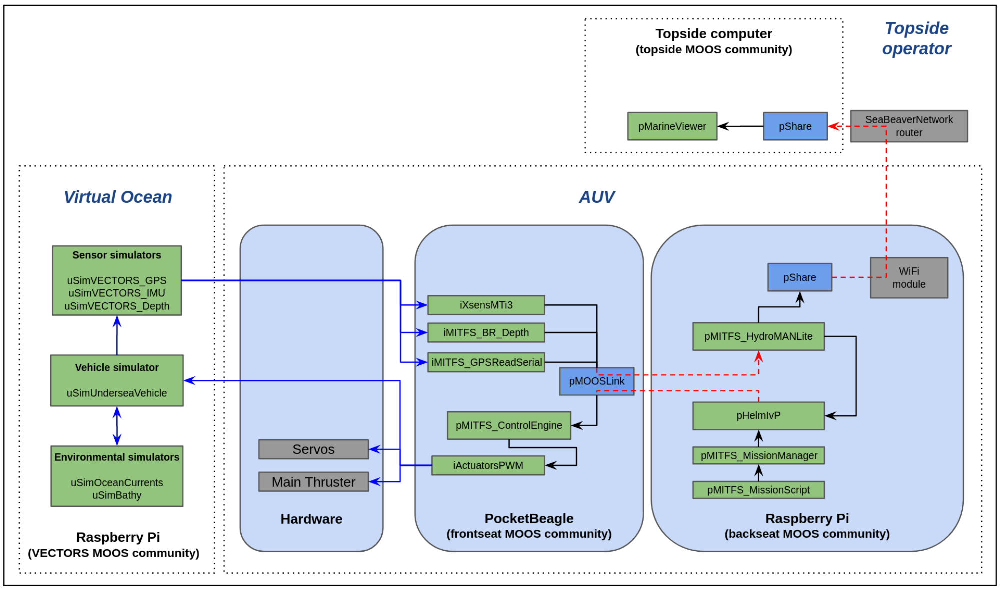
Lab 5 - Software and Hardware in the Loop Simulations

2.S01 Introduction to Autonomous Underwater Vehicles



Spring 2025

Supun Randeni, supun@mit.edu
Michael Benjamin, mikerb@mit.edu
Mike Sacarny, msacarny@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Objectives

- Setting up your topside computer: downloading and building the latest versions of the StackUxV, StackUxV-drivers, VECTORS and missions-StackUxV software trees.
- Setting up the Raspberry Pi in your training-kit: updating and rebuilding the StackUxV, StackUxV-drivers, VECTORS and missions-StackUxV software trees.
- Setting up the PocketBeagle in your training-kit: updating and rebuilding the StackUxV, StackUxV-drivers, VECTORS and missions-StackUxV software trees.
- Learning how to run software-in-the-loop (SITL) simulations on your topside laptop.
- Learning how to run hardware-in-the-loop (HITL) simulations using the training-kit and topside laptop.

2 Building the software stack on the topside computer

SeaBeaver III AUVs utilize several MIT software projects. Before we download and build these software on your topside laptop, it is helpful to have a basic understanding of their functions:

1. **moos-ivp**: This open-source software project includes our middleware, `MOOS`, and autonomy helm software, `pHelmIvP`. It also provides several utility tools such as `uXMS`, `uPokeDB`, `pMarineViewer`, `alogview`, and others that we will use throughout this class. We will build `moos-ivp` from source code, downloaded from a Git repository. `moos-ivp` is a dependency for all following software projects.

2. **VECTORS**: This is our virtual ocean environment software. Although it is not used during runtime (i.e., when the vehicle is deployed in water), we will use `VECTORS` for software-in-the-loop and hardware-in-the-loop simulations. In 2.S01, we will not download the VECTORS source code; instead, we will install it via a Linux package distribution.

3. **StackUxV**: This is our primary embedded software stack, which includes mission planning (`pMITFS_MissionScript`), mission commanding (`pMITFS_MissionManager`), underwater navigation (`pMITFS_HydroMANLite`), and control (`pMITFS_ControlEngine`) software. We will build `StackUxV` from source code, downloaded from a private Git repository.

4. **StackUxV-drivers**: This repository contains low-level software drivers that interface with the SeaBeaver AUV's hardware components. Examples include the AHRS driver `iXsensMTi3`, actuator driver `iActuatorsPWM`, and GPS driver `iMITFS_GPSReadSerial`. We will build `StackUxV-drivers` from source code, downloaded from a private Git repository.

5. **missions-StackUxV**: This is our mission launching utility, which we discussed during our lab. It primarily includes configuration files for various MOOS apps, along with shell scripts designed to support seamless mission execution. As this repository does not contain C++ code, no building is necessary. It is downloaded from a private Git repository.

Some of the software projects listed above are hosted in private Git repositories. The class topside laptops provided to you are already configured with access to these repositories. However, if you are using a different laptop or a different user account, you will need to be granted access. Please provide one of the following to Supun:

- The public key of the SSH key pair generated on your laptop; or
- Your username in https://github.com/

If you would like to retain continuous access to these repositories beyond the 2.S01 class, please speak with Supun.

Some of these software projects have their own dependencies. For example, StackUxV requires the Google Protocol Buffers and Boost C++ libraries, while StackUxV-drivers depends on the Robot Control Library. Dependencies must be installed before building our software.

2.1 Building moos-ivp

At this point, you should have already completed Lab 01: Course Laptop Setup (https://oceanai.mit.edu/2.S01/labs). We assume that you have successfully built moos-ivp on your topside laptop and are able to run the Alpha Mission example. If not, please complete Lab 01: Course Laptop Setup before proceeding.

2.2 Installing VECTORS and building StackUxV

For the 2.S01 class, the VECTORS software package is provided to you with StackUxV repository; therefore, we need to download StackUxV first. Your topside and embedded computers are given permission to access StackUxV via https://github.com/supun-randeni/StackUxV. First you need to clone this repository to your home directory (e.g. /home/sb-topside-4/)

```
$ cd ~
$ git clone git@github.com:supun-randeni/StackUxV.git
```

If you successfully cloned StackUxV to your topside computer, you will see something similar to this.

```
sb-topside-4@sb-topside-4:~$ git clone git@github.com:supun-randeni/StackUxV.git
Cloning into 'StackUxV'...
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/supun/.ssh/known_hosts:4
Matching host key in /home/supun/.ssh/known_hosts:60
Are you sure you want to continue connecting (yes/no)? yes
remote: Enumerating objects: 1009, done.
remote: Counting objects: 100% (245/245), done.
remote: Compressing objects: 100% (158/158), done.
remote: Total 1009 (delta 154), reused 163 (delta 87), pack-reused 764 (from 1)
Receiving objects: 100% (1009/1009), 41.70 MiB | 2.39 MiB/s, done.
Resolving deltas: 100% (703/703), done.
```

However, if you get the following error message, your permission is not properly set. Please contact an instructor:

```
sb-topside-4@sb-topside-4:~$ git clone git@github.com:supun-randeni/StackUxV.git
Cloning into 'StackUxV'...
Warning: Permanently added the ECDSA host key for IP address '140.82.114.3' to
the list of known hosts.
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

Once you cloned successfully, if you use the `ls` command, you can see `StackUxV` in your home directory:

```
$ cd ~
$ ls
Desktop Documents   Downloads   StackUxV    moos-ivp
Music   Pictures    Public  snap    Templates   Videos
```

Let's go to `StackUxV` and explore:

```
$ cd ~/StackUxV/
$ ls
build.sh        cmake_modules                  README.md            scripts
CMakeLists.txt        COPYING-StackUxV.txt  release-notes.txt  src
```

The Git version control system allows us to create branches within the software, enabling development of new features or modifications without affecting the main code in the `master` branch or disrupting others' workflows. We encourage you to learn more about version control, Git, and Git branching in your own time.

`StackUxV` is actively used by several MIT and external programs. To avoid interfering with those projects, we've created a dedicated branch called `2025_2s01` specifically for this class. You can check your current branch by running the `git branch` command. If you're on the `master` branch, you'll see an output similar to the following:

```
$ git branch
* master
```

Let's switch to the `2025_2s01` branch by using the command `git checkout 2025_2s01`. If you successfully switched to the branch, you will see an output similar to this:

```
$ git checkout 2025_2s01
Switched to branch '2025_2s01'
Your branch is up to date with 'origin/2025_2s01'.
```

You can use the `git pull` command to update the repository (i.e. to get the latest version of the code).

```
$ git pull
Already up to date.
```

Now that we are on the `2025_2s01` branch with the latest code, the next step is to build `StackUxV`. Before doing so, we need to install its dependencies, including `VECTORS`. To simplify this process, we have provided an automated setup script located in the `StackUxV/scripts` directory. Navigate to that directory and run the script as follows:

```
$ cd ~/StackUxV/scripts/
$ sudo ./dependencies.sh
[sudo] password for sb-topside-4:
```

When prompted, please enter the password for the topside computer. Installing the dependencies may take several minutes. Once the installation is complete, you can verify that `VECTORS` was successfully installed by running the command `which iVECTORS_Gateway`. If the installation was successful, you should see an output similar to the following. If the command returns nothing, please contact an instructor.

```
$ which iVECTORS_Gateway
/usr/local/bin/iVECTORS_Gateway
```

Now you have installed all the dependencies. It is time to build `StackUxV`:

```
$ cd ~/StackUxV/
$ ./build.sh
```

Let it build.. While it is building, check if it exits with errors.

2.3 Building `StackUxV-drivers`

This repository contains low-level software drivers that interface with the SeaBeaver AUV's hardware components. Both your topside and embedded computers have been granted access to `StackUxV-drivers` via https://github.com/supun-randeni/StackUxV-drivers. Similarly to `StackUxV`, you first need to clone this repository into your home directory (e.g. `/home/sb-25-topside-4/`)

```
$ cd ~
$ git clone git@github.com:supun-randeni/StackUxV-drivers.git
```

If you successfully cloned `StackUxV-drivers` to your topside computer, you will see it in your home directory:

```
$ cd ~
$ ls
Desktop    Downloads    StackUxV-drivers  Music   Public  Templates
Documents  StackUxV  moos-ivp   Pictures  snap  Videos
```

Let us go to `StackUxV-drivers` and explore:

```
$ cd ~/StackUxV-drivers/
$ ls
build.sh    cmake_modules   README.md   scripts
CMakeLists.txt  COPYING-StackUxV-drivers.txt  release-notes.txt  src
```

Just like with `StackUxV`, we need to switch to the `2025_2s01` branch of `StackUxV-drivers` by running the command `git checkout 2025_2s01`. If the branch switch is successful, you'll see output similar to the following:

```
$ git checkout 2025_2s01
Branch '2025_2s01' set up to track remote branch '2025_2s01' from 'origin'.
Switched to a new branch '2025_2s01'
```

You can use the `git pull` command to update the repository (i.e. to get the latest version of the code).

```
$ git pull
Already up to date.
```

Let's install the dependency software packages of `StackUxV-drivers` by running the `dependencies.sh` script located in in `StackUxV-drivers/scripts` directory:

```
$ cd ~/StackUxV-drivers/scripts/
$ sudo ./dependencies.sh
[sudo] password for sb-topside-4:
```

When prompted, please enter the password for the topside computer. The dependency installation may take several minutes. During installation, you may see a screen similar to the one in Figure 1 while the Robot Control Library is being installed. When this happens, please select the option `none`.

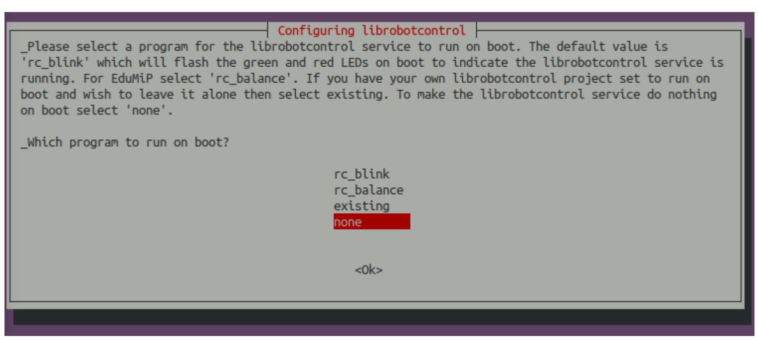


Figure 1: Configuring librobotcontrol pop-up window

When it is done, you can check if librobotcontrol is successfully installed by using the command `which rc_test_drivers`. It should give an output similar to follows. If you get a blank response, please talk to an instructor.

```
$ which rc_test_drivers
/usr/bin/rc_test_drivers
```

Now you have installed all the dependencies. It is time to build `StackUxV-drivers`:

```
$ cd ~/StackUxV-drivers/
$ ./build.sh
```

Let the build process run. While it's building, monitor the output to check if it exits with any errors.

2.4 Downloading `missions-StackUxV`

`missions-StackUxV` is our mission launching utility. Your topside and embedded computers have been granted access to `missions-StackUxV` via https://github.com/supun-randeni/missions-StackUxV. This repository contains only shell scripts and MOOS configuration files, so there is no need to build the code. Simply clone the repository into your home directory (e.g., `/home/sb-topside-4/`).

```
$ cd ~
$ git clone git@github.com:supun-randeni/missions-StackUxV.git
```

If you successfully cloned `missions-StackUxV` to your topside computer, you will see it in your home directory:

```
$ cd ~
$ ls
Desktop Downloads    StackUxV   moos-ivp    Pictures    snap    Videos
Documents   missions-StackUxV   StackUxV-drivers Music  Public   Templates
```

Let's navigate to the `missions-StackUxV` directory and explore its contents:

```
$ cd ~/missions-StackUxV/
$ ls
architecture   clean.sh  launch_scripts  plugs              vehicle
build_scripts  cruise    logs            README.md          virtual_experiment
clean_logs.sh  fleet     meta            release-notes.txt
```

Finally, switch to the `2025_2s01` branch by running the command `git checkout 2025_2s01`. If the branch switch is successful, you will see an output similar to the following:

```
$ git checkout 2025_2s01
Branch '‘2025_2s01’ set up to track remote branch ‘2025_2s01’ from ‘origin’.
Switched to a new branch ‘2025_2s01’
```

You can use the `git pull` command to update the repository (i.e. to get the latest version of the code).

```
$ git pull
Already up to date.
```

2.5 Adding to shell path

The locations of the `StackUxV` and `StackUxV-drivers` binaries need to be added to your shell path. Additionally, we must inform the dynamic link loader of the locations of the `StackUxV` and `StackUxV-drivers` libraries by adding them to the `LD_LIBRARY_PATH`. To simplify this, we have created a text file with the necessary information: `StackUxV-drivers/scripts/dot_bash_seabeaver`. All you need to do is call this file from your `.bashrc`, and `dot_bash_seabeaver` will handle the rest.

Edit your `.bashrc` file using your favorite text editor, and append the following lines to the end of the file:

```
if [ -f ~/StackUxV-drivers/scripts/dot_bash_seabeaver ]; then
    . ~/StackUxV-drivers/scripts/dot_bash_seabeaver
fi
```

Don't forget to source the `.bashrc` file.

2.6 Self checkoff assessment

1. Ensure that `StackUxV` is built correctly and that its binaries are included in your shell path by running the following command:

   ```
   $ which iVECTORS_MITFS pMITFS_ControlEngine pMITFS_HydroMANLite pMITFS_MissionManager
   ```

   It should output something similar to this:

   ```
   $ which iVECTORS_MITFS pMITFS_ControlEngine pMITFS_HydroMANLite pMITFS_MissionManager

   /home/sb-topside-4/StackUxV/bin/iVECTORS_MITFS
   /home/sb-topside-4/StackUxV/bin/pMITFS_ControlEngine
   /home/sb-topside-4/StackUxV/bin/pMITFS_HydroMANLite
   /home/sb-topside-4/StackUxV/bin/pMITFS_MissionManager
   ```

2. Ensure that `StackUxV-drivers` is built correctly and that its binaries are included in your shell path by running the following command:

   ```
   $ which calibrate_xsens_mti3 test_br_thruster iActuatorsPWM iXsensMTi3 mitfs_safety_daemon
   ```

   It should output something similar to this:

```
$ which calibrate_xsens_mti3 test_br_thruster iActuatorsPWM iXsensMTi3 mitfs_safety_daemon

/home/sb-topside-4/StackUxV-drivers/bin/calibrate_xsens_mti3
/home/sb-topside-4/StackUxV-drivers/bin/test_br_thruster
/home/sb-topside-4/StackUxV-drivers/bin/iActuatorsPWM
/home/sb-topside-4/StackUxV-drivers/bin/iXsensMTi3
/home/sb-topside-4/StackUxV-drivers/bin/mitfs_safety_daemon
```

2.7 Extra activities – running the GPS navigation mission with real-time topside viewer

Now that you have installed our software stack on your topside laptop, you can run the GPS navigation mission alongside the topside system to track the position of the training-kit in real time.

We typically launch the Topside MOOS community before launching the backseat or frontseat MOOS communities. Before launching the topside, we need to configure the architecture, vehicle and cruise on the topside laptop:

```
$ cd ~/missions-StackUxV/launch_scripts/
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_vehicle.sh cap
$ ./configure_cruise.sh lab_4_do_nothing_log_data
```

Now we can launch the topside:

```
$ ./launch_topside.sh
```

the `pMarineViewer` window should open. However, you will not see any vehicles until you launch the backseat and frontseat.

Follow the instructions provided in "Lab 4 – Surface Navigation Using GPS" to launch the `lab_4_do_nothing_log_data` mission on the Raspberry Pi.

```
$ cd ~/missions-StackUxV/launch_scripts/
$ ./configure_architecture.sh seabeaver_iii
$ ./configure_cruise.sh lab_4_do_nothing_log_data
$ ./configure_vehicle.sh cap
$ ./launch_runtime.sh --topside=10.42.0.1
```

When executing the launch script, note that we have appended the IP address of your topside laptop; e.g. `10.42.0.1` in my case.

After a few seconds, a vehicle icon named `CAP` should appear on the `pMarineViewer` window. As you change the orientation of your training-kit, the icon should dynamically update its orientation in real time to reflect those changes.

3 Software-in-the-loop (SITL) simulations

The `missions-StackUxV` launching utility is designed to support mission execution in both simulation and runtime modes. Missions can be tested using software-in-the-loop (SITL) simulations on the topside laptop or hardware-in-the-loop (HITL) simulations on the actual vehicle. This approach allows us to verify mission setup and hardware functionality before field deployment. In SITL simulations (see Figure 2), the frontseat, backseat, topside and VECTORS MOOS communities are all launched on the topside laptop computer.

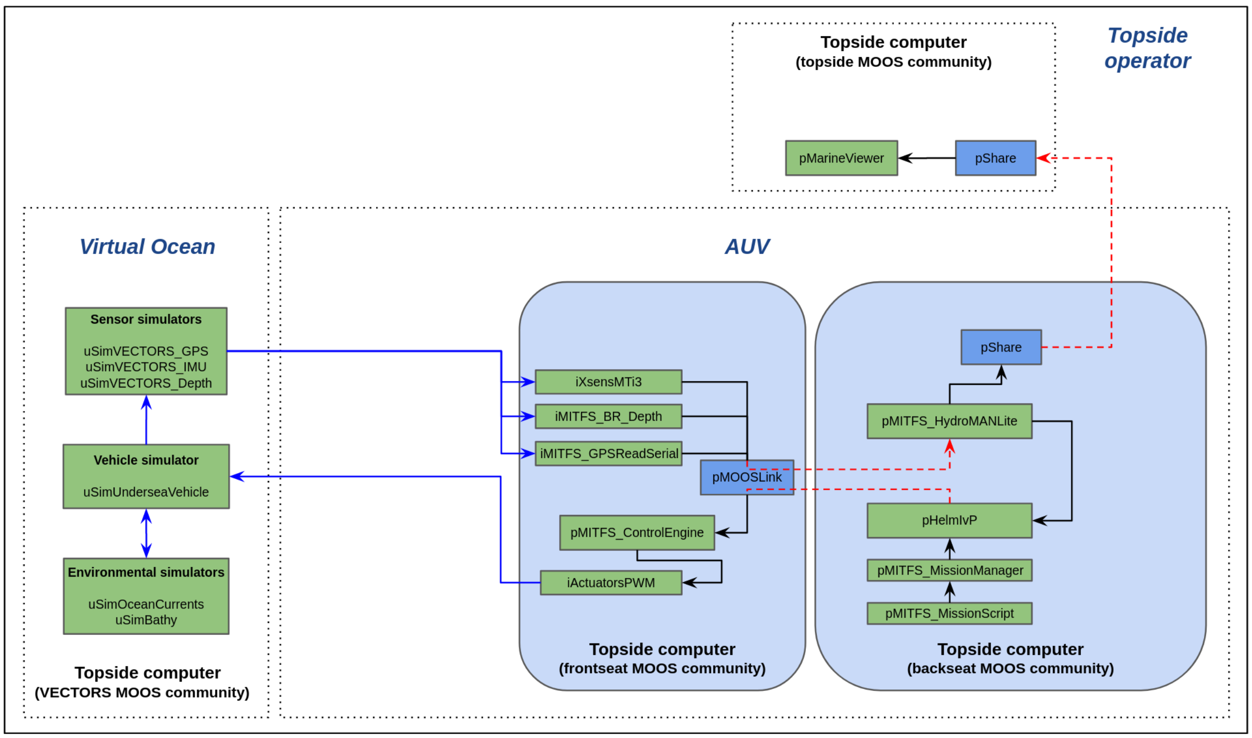


Figure 2: In software-in-the-loop simulations, frontseat, backseat, topside and VECTORS MOOS communities are all launched on the topside computer

3.1 Defining the architecture, vehicle and cruise

As discussed in the previous lab, you must configure the architecture, vehicle, and cruise before launching a mission using `missions-StackUxV`. This requirement applies regardless of whether you are running a SITL simulation, HITL simulation, or a runtime deployment.
On your topside laptop, select the following configurations. If you need a refresher, refer to Lab 4 – Surface Navigation Using GPS. Keep in mind that, since you are running a software-in-the-loop simulation, all configurations should be performed on the topside laptop, not the Raspberry Pi.

1. Set the vehicle architecture to: `seabeaver_iii`
2. Set the vehicle to: `cap`
3. Set the cruise to: `shipwreck_search`

Configuring and launching is done in the /missions-StackUxV/launch_scripts/ directory of the topside laptop.

3.2 Launching a software-in-the-loop simulated mission

Now you can launch the mission as a software-in-the-loop (SITL) simulation by running the launch_simulation.sh script:

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

In the terminal, you will see an output similar to this:

```
$ ./launch_simulation.sh
Launching Frontseat-Backseat mode..
SITL launch in process..
Launching mode is: FRONTBACK_FRONT
Current HydroMAN version: lite
Assembling targ_Frontseat.moos ... Done
Assembling targ_Frontseat.bhv ... Done
Launching Frontseat MOOS Community. WARP is 1
Done
SITL launch in process..
Launching mode is: FRONTBACK_BACK
Current HydroMAN version: lite
Assembling targ_Backseat.moos ... Done
Assembling targ_Backseat.bhv ... Done
Launching Backseat MOOS Community. WARP is 1
Done
Launching VECTORS MOOS Community. WARP is 1
Cleaning previous targ files..
~/missions-StackUxV/meta/meta_vectors ~/missions-StackUxV/launch_scripts
Cleaning previous targ files COMPLETE!
Populating VECTORS meta segments for the node  CAP
Populating VECTORS meta segments for the node  CAP COMPLETE!
Building the VECTORS master meta file
Building the VECTORS master meta file COMPLETE!
Launching VECTORS MOOS Community. WARP is 1
Launching VECTORS MOOS Community. WARP is 1  COMPLETE!
Use 'screen -r VECTORS' command to attach to VECTORS uMAC terminal.
TOPSIDE launch in process..
Launching mode is:
Current HydroMAN version: lite
Assembling targ_Topside.moos ... Done
Launching Topside MOOS Community. WARP is 1
Done
Launch is complete..
Use 'screen -ls' command to list all screen sessions running on this computer.
```

Two pMarineViewer windows will appear shortly—one associated with the topside MOOS community, and the other with the vehicle's backseat MOOS community. The mission will automatically begin after 30 seconds and run for a total of 10 minutes. During this mission, the vehicle will go to the middle of the river, transit between several predefined waypoints, and loiter at the final waypoint until the 10-minute runtime concludes. While observing the mission, pay attention to the following:

- You will see two vehicle icons: one labeled `CAP`, and the other `CAP_GT`. Both vehicles start at the same location, but gradually diverge as the mission progresses. The `GT` in `CAP_GT` stands for *ground truth*, representing the actual position of the simulated vehicle, while `CAP` shows the estimated position computed by the onboard navigation system, specifically the `pMITFS_HydroMANLite` application. We will explore the reasons behind this divergence and strategies to minimize it during our underwater navigation session.
- You will also observe that the vehicle does not follow a clear straight-line pattern between waypoints and instead exhibits a wiggling motion. During our controls session, we will discuss the underlying reasons for this behavior and explore strategies to minimize it.
- Observe the heading, speed, and depth values of the vehicle displayed at the bottom of the `pMarineViewer` window.

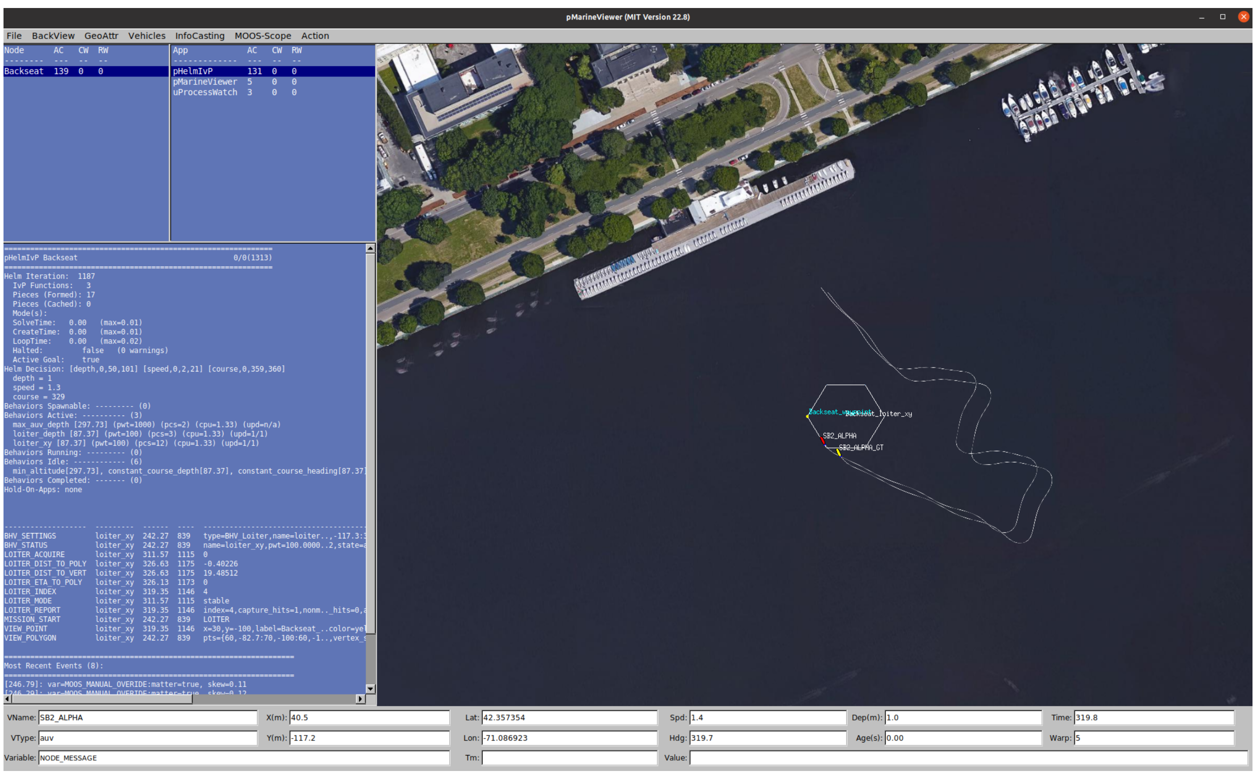


Figure 3: `pMarineViewer` window for the `shipwreck_search` mission

While the mission is running, observe that there are four MOOS communities active by listing the screen sessions using the `screen -ls` command. You can try attaching to and detaching from the `uMAC` applications of the different MOOS communities to monitor their real-time status (hint: this was discussed in the previous lab).

3.3 Killing the mission

Since the software is running in the background, you cannot use `Ctrl+C` to stop it. Instead, you must use the `ktstack` command (short for "kill-the-stack") to terminate all running processes. It is good practice to run `ktstack` before launching a new mission to ensure that there are no unwanted processes lingering in the background.

3.4 Speeding up simulations

The software-in-the-loop simulations can be sped-up using time warping, which allows missions to run faster than real time. To do this, simply append the desired time warp factor at the end of the `./launch_simulation.sh` command. For example, to run the simulation at 5× real-time speed, use the following command:

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

Please do NOT time warp hardware-in-the-loop simulations since the servo motors and the thruster can be damaged.

3.5 Self checkoff assessment

- Are you able to run the software-in-the-loop simulated mission on your topside computer?
- Can you run the same simulated mission in higher time warps?

3.6 Extra activities - modifying the mission

When you list available cruises or missions using the `./configure_cruise.sh` command, the script works by listing the directories inside `missions-StackUxV/cruise`. When you select a specific cruise (e.g., `shipwreck_search`), the script creates a symbolic link named `current_cruise`, which points to the `shipwreck_search` directory. A symbolic link is essentially a file that acts as a reference or shortcut to another file or directory, similar to a shortcut in Windows.

Navigate to the `missions-StackUxV/cruise/shipwreck_search` directory and explore the files inside. The two key files that define the mission-related parameters of the cruise are:

- `cruise.def`
- `pMITFS_MissionScript.plug`

These files specify the vehicle's configuration and behavior during the mission. Open and review them to understand how the mission is structured. Try modifying parameters such as the waypoint local_x, local_y, loiter radius, or speed to change the mission profile. You can go back to the original state of the mission by using following git operation:

```
$ git reset --hard
```

4 Building the software stack on the embedded computers

The Raspberry Pi and PocketBeagle computers in your training kit already have `moos-ivp`, `StackUxV`, `StackUxV-drivers`, and `missions-StackUxV` downloaded and built. (For your reference, we followed the same steps outlined in Section 2.) However, since the instructors continuously update the code as the class progresses, it is good practice to update and re-build `StackUxV`, `StackUxV-drivers`, and `missions-StackUxV` on both the Raspberry Pi and PocketBeagle before starting each lab.

To simplify the code update process, we've provided a utility called `build_stackuxv`, which automatically updates and rebuilds `StackUxV`, `StackUxV-drivers`, and `missions-StackUxV` on both the Raspberry Pi and PocketBeagle. This program is located in the `/missions-StackUxV/build_scripts` directory. Since this directory is already included in the Raspberry Pi's shell path, you can run the script from anywhere on the Raspberry Pi. When executed on the Raspberry Pi, it will update and rebuild the code on both the Raspberry Pi and the PocketBeagle. However, if run directly on the PocketBeagle, it will update and rebuild only on the PocketBeagle.

Log into your training kit's Raspberry Pi embedded computer using either wired or wireless approach and run the `build_stackuxv` program. You will see an output similar to this:

```
$ build_stackuxv

Building StackUxV on SeaBeaver Raspberry Pi..

StackUxV is available.
StackUxV is in branch:  2025_2s01

StackUxV-drivers is available.
StackUxV-drivers is in branch:  2025_2s01

missions-StackUxV is in branch:  2025_2s01

Updating StackUxV..
Already up to date.

Updating StackUxV-drivers..
Already up to date.

Updating missions-StackUxV..
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 15 (delta 12), reused 14 (delta 12), pack-reused 1 (from 1)
Unpacking objects: 100% (15/15), 1.61 KiB | 51.00 KiB/s, done.
From github.com:supun-randeni/missions-StackUxV
```

Let it update and build.. At the end, you will be prompted to enter the PocketBeagle's password. This is necessary to grant root privileges to certain hardware driver applications such as `iActuatorsPWM`, `test_control_surfaces`, and `test_br_thruster`, allowing them to access the PocketBeagle's hardware modules.

4.1 Self checkoff assessment

1. Are you able to update and rebuild `StackUxV`, `StackUxV-drivers`, and `missions-StackUxV` on both the Raspberry Pi and PocketBeagle?

5 Hardware-in-the-loop (HITL) simulations

Hardware-in-the-loop (HITL) simulations allow us to validate most software and hardware components at a lower cost, reducing both the need for in-water testing and the risk of costly failures. In SeaBeaver HITL simulations, we aim to run software on the same hardware it will use during real deployments to minimize discrepancies. That is:

- The frontseat MOOS community runs on the PocketBeagle.
- The backseat MOOS community runs on the Raspberry Pi.
- The topside MOOS community runs on the topside computer.
- The VECTORS MOOS community runs on the Raspberry Pi (this is the only deviation from real life deployments).

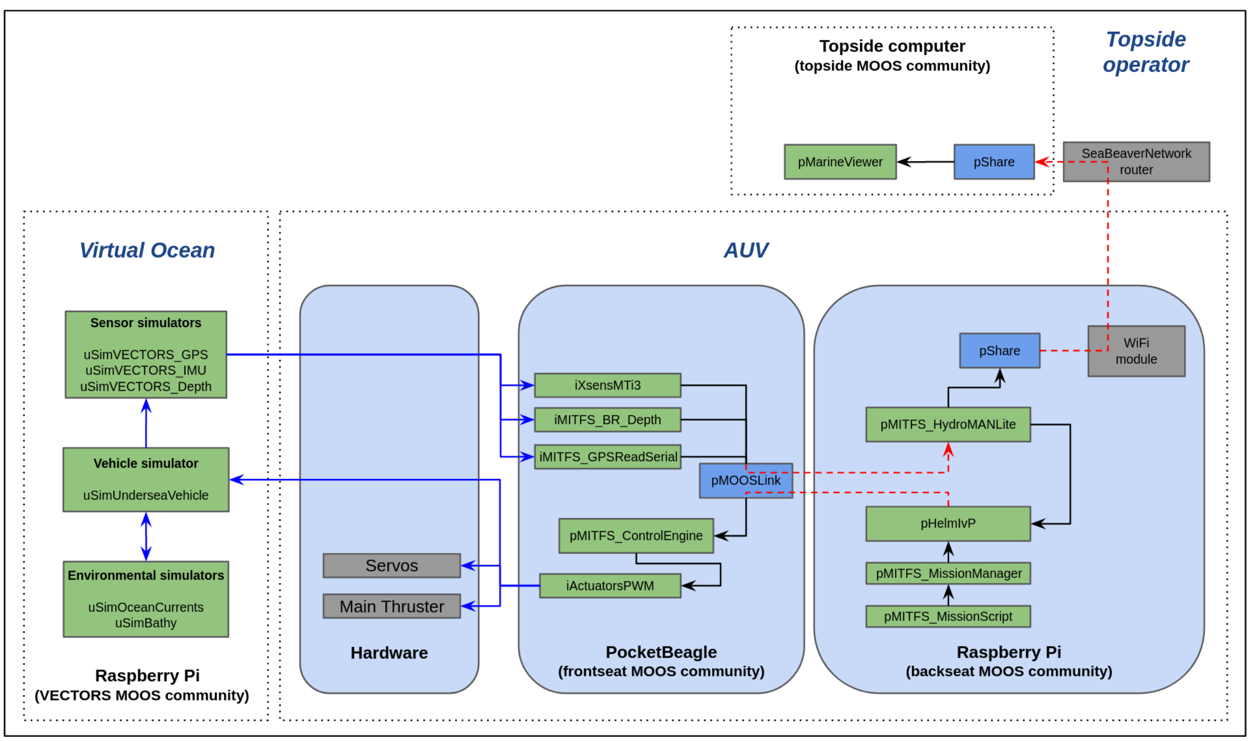


Figure 4: Four MOOS communities are launched during hardware-in-the-loop simulations

5.1 Defining the architecture, vehicle and cruise

As discussed in the previous lab, you must configure the architecture, vehicle, and cruise before launching a mission using `missions-StackUxV`. This requirement applies regardless of whether you are running a SITL simulation, HITL simulation, or a runtime deployment.
When running a hardware-in-the-loop (HITL) simulation, the configuration setup must be performed on **both** the topside laptop and the Raspberry Pi. On your topside laptop, begin by selecting the following configurations (if you need a refresher, refer to Lab 4 – Surface Navigation Using GPS):

1. Set the vehicle architecture to: `seabeaver_iii`
2. Set the vehicle to: `cap`
3. Set the cruise to: `shipwreck_search`

Then log into the Raspberry Pi and apply the same configurations on the Raspberry Pi.

5.2 Launching the topside

We typically launch the topside MOOS community before launching the backseat or frontseat MOOS communities. On your topside laptop, navigate to `/missions-StackUxV/launch_scripts` to launch the topside MOOS community:

```
$ ./launch_topside.sh
```

the `pMarineViewer` window should open. However, you will not see any vehicles until you launch the backseat and frontseat.

5.3 Launching the hardware-in-the-loop simulation mission

Similar to the runtime launch, when you launch an HITL mission on the Raspberry Pi (i.e., the backseat), it will automatically trigger the launch of the frontseat on the PocketBeagle. To launch the mission, use the `./launch_hitl.sh` script. However, you must inform the backseat MOOS community of the IP address of the designated topside laptop using the `--topside=` flag. For example:

```
$ ./launch_hitl.sh --topside=10.42.0.1
```

Now, the two vehicle icons—`CAP` and `CAP_GT`—should appear on the `pMarineViewer` window, and they should begin executing the same `shipwreck_search` mission that we previously ran in SITL mode. However, this time, the mock-up control surfaces and thruster on your training kit should physically move, reflecting the actuator commands being issued during the simulation.

Observe the following behaviors during the HITL simulation:

- Movement of the rudder with heading changes.
- Movement of the elevators with heading changes.
- Movement of the elevators with depth (note that the mission expects to run the vehicle at a depth of 1 m).

Please do NOT time warp hardware-in-the-loop simulations since the servo motors and the thruster can be damaged.

5.4 Killing the mission on the embedded computers

Since the software is running in the background on both the Raspberry Pi and PocketBeagle, you cannot use `Ctrl+C` to stop it. Instead, you must use the `ktstack` command (short for 'kill-the-stack') to terminate the processes. It will terminate the software on both the Raspberry Pi and PocketBeagle. It is good practice to run `ktstack` before launching a new mission to ensure that there are no unwanted processes lingering in the background.

5.5 Killing the topside

Terminate the Topside MOOS community by using the `ktstack` command (short for "kill-the-stack") on the topside laptop.

5.6 Self checkoff assessment

- Are you able to run the hardware-in-the-loop simulated mission successfully?
- Do you see vehicle on your topside `pMarineViewer` window?
- Do you see the servo movements in your training-kit's mock-up tailcone?
- Are you able to terminate the mission?