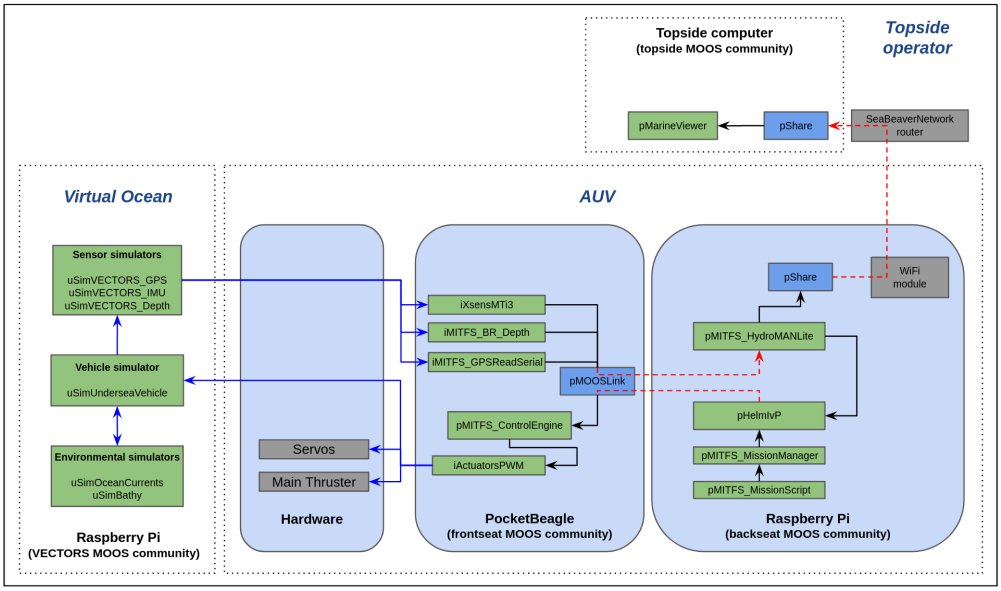
Lab 5 - Software and Hardware in the Loop Simulations

2.S01 Introduction to Autonomous Underwater Vehicles



Spring 2026

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

1 Objectives

- Setting up your topside computer: downloading and installing the latest versions of the StackUxV, StackUxV-drivers, VECTORS and missions-StackUxV software projects.
- Setting up the Raspberry Pi in your training-kit: updating the StackUxV, StackUxV-drivers, VECTORS and missions-StackUxV software projects.
- Setting up the PocketBeagle in your training-kit: updating the StackUxV, StackUxV-drivers, VECTORS and missions-StackUxV software projects.
- 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 downloading and installing this software on your topside laptop, it is helpful to have a basic understanding of what each project does:

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. **StackUxV**: `StackUxV` is a unified software stack for marine robotic systems. By parameterizing vehicle-specific functions, it enables rapid development of underwater, surface, ground, and aerial robots without requiring core software to be rewritten for each platform; most customization is done through configuration. `StackUxV` includes the core software subsystems needed for a functional robot, including mission planning (`pMITFS_MissionScript`), mission management (`pMITFS_MissionManager`), underwater navigation (`pMITFS_HydroMANLite`), and control (`pMITFS_ControlEngine`). It also integrates established software frameworks such as LAMSS, MOOS-IvP, Goby3, HydroMAN, and VECTORS for acoustic processing, autonomy behaviors, communication, navigation, and simulation. A key design principle of `StackUxV` is to establish a common behavioral and decision-making baseline across different vehicle types, enabling interoperability and mutual predictability. We will install `StackUxV` from a private Debian package distribution.

3. **VECTORS**: This software project provides our virtual ocean environment. Although it is not used during real vehicle operations in the water, it plays an important role in software-in-the-loop and hardware-in-the-loop simulation. In 2.S01, we will not install `VECTORS` from source code; instead, we will install it from a private Debian package distribution.

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

5. **missions-StackUxV**: StackUxV is a powerful but inherently complex framework, designed to support flexibility at the fleet, architecture, vehicle, and mission levels. To manage this complexity, we have developed a unified software configuration and launch utility called *missions-StackUxV*. This utility is capable of handling diverse software configurations across different vehicle types, while also supporting mission setup and software launch. It primarily consists of configuration files for various MOOS applications, together with shell scripts that enable seamless mission execution. Since this repository does not contain any C++ source code, no building is required. 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 for 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 separately. In that case, 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/

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 StackUxV, StackUxV-drivers and VECTORS

For the 2.S01 class, we distribute StackUxV, StackUxV-drivers and VECTORS software projects as Debian packages. These Debian packages are hosted in a private git repository called StackUxV-DEBIAN-PKG-ARCHIVE. Your topside computer and embedded computers within the training-kit are given permission to access StackUxV-DEBIAN-PKG-ARCHIVE via https://github.com/supun-randeni/StackUxV-DEBIAN-PKG-ARCHIVE.

We have already downloaded this repository onto the Raspberry Pi and PocketBeagle in your training kit, and have installed the software on those devices. That is why you were able to run unit tests and missions using your kit. Now it is time to install the same software on your topside laptop.

First let's clone the StackUxV-DEBIAN-PKG-ARCHIVE repository into your home directory:

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

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

```
$ git clone git@github.com:supun-randeni/StackUxV-DEBIAN-PKG-ARCHIVE.git
Cloning into 'StackUxV-DEBIAN-PKG-ARCHIVE'...
Warning: Permanently added the ECDSA host key for IP address '140.82.113.4' to the list of
known hosts.
remote: Enumerating objects: 131, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 131 (delta 2), reused 6 (delta 1), pack-reused 116 (from 1)
Receiving objects: 100% (131/131), 109.64 MiB | 2.59 MiB/s, done.
Resolving deltas: 100% (45/45), done.
```

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

```
$ git clone git@github.com:supun-randeni/StackUxV-DEBIAN-PKG-ARCHIVE.git
Cloning into 'StackUxV-DEBIAN-PKG-ARCHIVE'...
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-DEBIAN-PKG-ARCHIVE` in your home directory:

```
$ cd ~
$ ls
Desktop    Downloads  Music     Public                        temp       Videos
Documents  moos-ivp   Pictures  StackUxV-DEBIAN-PKG-ARCHIVE  Templates
```

Let's go to `StackUxV-DEBIAN-PKG-ARCHIVE` and explore:

```
$ cd ~/StackUxV-DEBIAN-PKG-ARCHIVE/
$ ls -F
DEBIAN-PKGS_StackUxV/          DEBIAN-PKGS_VECTORS/  README.md    uninstall_packages.sh*
DEBIAN-PKGS_StackUxV-drivers/  install_packages.sh*  SYSTEM_sb3/
```

It is helpful to know the content of this repository:

- `DEBIAN-PKGS_StackUxV`: This directory is the Debian package archive for the `StackUxV` project. It contains binary files compiled for several computer architectures and operating systems. It includes the latest release of `StackUxV` as well as some earlier releases. In addition, it also includes the software dependencies required by `StackUxV`. When you use the `install_packages.sh` script, these dependencies will be installed automatically before `StackUxV` itself is installed.
- `DEBIAN-PKGS_StackUxV-drivers`: This directory is the Debian package archive of `StackUxV-drivers` project.
- `DEBIAN-PKGS_VECTORS`: This directory is the Debian package archive of `VECTORS` Virtual Ocean

project.

- DEBIAN-PKGS_VECTORS: This directory is the Debian package archive of VECTORS Virtual Ocean project.
- install_packages.sh: This is an interactive script that allows you to easily install and update the software packages distributed through StackUxV-DEBIAN-PKG-ARCHIVE.
- uninstall_packages.sh: This is an interactive script that you can use to uninstall these packages.
- SYSTEM_sb3: This directory contains some of the system files required for the SeaBeaver III AUV.

Feel free to cd into these directories and explore them.

Now let's use the install_packages.sh script to install StackUxV, StackUxV-drivers and VECTORS software projects onto your topside laptop:

```
$ sudo ./install_packages.sh
```

You will be prompted to enter the password of your computer. After that, the script will interactively guide you through selecting which software packages and versions to install.

```
$ sudo ./install_packages.sh
[sudo] password for supun-toughbook:

Available Software Packages:
  1) StackUxV
  2) VECTORS
  3) StackUxV-drivers

Enter A to install all, or enter package numbers separated by spaces:
```

Since we need to install all three packages, type 'A' and press the return key.

```
Selected software package(s):
  StackUxV
  VECTORS
  StackUxV-drivers

Software package: StackUxV
Available versions:
  L) latest   [version: 2.2.0-1]
  1) 1.0.0-1

Enter L for latest, or one archived version number for StackUxV:
```

Please install the latest version of StackUxV; i.e., type 'L' and press the return key.

```
Software package: VECTORS
Available versions:
  L) latest   [version: 3.0.0-1]

Enter L for latest, or one archived version number for VECTORS: l

Software package: StackUxV-drivers
Available versions:
  L) latest   [version: 2.2.0-1]
  1) 1.1.0-1

Enter L for latest, or one archived version number for StackUxV-drivers: l
```

Please do the same for VECTORS and StackUxV-drivers projects as well.

You will then be shown which files are going to be installed and asked to confirm.

```
Detected your host system:
  OS:           Ubuntu
  Version:      20.04
  Codename:     focal
  Architecture: amd64

Packages selected for installation:
  stackuxv_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb
  vectors_3.0.0-1~focal_amd64__26-03-26__master.deb
  stackuxv-drivers_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb

Proceed with installation? [y/N]:
```

Please type y to confirm and press the return key. The installation will then begin. First, the script will install the dependencies for each project. It will then uninstall any previous installations of the project and install the selected versions. This process may take a few minutes. Watch for any error messages as the installation proceeds. If you encounter any errors, please contact the instructors. If the installation completes successfully, you should see something similar to the following:

```
Processing stackuxv_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb...
Running dependencies for StackUxV...
Installing distribution non-specific dependencies
Get:1 https://dl.google.com/linux/chrome/deb stable InRelease [1,825 B]
Get:2 https://dl.google.com/linux/chrome/deb stable/main amd64 Packages [1,213 B]

...

Done with DEPENDENCIES
stackuxv is not currently installed. Skipping uninstall step.
Installing stackuxv_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb...
Reading package lists... Done
Building dependency tree
Reading state information... Done
stackuxv_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb'
The following NEW packages will be installed:
  stackuxv
0 upgraded, 1 newly installed, 0 to remove and 65 not upgraded.
Need to get 0 B/8,390 kB of archives.

...

Installed stackuxv_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb successfully.

Processing vectors_3.0.0-1~focal_amd64__26-03-26__master.deb...
No dependencies directory found for VECTORS. Skipping dependency step.
vectors is not currently installed. Skipping uninstall step.
Installing vectors_3.0.0-1~focal_amd64__26-03-26__master.deb...

...

Installed vectors_3.0.0-1~focal_amd64__26-03-26__master.deb successfully.

Processing stackuxv-drivers_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb...
Running dependencies for StackUxV-drivers...
Installing librobotcontrol..
Selecting previously unselected package librobotcontrol.

...


Done with DEPENDENCIES
stackuxv-drivers is not currently installed. Skipping uninstall step.
Installing stackuxv-drivers_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb...

...

Installed stackuxv-drivers_2.2.0-1~focal_amd64__26-03-26__2026_2s01.deb successfully.

Installation process complete.
```

Let us check whether the installation was successful by verifying that the system can find the binaries we just installed.

```
$ which pMITFS_MissionManager iMITFS_GPSReadSerial uSimVECTORS_IMU
```

If you see something similar to this, it means your system can find these newly installed MOOS applications, and the installation was successful. Otherwise, please contact an instructor.

```
$ which pMITFS_MissionManager iMITFS_GPSReadSerial uSimVECTORS_IMU
/usr/local/bin/pMITFS_MissionManager
/usr/local/bin/iMITFS_GPSReadSerial
/usr/local/bin/uSimVECTORS_IMU
```

2.3 Adding to shell path

When StackUxV, StackUxV-drivers, and VECTORS are installed from their Debian packages, their binaries, compiled libraries, and header files are installed into /usr/local/bin, /usr/local/lib, and /usr/local/include, respectively. This is why the earlier which command found the executables in /usr/local/bin. In most cases, /usr/local/ is already included in your computer's shell path.

However, several additional environment variables still need to be configured. For example, we must tell the dynamic linker where to find the StackUxV and StackUxV-drivers libraries by adding their locations to LD_LIBRARY_PATH. To make this easier, we have provided a text file containing the required settings: StackUxV-DEBIAN-PKG-ARCHIVE/SYSTEM_sb3/dot_bash_seabeaver. All you need to do is source this file from your .bashrc, and dot_bash_seabeaver will handle the rest (*feel free to open the file and try to understand what it is doing*).

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

```
if [ -f ~/StackUxV-DEBIAN-PKG-ARCHIVE/SYSTEM_sb3/dot_bash_seabeaver ]; then
    . ~/StackUxV-DEBIAN-PKG-ARCHIVE/SYSTEM_sb3/dot_bash_seabeaver
fi
```

Don't forget to source the .bashrc file.

2.4 Downloading missions-StackUxV

StackUxV is a unified software stack for marine robotic systems. By parameterizing vehicle-specific functions, it enables rapid development of underwater, surface, ground, and aerial robots without the need to rewrite core software for each platform; instead, most customization is handled through configuration. StackUxV provides the core software subsystems required for a functional robot, including sensing, control, mission management, and hardware drivers, while integrating established frameworks such as LAMSS, MOOS-IvP, Goby3, HydroMAN, and VECTORS for acoustic processing, autonomy behaviors, communication, navigation, and simulation. A key design principle of StackUxV is to establish a common behavioral and decision-making baseline across different vehicle types, enabling interoperability and mutual predictability.

This led to a powerful but inherently complex framework that supports flexibility at the fleet, architecture, vehicle, and mission levels. To manage this complexity, we developed a unified software

configuration and launch utility called *missions-StackUxV*. This utility is designed to handle diverse configurations across different vehicle types, while also supporting software setup and mission launching.

For 2.S01, 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          moos-ivp  Pictures  StackUxV-DEBIAN-PKG-ARCHIVE  Templates
Documents  missions-StackUxV  Music     Public    temp                         Videos
```

Let's navigate to the `missions-StackUxV` directory and explore its contents. This should look familiar to you since we used it during Lab 4 (Surface navigation using GPS):

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

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.

`missions-StackUxV` is actively used by several MIT and external programs. To avoid interfering with those projects, we've created a dedicated branch called `2026_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 `2026_2s01` branch by using the command `git checkout 2026_2s01`. If you successfully switched to the branch, you will see an output similar to this:

```
$ git checkout 2026_2s01
Switched to branch ‘2026_2s01’
Your branch is up to date with ‘origin/2026_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 Self checkoff assessment

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

   ```
   $ which pMITFS_ControlEngine pMITFS_HydroMANLite pMITFS_MissionManager
   ```

   It should output something similar to this:

   ```
   $ which pMITFS_ControlEngine pMITFS_HydroMANLite pMITFS_MissionManager

   /usr/local/bin/pMITFS_ControlEngine
   /usr/local/bin/pMITFS_HydroMANLite
   /usr/local/bin/pMITFS_MissionManager
   ```

2. Ensure that `StackUxV-drivers` is installed 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

   /usr/local/bin/calibrate_xsens_mti3
   /usr/local/bin/test_br_thruster
   /usr/local/bin/iActuatorsPWM
   /usr/local/bin/iXsensMTi3
   /usr/local/bin/mitfs_safety_daemon
   ```

3. Ensure that `VECTORS` is installed correctly and that its binaries are included in your shell path by running the following command:

```
$ which uSimVECTORS_IMU  iVECTORS_Gateway  uVECTORSUnderseaVehicle
```

It should output something similar to this:

```
$ which uSimVECTORS_IMU  iVECTORS_Gateway  uVECTORSUnderseaVehicle

/usr/local/bin/uSimVECTORS_IMU
/usr/local/bin/iVECTORS_Gateway
/usr/local/bin/uVECTORSUnderseaVehicle
```

4. Ensure that you are sourcing StackUxV-DEBIAN-PKG-ARCHIVE/SYSTEM_sb3/dot_bash_seabeaver correctly from your .bashrc, and that the Bash configurations defined in dot_bash_seabeaver are working properly, by running the following command:

```
$ which ktstack  update_missions_stackuxv turn_off_seabeaver
```

It should output something similar to this:

```
$ which ktstack  update_missions_stackuxv turn_off_seabeaver

/home/supun-toughbook/missions-StackUxV/build_scripts/ktstack
/home/supun-toughbook/missions-StackUxV/build_scripts/update_missions_stackuxv
/home/supun-toughbook/missions-StackUxV/build_scripts/turn_off_seabeaver
```

2.6 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.

Use the ktstack command on both the topside laptop and RasPi to kill the mission.

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 1), the frontseat, backseat, topside and VECTORS MOOS communities are all launched on the topside laptop computer.

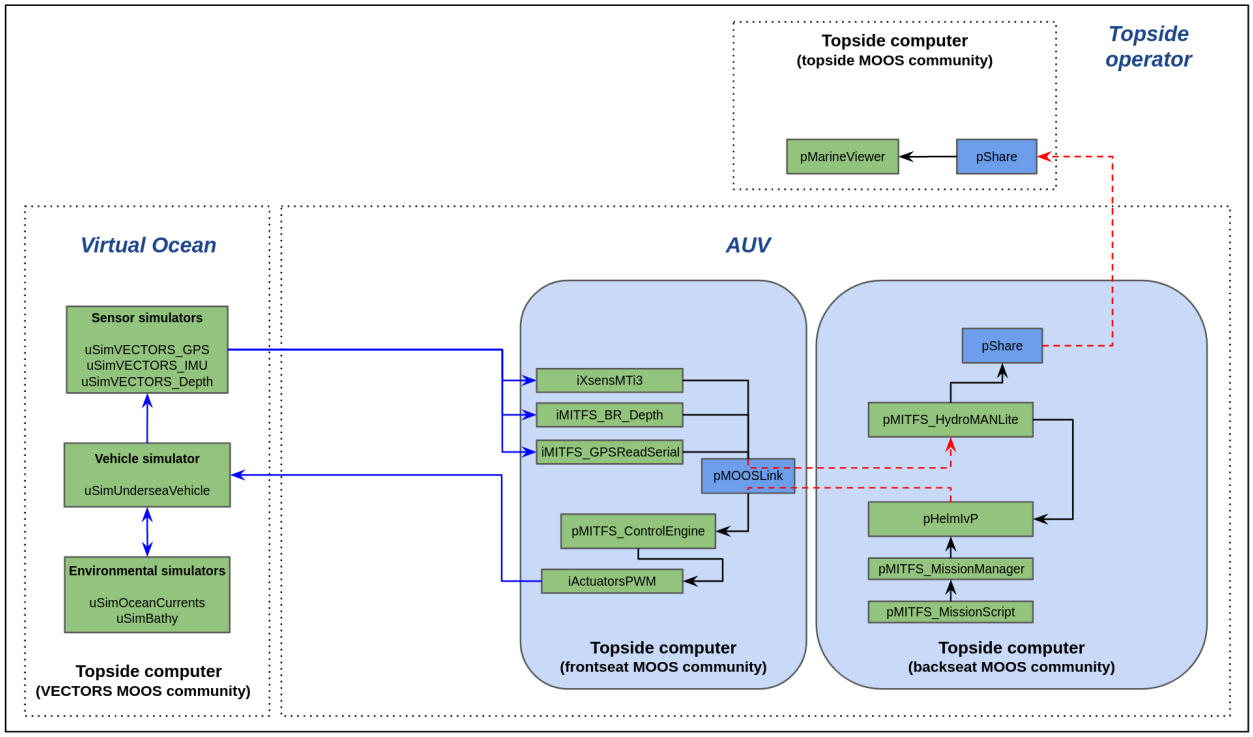


Figure 1: In software-in-the-loop simulations, frontseat, backseat, topside and VECTORS MOOS communities are all launched on the topside computer

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 an SITL simulation in supun-toughbook..
Terminating any previous MOOS apps running in the background..
Killing MOOS, Goby and ROS processes in supun-toughbook..
sending self-exit instruction to 224.1.1.3:4000
moos was not running, doing nothing.
moos was not running, doing nothing.
Terminating any previous MOOS apps running in the background COMPLETE!

Cleaning previous targ files..
Cleaning previous targ files COMPLETE!
Cleaning previous temp files..
Cleaning previous temp files COMPLETE!
A fleet is not selected. A solo mission will be launched with the following configuration:
Architecture is set to:
   seabeaver_iii
Cruise is set to:
   shipwreck_search
Vehicle is set to:
   cap

Assembling CAP_Backseat..
There are 11 apps assembled in the CAP_Backseat community.
Assembling CAP_Backseat COMPLETE!
Assembling CAP_Frontseat..
There are 7 apps assembled in the CAP_Frontseat community.
Assembling CAP_Frontseat COMPLETE!
Assembling CAP_HydroMAN..
No apps are configured to run in CAP_HydroMAN community. Hence it will not be launched.
Assembling CAP_HydroMAN COMPLETE!
Assembling Topside..
There are 6 apps assembled in the Topside community.
Assembling Topside COMPLETE!
Assembling VirtualOcean..
There are 10 apps assembled in the VirtualOcean community.
Assembling VirtualOcean COMPLETE!
Launching targ_CAP_Backseat
Launching targ_CAP_Backseat COMPLETE!
Launching targ_CAP_Frontseat
Launching targ_CAP_Frontseat COMPLETE!
Launching targ_Topside
Launching targ_Topside COMPLETE!
Launching targ_VirtualOcean
Launching targ_VirtualOcean COMPLETE!
Launching * ROS domain
Launching * COMPLETE!
Launching an SITL simulation in supun-toughbook 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.

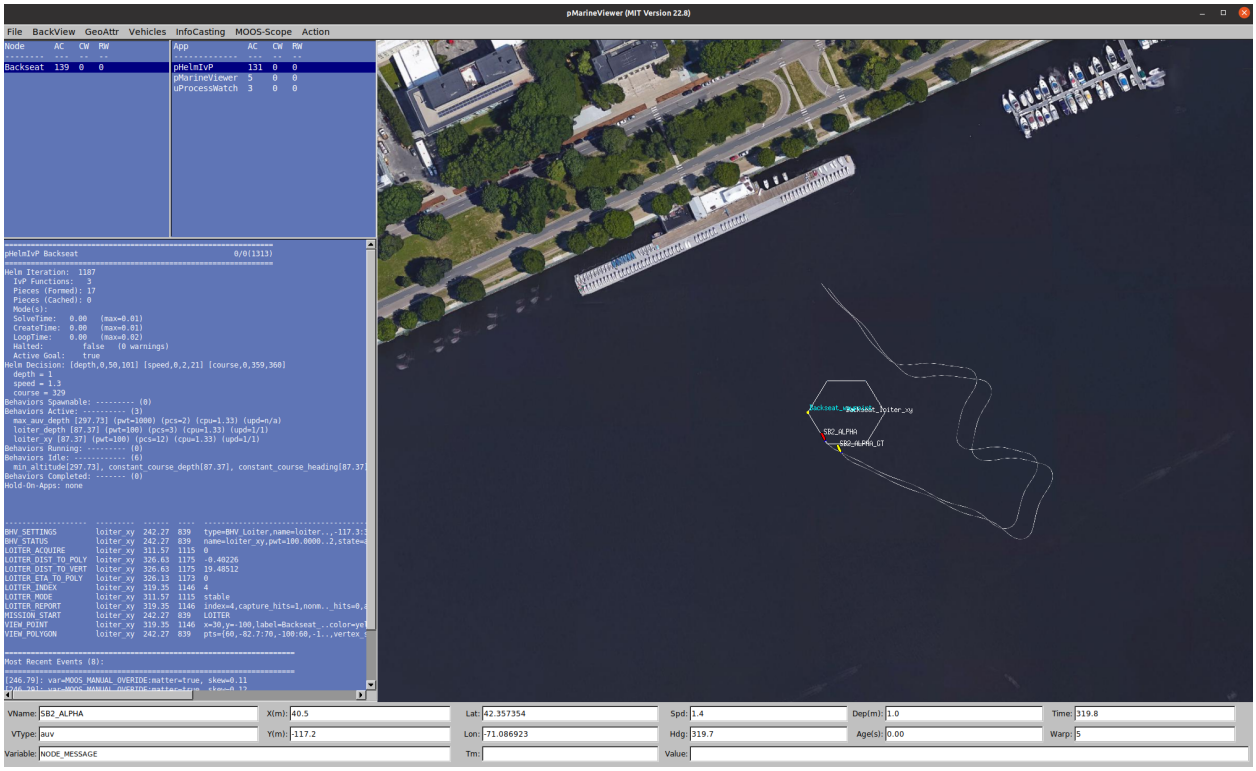


Figure 2: `pMarineViewer` window for the `shipwreck_search` mission

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 and directories inside:

- cruise.def: contains higher-level mission-related parameters such as mission start and end times.
- cruise_plugs: a "drag-and-drop" directory structure used to define additional MOOS applications associated with this cruise in each MOOS community. The cruise_plugs directory contains five subdirectories, each with configuration files for those additional applications.
  - cruise_plugs/backseat_plugs: additional mission-specific applications to run in the Backseat community.

- cruise_plugs/frontseat_plugs: additional mission-specific applications to run in the Frontseat community.
- cruise_plugs/hydroman_plugs: additional mission-specific applications to run on the HydroMAN community.
- cruise_plugs/topside_plugs: additional mission-specific applications to run on the Topside community.
- cruise_plugs/virtual_ocean_plugs: additional mission-specific applications to run on the Virtual Ocean community.

One important mission-specific application is pMITFS_MissionScript, which contains the mission plan. It is intended to run in the Backseat community; therefore, its configuration file is located at shipwreck_search/cruise_plugs/backseat_plugs/pMITFS_MissionScript.plug.

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 and missions-StackUxV downloaded and built from source, as well as StackUxV, StackUxV-drivers, and VECTORS installed from the StackUxV-DEBIAN-PKG-ARCHIVE repository (*for your reference, the same steps outlined in Section 2 were followed to install them*). However, because the instructors continuously update the code throughout the semester, it is good practice to update StackUxV, StackUxV-drivers, VECTORS, and missions-StackUxV on both the Raspberry Pi and PocketBeagle before the start of each lab.

4.1 Updating StackUxV, StackUxV-drivers and VECTORS on embedded computers

To update StackUxV, StackUxV-drivers, and VECTORS, first cd into the StackUxV-DEBIAN-PKG-ARCHIVE directory on the Raspberry Pi:

```
$ 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 PocketBeagle.

4.2 Updating `missions-StackUxV` on 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
```

4.3 Self checkoff assessment

1. Are you able to update and reinstall `StackUxV`, `StackUxV-drivers`, and `VECTORS` using `StackUxV-DEBIAN-PKG-ARC` on both the Raspberry Pi and the PocketBeagle?
2. Are you able to update `missions-StackUxV` on both the Raspberry Pi and the 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).

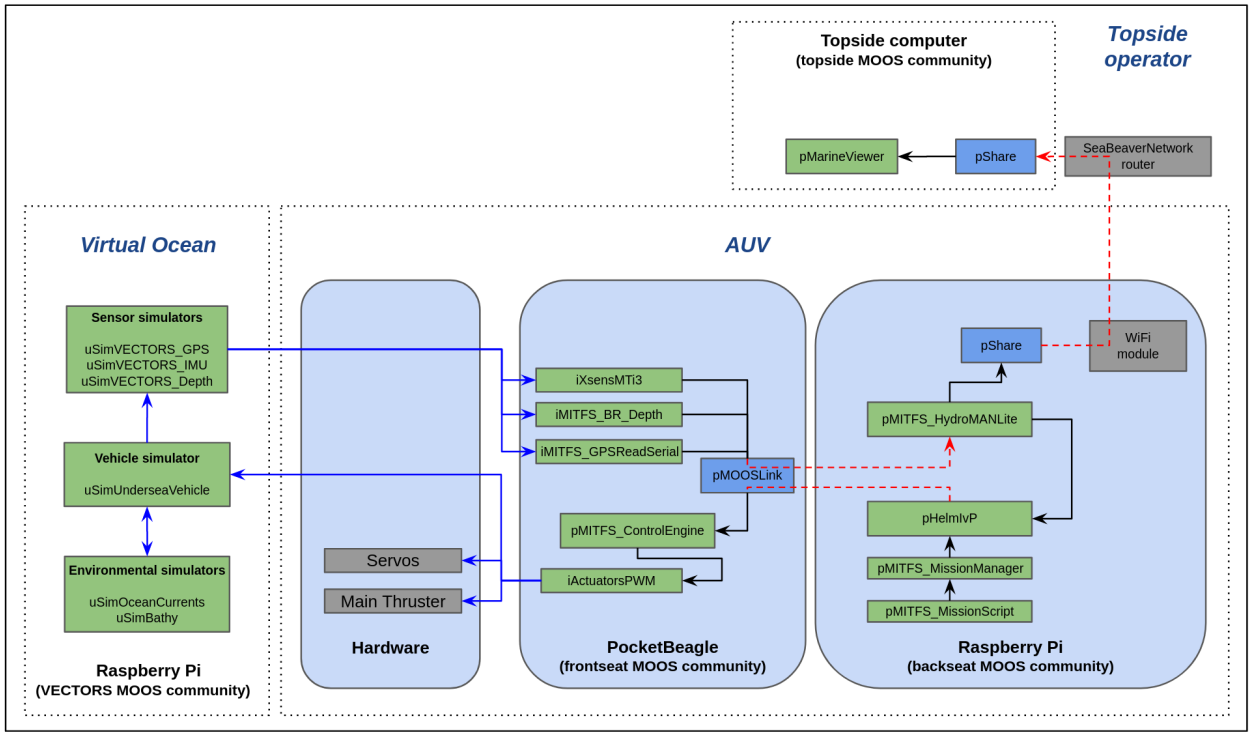


Figure 3: Four MOOS communities are launched during hardware-in-the-loop simulations

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?