Lab 05 - Introduction to Constrained Inter-Vehicle Messaging

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

The focus of this lab is an introduction to inter-vehicle messaging, focusing first on messaging in simulation. In this lab, two MOOS apps, used for the first time in our labs, will be our focus:

- uFldMessageHandler - A MOOS handler for incoming message from other vehicles.
- uFldNodeComms - A MOOS shoreside arbiter of inter-vehicle messages.

A summary of today's topics:

- Introduction to the uField Toolbox Inter-Vehicle Messaging Apps
- Implement basic messaging in a two-vehicle example
- Range-limited inter-vehicle messaging
- Recovering from an messaging out-of-range situation

2 Preliminaries

Make Sure You Have the Latest Updates

Always make sure you have the latest code:

```
$ cd moos-ivp
$ svn update
```

And rebuild if necessary:

```
$ ./build-moos.sh
$ ./build-ivp.sh
```

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

- The slides from Messaging lecture:
  http://oceanai.mit.edu/tx/lecture_06.pdf
- The `uFldMessageHandler` documentation
  http://oceanai.mit.edu/ivpman/apps/uFldMessageHandler
- The `uFldNodeComms` documentation
  http://oceanai.mit.edu/ivpman/apps/uFldNodeComms

Inter-Vehicle Messaging

The exercises in this lab involve inter-vehicle messaging. The goal will be to construct a two vehicle mission where each vehicle is loitering in a pattern on the west and east side of an operation area respectively. Each vehicle will periodically send the other vehicle a message containing a new latitude (Y value in local coordinates) to shift its loiter pattern.

The primary new modules being used in this lab are the `uFldMessageHandler` and `uFldNodeComms` modules. See the lecture slides for a description, and see the MOOS-IvP Tools documentation available on the http://oceanai.mit.edu/ivpman website for full documentation. The figure below illustrates the basic setup. The focus in this lab is on these two modules.

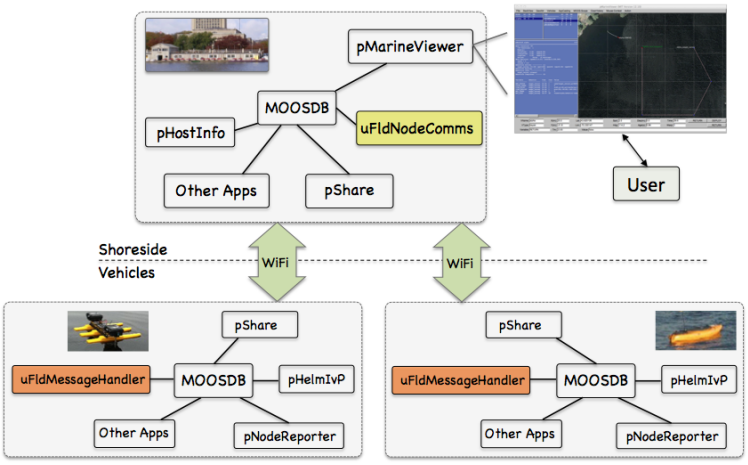

Figure 1: **The uFldMessageHandler and uFldNodeComms Modules:** The `uFldMessageHandler` runs on each vehicle and parses incoming `NODE_MESSAGE` postings. The `uFldNodeComms` module runs on the shoreside and routes messages to their destination vehicle(s).

3 Exercises

3.1 Exercise 1 - The Charlie Dana Baseline Mission

The first step is to get and run the baseline mission we will be starting from in this lab. Get a copy and put it in your local folder.

```
$ cd <your lab exercice folder>
$ cp -rp moos-ivp/ivp/missions-2680/lab_09_charlie_dana_baseline s20_charlie_dana_baseline
```

This mission is configured for two vehicles, `charlie` and `dana`, each doing a simple loiter mission with the ability to return and station-keep at any time. It should contain nothing you have not seen before in prior labs. It should be launchable with

```
$ cd s20_charlie_dana_baseline
$ ./launch.sh 15
```

It is our starting point for this lab, and should look something like the video posted at:

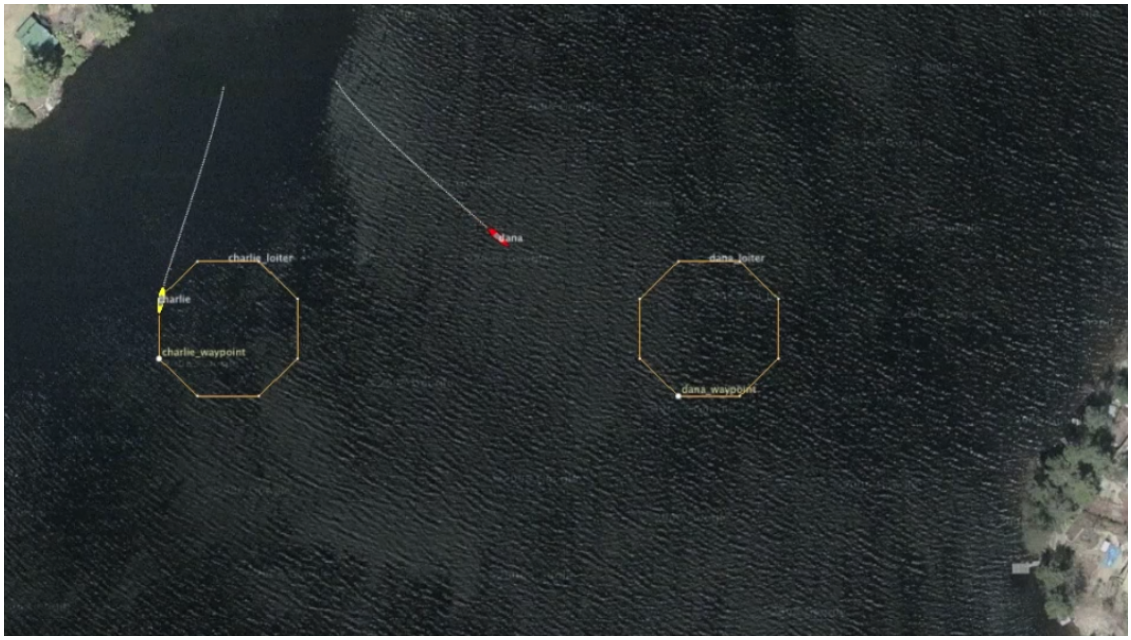

Figure 2: A simple two-vehicle baseline mission with both vehicles loitering at a fixed location.

video:(0:18): https://vimeo.com/88522655

3.2 Exercise 2 - The Charlie Dana NodeComms Mission

The next step is to augment your baseline mission to support inter-vehicle communications. The first key addition is the module uFldNodeComms which runs on the shoreside and accepts node reports from all connected vehicles. To get started, make a copy of your baseline mission into a folder called s21_charlie_dana_nodecomms.

```
$ cp -rp s20_charlie_dana_baseline  s21_charlie_dana_nodecomms
```

The first step is to add a uFldNodeComms to your pAntler configuration block in meta_shoreside.moos:

```
ProcessConfig = ANTLER
{
  MSBetweenLaunches = 100

  Run = MOOSDB          @ NewConsole = false
  Run = pMarineViewer   @ NewConsole = false
  Run = pLogger         @ NewConsole = false
  Run = uXMS            @ NewConsole = false
  Run = uProcessWatch   @ NewConsole = false
  Run = pShare          @ NewConsole = false
  Run = pHostInfo       @ NewConsole = false
  Run = uFldShoreBroker @ NewConsole = false
  Run = uFldNodeComms   @ NewConsole = false   <-- Add this line (but not this comment)
}
```

The next step is to add a uFldNodeComms configuration block also to the meta_shoreside.moos configuration file. You are encouraged to take a read through the uFldNodeComms documentation to know more about the below parameters, but here is an example configuration block:

```
ProcessConfig = uFldNodeComms
{
  AppTick    = 2
  CommsTick  = 2

  comms_range      = 120
  min_msg_interval = 15
  max_msg_length   = 1000

  view_node_rpt_pulses = true
}
```

The key parameter initially is the comms_range parameter. Setting this value here to 120 means that any two vehicles must be within 120 meters of one another in order for an inter-vehicle message to go through. The other parameters limit the message frequency and length, and the last parameter affects whether visual artifacts are also produced by uFldNodeComms when it is running. Once you have these changes in place, re-launch the mission.

```
$ cd s21_charlie_dana_nodecomms
$ ./launch.sh 15
```

It should look something like the video posted at:

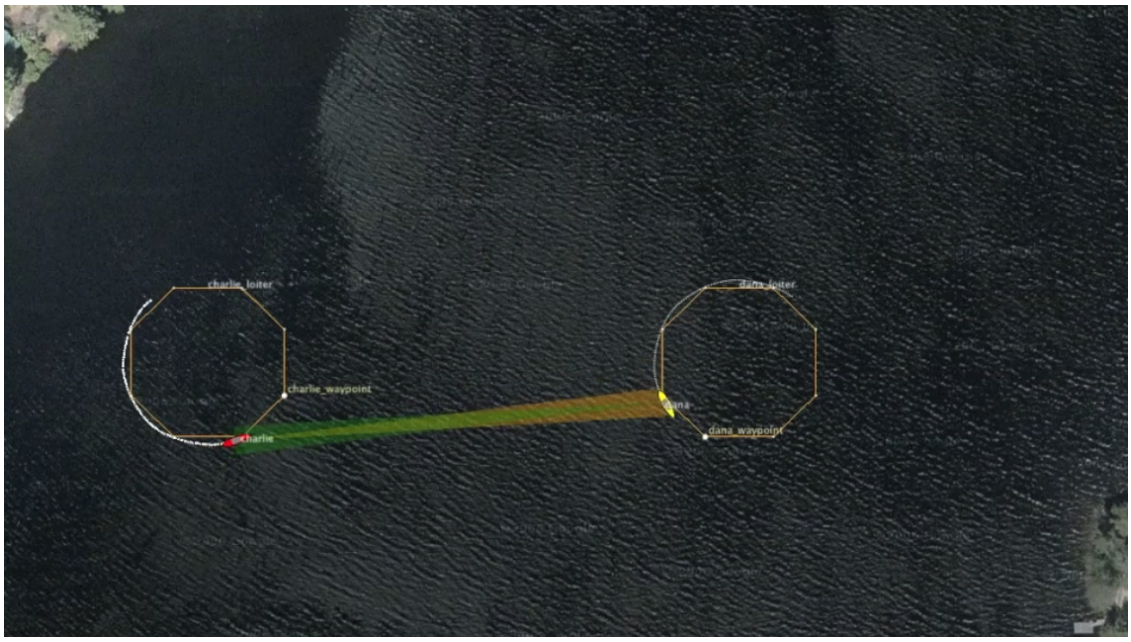

Figure 3: A simple two-vehicle baseline mission with both vehicles loitering at a fixed location, and inter-vehicle communications supported with uFldNodeComms running. The colored cones between vehicles indicate the vehicles are within comms range of each other.

video:(0:21): https://vimeo.com/88540265

3.3 Exercise 3 - The Charlie Dana Message Mission

The next step is to augment your mission to test inter-vehicle communications. The first key addition is the module uFldMessageHandler which runs on each of the vehicles and accepts node messages from other vehicles. To get started, you can make a copy of your previous mission into a folder called charlie_dana_message.

```
$ cp -rp s21_charlie_dana_nodecomms  s22_charlie_dana_message
```

The first step is to add a uFldMessageHandler to your pAntler configuration block in meta_vehicle.moos:

```
ProcessConfig = ANTLER
{
  MSBetweenLaunches = 100

  Run = MOOSDB           @ NewConsole = false
  Run = uProcessWatch    @ NewConsole = false
  Run = pShare           @ NewConsole = false
  Run = uSimMarine       @ NewConsole = false
  Run = pLogger          @ NewConsole = false
  Run = pNodeReporter    @ NewConsole = false
  Run = pMarinePID       @ NewConsole = false
  Run = pHelmIvP         @ NewConsole = false
  Run = pHostInfo        @ NewConsole = false
  Run = uFldNodeBroker   @ NewConsole = false
  Run = uFldMessageHandler @  NewConsole = false   <-- Add this line (but not this comment)
}
```

The next step is to add a uFldMessageHandler configuration block also to the meta_vehicle.moos configuration file. You are encouraged to take a read through the uFldMessageHandler if you have time, but here is an example configuration block:

```
ProcessConfig = uFldMessageHandler
{
  AppTick    = 2
  CommsTick  = 2

  strict_addressing = false
}
```

The strict_addressing configuration parameter indicates whether incoming messages need to be addressed strictly to the named vehicle or whether messages sent to "all" vehicles will also be accepted. We leave it set to false so both kinds of messages are accepted. You can also put the above configuration block into a plug_uFldMessageHandler.moos file and #include it in the meta_vehicle.moos file, but either way is fine for now.

The last step is to share outgoing vehicle messages, generated on a vehicle, to the shoreside for distribution to other vehicles. To do this, the plug_uFldNodeBroker.moos file needs to be augmented:

```
  BRIDGE = src=VIEW_POINT
  BRIDGE = src=VIEW_SEGLIST
  BRIDGE = src=APPCAST
  BRIDGE = src=NODE_REPORT_LOCAL, alias=NODE_REPORT
  BRIDGE = src=NODE_MESSAGE_LOCAL,  alias=NODE_MESSAGE  <-- Add this line (but not this comment)
```

Once you have these changes in place, re-launch the mission.

```
$ cd s22_charlie_dana_message
$ ./launch.sh 15
```

At this point, the vehicles are ready to send messages between each other, but nothing is happening yet.

How does charlie, meaning to be send a message to dana, initiate this message? Charlie posts to its MOOSDB, an outgoing message in the variable NODE_MESSAGE_LOCAL. As described in the uFldMessageHandler documentation, this message has four main components:

- src_node: The name of the vehicle originating the message, in this case charlie.
- dest_node: The name of the vehicle for which the message is intended to be sent, in this case dana.
- var_name: Name of the MOOS variable we want written in the receiving vehicle's MOOS community. In this case the variable name is UP_LOITER because each vehicle is running with a loiter behavior configured with updates=UP_LOITER to accept dynamic behavior parameter changes. See the loiter configuration block inside meta_vehicle.bhv.
- string_val: The contents of the MOOS variable when posted in the receiving vehicle's MOOS community. In this case we use string_var to indicate we are posting a string rather than a double. And in this case we are passing a new loiter y-position to the receiving vehicle. The parameter ycenter_assign is a defined parameter for the loiter behavior, meant just for these situations.

So, the outgoing node message will have the following four components:

- src_node=charlie
- dest_node=dana
- var_name=UP_LOITER
- string_val=ycenter_assign=-50

If you'd like to poke the loiter change *directly* into dana, try this:

```
$ uPokeDB targ_dana.moos UP_LOITER=ycenter_assign=-50
```

To simulate a message going from charlie to dana (the goal/criteria of this exercise), poke charlie with an outgoing message formed as above:

```
$ uPokeDB targ_charlie.moos NODE_MESSAGE_LOCAL="src_node=charlie,dest_node=dana,
          var_name=UP_LOITER,string_val=ycenter_assign=-50"
```

Keep a command window open and try poking the above message alternating between -50 and -100 for the new y-loiter position for dana. It should look something like the video posted at:

Figure 4: A simple two-vehicle baseline mission with both vehicles loitering at a fixed location, and inter-vehicle communications supported with uFldNodeComms running. A message is periodically sent from charlie to dana, indicated by the brief white comms cone between the two vehicles. When they are outside of comms range, no colored comms cones rendered, there are no inter-vehicle messages.
video:(0:29): https://vimeo.com/88569374

3.4 Exercise 4 - The Charlie Dana ReAssign Mission

We augment the previous mission where messages were sent by poking the MOOSDB on the sender vehicle. In this mission, messages originate from both vehicles automatically with a simple timer script to re-assign the location of the other vehicle. To get started, make a copy of your previous mission into a folder called `charlie_dana_nodecomms`.

```
$ cp -rp s22_charlie_dana_message  s23_charlie_dana_reassign
```

The first step is to add a timer script to your vehicles. The timer script essentially does the same thing you did manually by poking the MOOSDB in the Charlie Dana Message mission. A script should be added to your mission by creating a `plug_uTimerScript.moos` file and including this plug in your `meta_vehicle.moos` file.

```
ProcessConfig = uTimerScript
{
  AppTick    = 2
  CommsTick  = 2

  condition  = DEPLOY=true
  randvar    = varname=YPOS, min=-125, max=0, key=at_reset

  // THE BELOW EVENT ALL ON ONE LINE IN THE ACTUAL MOOS FILE

  event      = var=NODE_MESSAGE_LOCAL, val="src_node=$(VNAME),dest_node=all,
               var_name=UP_LOITER,string_val=ycenter_assign=$[YPOS]", time=60:90

  reset_max = nolimit
  reset_time = all-posted
}
```

A re-assign message is periodically sent to the other vehicle. Actually with the syntax above, the message is sent to "all" other vehicles, but in our simple example there is only one other vehicle, and uFldMessageHandler and uFldNodeComms prevent messages from being sent back to the sender. The additional condition of DEPLOY=true is meant to prevent the re-assigning from happening until the vehicles are deployed.
It should look something like the video posted at:

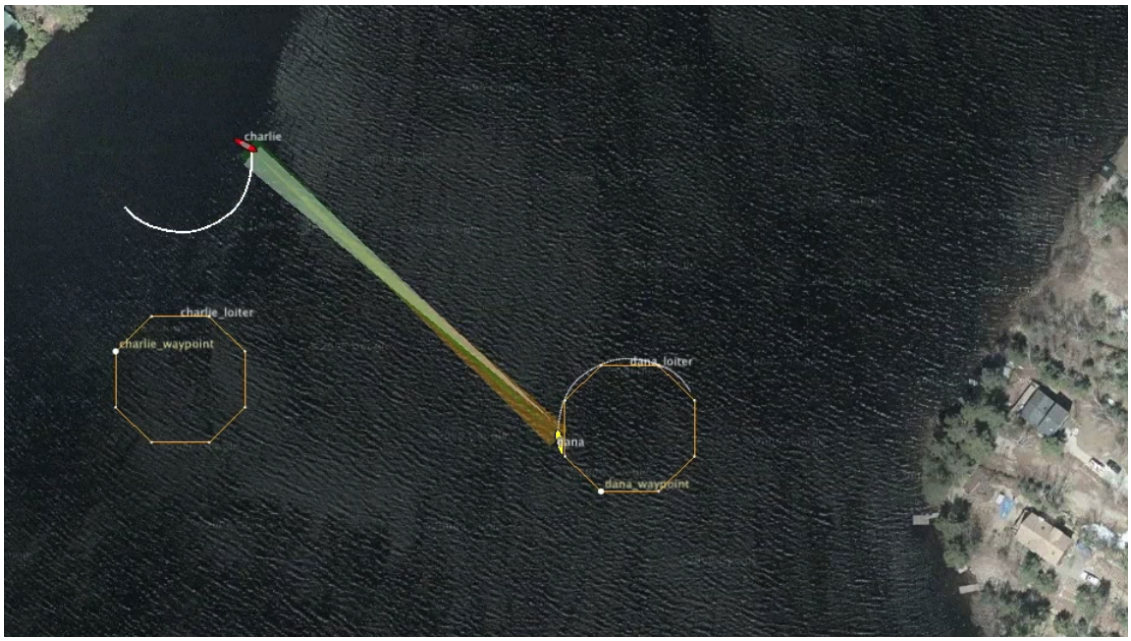

Figure 5: The same simple two-vehicle baseline mission with both vehicles loitering, initially at a fixed location. Inter-vehicle communications is supported with uFldNodeComms and uFldMessageHandler running. A message is periodically sent from charlie to dana, and vice versa, indicated by the brief white comms cone between the two vehicles. Each message re-assigns the other vehicle to a different loiter location either a little bit north or south of its present position. When they are outside of comms range, with no colored comms cones rendered, there are no inter-vehicle messages.
video:(0:52): https://vimeo.com/88640064

3.5 Exercise 5 (bonus!) - The Charlie Dana Recover Mission

In the previous mission, Charlie Dana ReAssign, note that it is possible that the two vehicles re-assign each other to be perpetually out of comms range with one another. Once they are out of comms range, they never change their loiter position to get back into comms range. You may even notice that this almost happens right at the start of the example video in Figure 5. Charlie is loitering in the north-west, and dana in the south-east but dana just manages to squeeze out a message to charlie.

The challenge is to come up with ways to recover from this scenario of being outside of comms range. How can it be detected? What can be done to recover? One way to recover is to have both vehicles return home, and re-connect this way. But can you do better? Can this be done without writing a new MOOS App?

Note that you can make the problem situation a bit more likely to occur, for the sake of efficient testing, if you shrink the comms range by reducing `comms_range=120` in the `uFldNodeComms` configuration block. Or you can expand the possible loiter locations to the north and south by adjusting the `randvar` parameter in the `uTimerScript` configuration on the vehicles.

4 Where to Find the Solutions

All solutions can be dowloaded 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/
```

4.1 Solutions to Exercise: Charlie Dana Baseline Mission

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc2/s20_charlie_dana_baseline.tar
```

4.2 Solutions to Exercise: Charlie Dana NodeComms Mission

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc2/s21_charlie_dana_nodecomms.tar
```

(The uFldNodeComms app added.)

4.3 Solutions to Exercise: Charlie Dana Message Mission

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc2/s22_charlie_dana_message.tar
```

(The uFldMessageHandler app added to each vehicle. Messaging done via uPokeDB to either the source vehicle, or receiving vehicle directly. Mods to uFldNodeBroker bridging to accommodate the message.)

4.4 Solutions to Exercise: Charlie Dana Message Re-Assign

Get the whole folder with:

```
$ wget http://oceanai.mit.edu/mc2/s23_charlie_dana_reassign.tar
```

(The uTimeScript app added to each vehicle. Messaging done on a fixed schedule to a randomly chosen Y location. At some point vehicles will get out of range for talking to each other.)