

Lab 5 - Simulation of Multi-Vehicle Deployments



Fall, 2020

Michael Benjamin, mikerb@mit.edu
Dept of Mechanical Engineering, MIT, Cambridge MA 02139
Michael Novitzky, michael.novitzky@westpoint.edu
United States Military Academy, West Point NY

1	Overview and Objectives	3
2	The Shoreside (Topside) / Vehicle Topology	4
3	Experimenting with pShare	4
3.1	Exercise 1 - The Alpha pShare Mission	4
3.2	Exercise 2 - The Alpha Bravo pShare Mission	6
4	Using the uField Toolbox to Facilitate Multi-Vehicle Simulations	8
4.1	Exercise 3 - The Henry Gilda Baseline Mission	8
4.2	Exercise 4 - The Henry Gilda Refuel Mission	10
4.3	Exercise 5 - The Henry Gilda Auto Refuel Mission	11
5	Where to Find the Solutions	13
5.1	Solutions to Exercise: Alpha pShare	13
5.2	Solutions to Exercise: Alpha Bravo pShare	13
5.3	Solutions to Exercise: Henry Gilda Baseline	13
5.4	Solutions to Exercise: Henry Gilda Refuel	13
5.5	Solutions to Exercise: Henry Gilda Auto Refuel	13

1 Overview and Objectives

In this lab we shift focus to autonomy configurations involving multiple vehicles. Ultimately the inter-MOOSDB or inter-vehicle communication may come over a acoustic modem link or a satellite link, our primary initial focus is on communications over an Internet connection, even if the multiple "nodes" are all running on your one laptop.

Gaining familiarity with this mode of operation will be essential for later labs and operation of vehicles on the water.

- The Shoreside and Vehicle(s) Topology
- Converting the Alpha Mission to use a Shoreside / Vehicle Topology
- Converting the Alpha Mission to a Two-Vehicle Mission with pShare
- Using the uField Toolbox to Ease pShare Configuration

Make Sure Key Executables are Built and In Your Path

This lab does assume that you have a working MOOS-IvP tree checked out and installed on your computer. To verify this make sure that the following executables are built and findable in your shell path:

```
$ which MOOSDB
/Users/you/moos-ivp/bin/MOOSDB
$ which pHelmIvP
/Users/you/moos-ivp/bin/pHelmIvP
```

Documentation Conventions

To help distinguish between MOOS variables, MOOS configuration parameters, and behavior configuration parameters, we will use the following conventions:

- MOOS variables are rendered in **green**, such as **IVPHELM_STATE**, as well as postings to the **MOOSDB**, such as **DEPLOY=true**.
- MOOS configuration parameters are rendered in **blue**, such as **AppTick=10** and **verbose=true**.
- Behavior parameters are rendered in **brown**, such as **priority=100** and **endflag=RETURN=true**.

More MOOS / MOOS-IvP Resources

A few further resources are worth mentioning for following up this lab with your own exploration.

- See the slides from the accompanying lecture which give a bit more background into marine autonomy and the IvP Helm: http://oceanai.mit.edu/pavlab/pdfs_txt/lecture_05.pdf
- The IvP Helm and Utilities documentation: <http://oceanai.mit.edu/ivpman>
- The moos-ivp.org website: <http://www.moos-ivp.org>

2 The Shoreside (Topside) / Vehicle Topology

The layout of interconnected MOOS communities used in this lab is depicted in the figure below. This layout will be used for the remainder of the workshop.

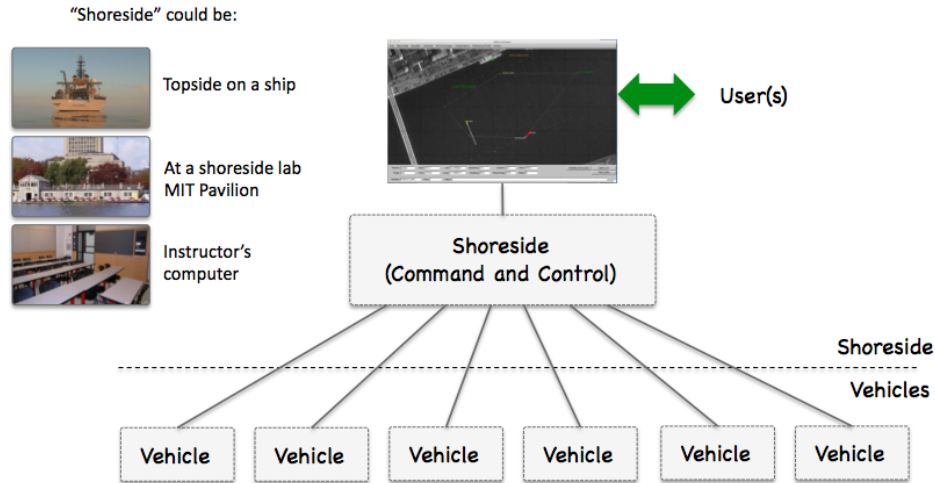


Figure 1: **Shoreside to Multi-Vehicle Topology:** A number of vehicles are deployed with each vehicle maintaining some level of connectivity to a shoreside command and control computer. Each node (vehicles and the shoreside) are comprised of a dedicated MOOS community. Modes and limits of communication may vary.

3 Experimenting with pShare

In the first exercise in today's lab, the goal is to become familiar with [pShare](#). In these first couple of missions, we will be configuring [pShare](#) explicitly in our .moos files. In later missions, we will be using components of the uField Toolbox to automatically configure the share configurations.

3.1 Exercise 1 - The Alpha pShare Mission

In this part we will:

- Prepare a copy of the alpha mission for experimenting.
- Create two .moos files to launch two [MOOSDB](#) processes.
- Implement the Alpha mission with two MOOS communities using [pShare](#).
- Launch the two communities and confirm that sharing works.

3.1.1 Make a copy of the alpha mission

The first step is to copy the alpha example mission from the moos-ivp class tree into your own directory.

```
$ cp -rp moos-ivp/ivp/missions/s1_alpha s15_alpha_pshare
```

3.1.2 Split the alpha mission into two separate MOOS communities

In this step you will create two separate MOOS communities: a *shoreside* community and an *alpha* vehicle community, by creating two separate .moos files. In the shoreside community there will be a `MOOSDB` and `pMarineViewer`. In the alpha community will be a `MOOSDB` and everything but `pMarineViewer`. In both communities you will also need to add a `pShare` configuration block and add `pShare` to the `Antler` configuration block.

The primary challenge here is to consider which variables to configure for sharing in each direction. One hint is that, from the vehicle to the shoreside you will need to share the `NODE_REPORT`. This is the message generated from `pNodeReporter` containing much of the vehicle state, and used by `pMarineViewer` to render the vehicle. It is generated locally on the vehicle as `NODE_REPORT_LOCAL` and should arrive in the shoreside as `NODE_REPORT`. The configuration in alpha's `pShare` config block will look something like:

```
output = src_name=NODE_REPORT_LOCAL, dest_name=NODE_REPORT, route=localhost:9200
```

You will also want to share the `VIEW_SEGLIST` and `VIEW_POINT` variables to enable `pMarineViewer` on the shoreside to have the visual feedback of the vehicle waypoints. What variables do you need to share from the shoreside to the vehicle?

3.1.3 Launch the two communities and confirm that sharing works

Using perhaps two separate terminal windows, launch both MOOS communities and confirm that the modified Alpha mission works as before. You still should be able to deploy and return the vehicle with the buttons in `pMarineViewer`. Consider what is being poked when hitting those buttons. (You can always find this out by looking at the `pMarineViewer` configuration block in the mission file). Make sure those variables are being properly shared to the alpha vehicle community.

If things are not working, consider the following trouble-shooting points:

- Make sure the port specified in the share input route on each side matches the destination port on the other, and vice versa.
- Make sure the port numbers used are not the same as the `MOOSDB` port numbers.
- Try just poking with `uPokeDB` to test the sharing. For example, poking `DEPLOY=true` in the shoreside community, and opening a scope in the destination (alpha) community.
- You should see a vehicle in the `pMarineViewer` window even before deploying the vehicle. If you don't check the `pShare` configuration on the vehicle side.

The result should look just like the basic `s1.alpha` mission, but, under the hood, things are just running in two separate MOOS communities:

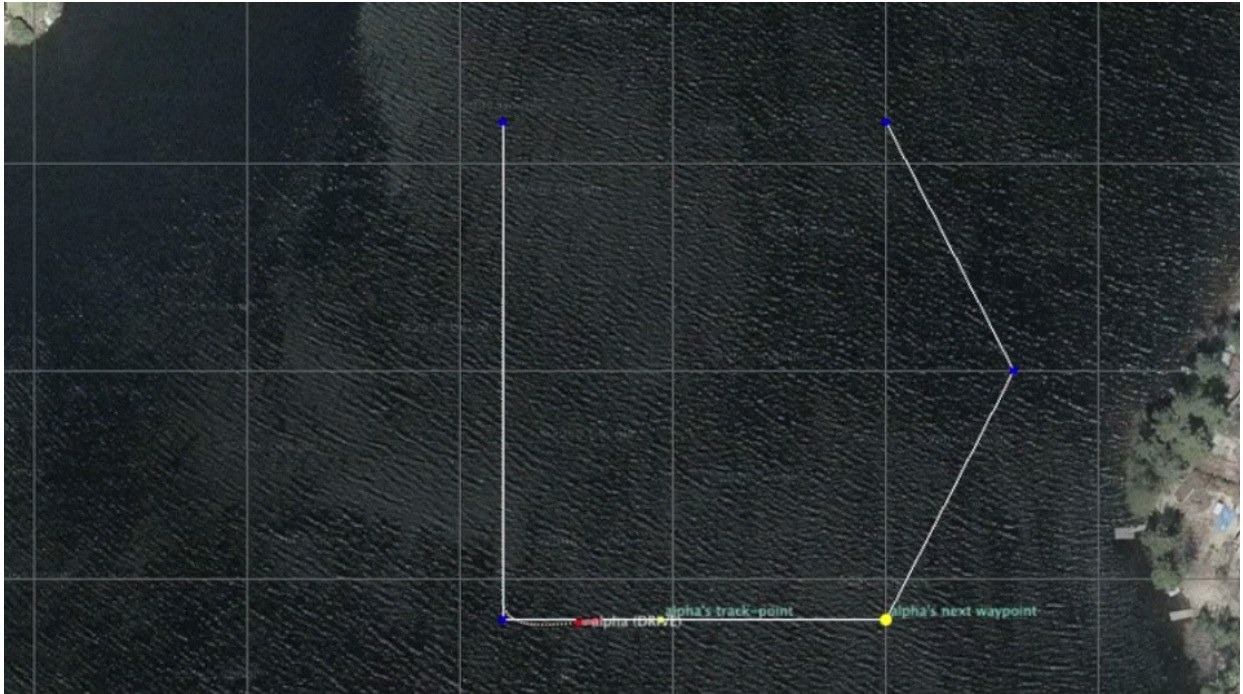


Figure 2: The Alpha mission.

video:(0:19): <https://vimeo.com/84549446>

3.2 Exercise 2 - The Alpha Bravo pShare Mission

In this part we will:

- Prepare a copy of the previous modified alpha mission for experimenting.
- Create a third .moos file, bravo.moos, to launch another simulated vehicle.
- Launch the three communities and confirm that sharing works, and deploy and return commands work for both vehicles with a single **pMarineViewer** button click.

3.2.1 Make a Copy of the Previous Two-MOOS-Community Alpha mission

The first step is to copy the alpha example mission from the previous exercise in Section 3.1. The file structure should be, calling it `s16_alpha_bravo_pshare`.

3.2.2 Create a New bravo.moos File for Simulating a Second Vehicle

In this step you will create a third mission file, `bravo.moos`, for simulating a second vehicle. In the `bravo.moos` file, you will need to configure it with a distinct community name, e.g., `bravo`, distinct port number, and distinct port number for `UDPListen` in the `pShare` configuration block.

A `bravo.bhv` file will also need to be created for this vehicle. The behavior mission is not the point of focus here, so just create a waypoint survey mission similar to alpha's with the vertices shifted 50 meters to the east, and 20 meters to the south. Shift the bravo vehicle's starting position

50 meters to the east (in the `uSimMarine` configuration block). The `shoreside.moos` will also need to be altered to share the `DEPLOY` and `RETURN` commands out to both vehicles with a single button click.

3.2.3 Launch the Three Communities and Confirm Things Work

Your final mission configuration should meet the following criteria:

1. You should be able to launch both vehicles and the shoreside community with a single shell script.
2. You should be able to deploy and return both vehicles with a single button click in `pMarineViewer`.
3. The vehicles and the mission waypoints for both vehicles should be viewable in `pMarineViewer`.
4. Your `pMarineViewer` should also be configured to deploy or return a single chosen vehicle in isolation. Hint: use the `actions` parameter in `pMarineViewer` to add `deploy-alpha`, `return-bravo` etc. capability in the Action pull-down menu.

It should look something like the video posted at:

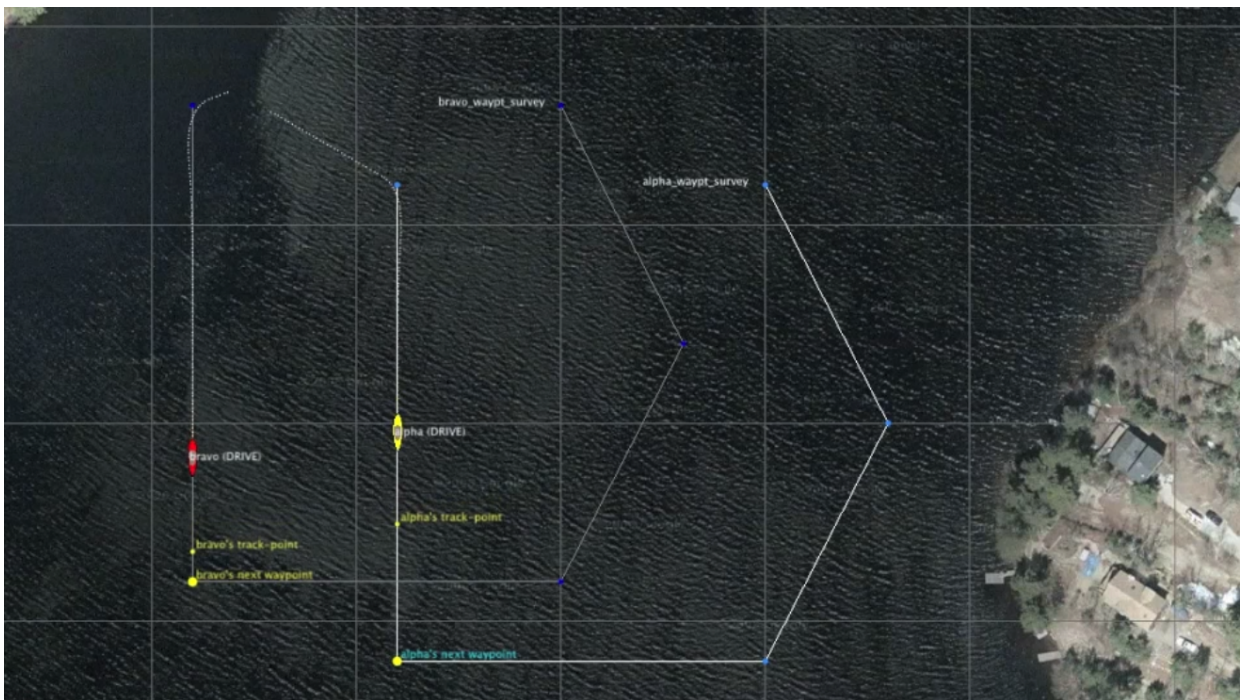


Figure 3: A simple two-vehicle mission connecting to vehicle communities, and a shoreside community, using pShare. video:(0:21): <https://vimeo.com/87900172>

4 Using the uField Toolbox to Facilitate Multi-Vehicle Simulations

In the next exercise, the goal is to create an autonomy mission that uses a few modules in the uField Toolbox to replace some of the `pShare` configuration steps used in the previous exercise. Our end goal is a two vehicle simulation that should be easily scalable to a larger number of vehicles.

Once we have the multi-vehicle simulation established, our goal in the next section will be to build a mission where two simulated vehicles are receiving points in the x-y plane, from the shoreside community, and traversing those points in a shortest-path trajectory.

4.1 Exercise 3 - The Henry Gilda Baseline Mission

In this part we will:

- Copy the baseline mission from the `moos-ivp/missions-2680` folder.
- Note the structure of the launch script and the `nsplug` setup.
- Note the roles of `uFldShoreBroker`, `uFldNodeBroker`, and `pHostInfo`.

Copy the baseline mission from the moos-ivp tree

Start by copying a baseline version of the mission from the `moos-ivp MIT 2.680` mission tree:

```
$ cp -rp moos-ivp/ivp/missions-2680/lab_08_henry_gilda_baseline s_17_henry_gilda_baseline
```

If you don't see the `lab_07_henry_gilda_baseline` folder, or even the `missions-2680`, try doing `svn update`. Confirm that the mission launches properly by typing `./launch.sh 10` from the command line in your newly created folder. You should see two vehicles appear on the screen. Deploy them with the `DEPLOY` button, and return them with the `RETURN` button. At any time you can station-keep them by hitting the `STATION` button. The vehicle should automatically enter the station keeping mode upon returning.

It should look something like the video posted at:



Figure 4: A simple two-vehicle mission connecting two loitering vehicle communities, and a shoreside community, using the uField Toolbox utilities for coordinating the pShare connections.

video:(0:20): <https://vimeo.com/87907093>

Understand the Launch Structure

Before moving on, take a look at how things are launched. See if you can understand what is going on inside the launch script. Note that the script is building the target `.moos` files by invoking an application called `nsplug`. This tool is a bit like the `cpp` pre-processor. It takes as an argument a file which can be thought of as a template of sorts, and produces another file with components of the template filled in. We use it so we can have just *one* mission and behavior file for both vehicles, with just a few of the details such as vehicle name, start position and MOOS community values filled in at launch time. You can learn a bit more about `nsplug` by:

```
$ nsplug -h
$ nsplug -m | less
```

Note that the `targ_*` files are generated automatically each time the launch script is invoked. If you edit these files, the changes will be lost the next time you launch!

Note that the launch script defines certain variables such as the vehicle name, `MOOSDB` port etc., and passes this info into `nsplug` for expansion.

Understand the uField Toolbox Operations

Before moving on, take a look at the relationship between `pHostInfo`, `uFldNodeBroker` and `uFldShoreBroker`. Note how they handle the `pShare` configuration for you on both ends. From this point forward,

using these tools, your configuration of share variables should be handled in this way. Please read the sections on these three applications in the uField Toolbox documentation on the course website.

For now, beginning with this baseline mission, share configuration will just work. But you *will* want to augment what is being shared for later steps in this lab, so try to understand how share configuration is handled in the `uFldNodeBroker` and `uFldShoreBroker` modules.

4.2 Exercise 4 - The Henry Gilda Refuel Mission

Copy your "Henry Gilda Baseline" mission from the previous exercise to create a new mission folder:

```
$ cp -rp s17_henry_gilda_baseline s18_henry_gilda_refuel
```

Augment this mission to accept a shoreside "refuel" command from a new `pMarineViewer` button. When the refuel command is given, the vehicle returns to its launch point, but automatically enters a new "refueling" mission mode while station keeping, for 60 seconds, and then automatically resumes loitering.

To accomplish this mission you will need to:

- Add a new refuel button to the `pMarineViewer` configuration that accepts a refueling command by posting `REFUEL_NEEDED_ALL=true`.
- Make sure this variable is shared out to all vehicles by augmenting the `uFldShoreBroker` configuration block in the `meta_shoreside.moos` file.
- (Remember - don't edit the `targ_*` files since these are auto-generated in the launch process and will be overwritten each time the mission is launched!)
- Add a new Timer behavior in the vehicle behavior configuration file that begins each time the vehicle returns for refueling, waits 60 seconds, and then posts endflags that result in the vehicle resuming its loiter missions. The vehicle should *automatically* go back to loitering after the timer ends.
- Station keeping should still work at any time, if commanded by the user.
- The vehicle may still be returned with the `RETURN` button, but when returned in this way, it acts as it did in the baseline mission - it station keeps and remains station keeping indefinitely until re-deployed.

It should look something like the video posted at:



Figure 5: The Henry Gilda Refuel mission has two loitering vehicles. When the user commands the vehicles to return, they return to their starting point and station keep for 60 seconds and automatically re-deploy afterwards. As with the baseline mission, both vehicles have their own MOOS community, connected to a shoreside community, with the connections coordinated using the uField Toolbox utilities.

video:(0:40): <https://vimeo.com/87950212>

4.3 Exercise 5 - The Henry Gilda Auto Refuel Mission

Copy your "Henry Gilda Refuel" mission from the previous assignment to create a new mission folder:

```
$ cp -rp s18_henry_gilda_refuel s19_henry_gilda_auto_refuel
```

Augment this mission such that (a) the vehicles automatically initiate the refueling after a fixed time "no-refuel-needed" interval (use 300 seconds for testing), (b) the so-called "no-refuel-needed" time duration is paused whenever the vehicle is in the station keeping mode. Presumably because no fuel is being expended. (c) the hierarchical mode structure has an explicit "refueling" mode while refueling.

To accomplish this mission you will need to:

- Add a new Timer behavior in vehicle behavior configuration file that begins when the mission begins and counts down until re-fueling is needed. It should post end-flags that trigger a mode change and the process of returning for re-fueling. It should be re-set after re-fueling is complete (it will need to have the `perpetual` parameter set to true).
- As before, station keeping should still work at any time, if commanded by the user. The

- need-to-refuel timer should also be paused when or if the vehicle is station-keeping.
- (Remember - don't edit the `targ_*` files since these are auto-generated in the launch process and will be overwritten each time the mission is launched!)

It should look something like the video posted at:



Figure 6: The Henry Gilda Refuel mission has two loitering vehicles. When the user commands the vehicles to return, they return to their starting point and station keep for 60 seconds and automatically re-deploy afterwards. As with the baseline mission, both vehicles have their own MOOS community, connected to a shoreside community, with the connections coordinated using the uField Toolbox utilities.

video:(0:47): <https://vimeo.com/87958055>

5 Where to Find the Solutions

All solutions can be downloaded from the MIT OceanAI server as below. Two utilities are needed for retrieving solution tar files. The `wget` and `tar` commands.

The `wget` utility is a readily available in package in both GNU/Linux and MacOS:

```
$ sudo apt-get install wget (GNU/Linux)
$ sudo port install wget (MacOS)
```

The `tar` utility is very likely native to your machine. To `untar` a tar file into a folder:

```
$ tar xvf example_folder.tar
$ ls
$ example_folder.tar example_folder/
```

5.1 Solutions to Exercise: Alpha pShare

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc/s15_alpha_pshare.tar
```

5.2 Solutions to Exercise: Alpha Bravo pShare

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc/s16_alpha_bravo_pshare.tar
```

5.3 Solutions to Exercise: Henry Gilda Baseline

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc/s17_henry_gilda_baseline.tar
```

5.4 Solutions to Exercise: Henry Gilda Refuel

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc/s18_henry_gilda_refuel.tar
```

5.5 Solutions to Exercise: Henry Gilda Auto Refuel

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc/s19_henry_gilda_auto_refuel.tar
```