Lab #9

Autonomous Collaborative Search

2.S998 Unmanned Marine Vehicle Autonomy, Sensing and Communications

Contents

1 Overview and Objectives

1.1 Structure of Today's Lab and Goals

The focus of today's lab will be on autonomous collaborative search. The goal will be to develop a two-vehicle mission configuration where the two vehicles are working together to locate and identify hazards in a given region. There will be constraints on the vehicle speed, inter-vehicle comms range, and sensor characteristics. The end-goal is to make the best educated report on the location and type of located objects in the search field. The performance score will be based on a reward/penalty system applied to correct/incorrect elements of the final report.

In this lab, there are three MOOS apps that are new arrivals to our labs:

- `uFldHazardSensor` - A MOOS simulated search sensor.
- `uFldMessageHandler` - A MOOS handler for incoming message from other vehicles.
- `uFldNodeComms` - A MOOS shoreside arbiter of inter-vehicle messages.

The goal of this lab is to begin exploring, in a tangible way, the relationship between autonomy, sensing and communications - the primary theme of the course. In this lab, the issue of inter-vehicle communications is introduced, with the sole limitation being a range dependency (any two vehicles need be within N meters to be able to communicate). Other than a range dependency, for now our inter-vehicle messaging is uninhibited. Later we will introduce restrictions on message frequency and message bandwidth.

A further goal of this lab is to stress the need for demonstrating the student final work during an in-class simulation where the shoreside community and shoreside MOOS apps are run by the TAs. To this end, we will make provisions to meet with students outside normal class / lab hours to make sure that missions are readily runnable when it comes time to the Lab 11, one week from this lab, when student work will be demonstrated.

In this lab, students are encouraged to work in pairs. By the end of today's lab, this partnering arrangement should be worked out for all students.

1.2 Preliminaries

Standard practice now before any lab should be to perform an svn update within your moos-ivp tree and rebuild if you see any updates pulled down from the SVN server. 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/MOOS/MOOSBin/MOOSDB
$ which uTimerScript
/Users/you/moos-ivp/bin/uTimerScript
$ which mykill
/Users/you/moos-ivp/scripts/mykill
```

1.3 More Resources on Sensing and Inter-Vehicle Comms for Today's Lab

The one document that may be most useful to the discussion in this lab is the documentation for the uField Toolbox linked below. See also the slides from today's lecture.

- Today's lecture notes.
- The uField Toolbox Documentation
  http://oceanai.mit.edu/moos-ivp-pdf/moosivp-ufield.pdf

  This contains the documentation for the `uFldHazardSensor`, `uFldMessageHandler`, and `uFldNodeComms` applications. All three applications are used in this lab for the first time.
- The IvP Helm documentation.
  http://oceanai.mit.edu/moos-ivp-pdf/moosivp-helm.pdf
- The moos-ivp.org website documentation.
  http://www.moos-ivp.org

2 Inter-Vehicle Messaging

The first exercise in today's lab involves inter-vehicle messaging. Since inter-vehicle messaging is a component of the overall collaborative search problem in later sections of the lab, we isolate this issue first with a simpler example.

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. A video of the working end-state may be found at:

http://www.youtube.com/watch?v=fTPugowyBg0&feature=youtu.be

Note that while the mission assigned in Section 3 of this lab is due next Thursday, March 15th, *this mission is due in lab next Tuesday, March 13th.* This assignment will not be handed in, but rather its working state is to be demonstrated in lab on Tuesday the 13th.

2.1 Building a Messaging Mission with the uField Toolbox Modules

The steps describe in this section are aimed at building up the two-vehicle mission, leading to a configuration that is all runnable on a single laptop. In Section **??**, the focus will be instead running on multiple machines.

2.1.1 Documentation and Module Topology for the uField Toolbox Modules

The primary new modules being used in this lab are the `uFldMessageHandler` and `uFldNodeComms` modules. See today's (lecture 09) class notes for a description, and see the uField Toolbox documentation available on the course website for full documentation. The figure below from today's lecture illustrates the basic setup. The focus today is on these two modules.

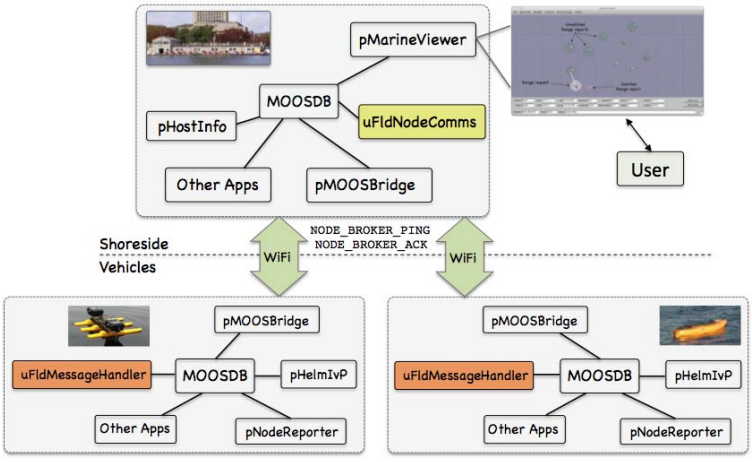

Figure 1: **The uFldMessageHandler and uFldNodeComms Modules:** The footnotesize `uFldMessageHandler` runs on each vehicle and parses incoming footnotesize `NODE_MESSAGE` postings. The footnotesize `uFldNodeComms` module runs on the shoreside and routes messages to their destination vehicle(s).

2.1.2 Getting the Baseline Mission

The first step is to copy the baseline example mission from the moos-ivp tree and copy it into your own moos-ivp-jsmith tree.

```
% cp -rp moos-ivp/ivp/missions/lab09one_baseline  moos-ivp-jsmith/missions/lab09one
```

The `lab09one` mission file should like:

```
% ls lab09one
  clean.sh*                  plug_pBasicContactMgr.moos  plug_pNodeReporter.moos
  launch.sh*                 plug_pHelmIvP.moos          plug_uFldNodeBroker.moos
  meta_shoreside.moos        plug_pHostInfo.moos         plug_uProcessWatch.moos
  meta_vehicle.bhv           plug_pLogger.moos           plug_uSimMarine.moos
  meta_vehicle.moos          plug_pMOOSBridgeV.moos      plug_uXMS.moos
  plug_origin_warp.moos      plug_pMarinePID.moos
```

As with previous missions, the actual mission files are auto generated upon executing the launch scripts. The script supports the `-h` switch and the `--just_build` switch if you want to just build the target files without actually launching. For example:

```
% cd lab09one
% launch.sh --just_build
% ls
  clean.sh*                  plug_pHostInfo.moos         plug_uXMS.moos
  launch.sh*                 plug_pLogger.moos           targ_gilda.bhv
  meta_shoreside.moos        plug_pMOOSBridgeV.moos      targ_gilda.moos
  meta_vehicle.bhv           plug_pMarinePID.moos        targ_henry.bhv
  meta_vehicle.moos          plug_pNodeReporter.moos     targ_henry.moos
  plug_origin_warp.moos      plug_uFldNodeBroker.moos    targ_shoreside.moos
  plug_pBasicContactMgr.moos plug_uProcessWatch.moos
  plug_pHelmIvP.moos         plug_uSimMarine.moos
```

Note the five target files newly created.

Confirm that the baseline mission launches and runs on your machine.

2.1.3 Add the uFldMessageHandler Configuration

A first step is to add the `uFldMessageHandler` app to each of the vehicle mission configurations (in the `meta_vehicle.moos` file. Create a new plug file, and a new `#include` for the plug-in, and add the app to the Antler block. In this case, all the default configuration values for this app are fine, so a simple block like the one below suffices:

```
  ProcessConfig = uFldMessageHandler
  {
    AppTick   = 4
    CommsTick = 4
  }
```

2.1.4 Add the uFldNodeComms Configuration

The next step is to add the `uFldNodeComms` app to the *shoreside* mission configuration (in the `meta_shoreside.moos` file. Create a new plug file, and a new `#include` for the plug-in, and add the app to the Antler block. In this case, there are only three configuration parameters of interest to us for this mission. They are described below, but further info can be found in the documentation.

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

  COMMS_RANGE      = 500    // Must be within 500 meters of each other
  MIN_MSG_INTERVAL = 0      // As often as we like
  MAX_MSG_LENGTH   = 0      // As long as we like
}
```

2.1.5 Simple Testing via Poking the MOOSDB

At this point you should be able to launch your mission, and even before deploying the vehicle, you should be able to test inter-vehicle messaging. This may be done by poking the MOOSDB of one vehicle with an outgoing message, and looking for it on the other vehicle. Try poking the MOOSDB of the vehicle *gilda* by:

```
$ uPokeDB targ_gilda.moos NODE_MESSAGE_LOCAL="src_node=gilda,dest_node=all,  \
                          var_name=TESTMSG,string_val=hello"
```

If things are going as they should, you should be able to see output similar to that below, in the terminal window of `uFldNodeComms`:

```
*************************************
uFldNodeCommsSummary:
*************************************

Node Report Summary
===================================
        Total Received: 1771
                 GILDA: 881       (0.5)
                 HENRY: 890       (0.5)
     ------------------
            Total Sent: 694
                 GILDA: 347
                 HENRY: 347

Node Message Summary
===================================
    Total Msgs Received: 1
                 GILDA: 1      (60.3)
     ------------------
```

```
            Total Sent: 1
                 HENRY: 1
      ------------------
     Total Blocked Msgs: 0
                Invalid: 0
           Stale Receiver: 0
               Too Recent: 0
             Msg Too Long: 0
            Range Too Far: 0
      ------------------
              Last Msgs:
src_node=gilda,dest_node=all,var_name=TEST_MSG,string_val=hello
```

2.1.6 Adding a uTimerScript Script for Automatic Messaging

In the next step, add a script with `uTimerScript` to post an outgoing message locally. This message, when received by the remote vehicle, should have the effect of altering the location of the region used by the loiter behavior. To do this in the simplest way, use the `ycenter_assign` configuration parameter for this behavior. To see a short summary of the this behavior's interface, go to:

http://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php?n=Behaviors.Loiter

In short, the message received by each vehicle should correspond to the the MOOS variable specified as the `updates` variable in your configuration. Configure your timer script to have two alternating events, one that re-assigns the loiter location to the north, and one that re-assigns it to the south.

2.1.7 Confirming that it Works

Finally, confirm that it all works. You should be able to launch all three MOOS communities (shoreside + two vehicles), launch both vehicles with a single button click and just watch the mission unfold. You should be able to see a visual confirmation of the inter-vehicle messaging with periodic appearances of a white cone between vehicles. The fat end of the cone indicates that it is the receiving vehicle. To see what this all should look like, check out clip 91 on the "lab videos" section of the class website, or go right to the YouTube link below:

http://www.youtube.com/watch?v=fTPugowyBg0&feature=youtu.be

2.2 Testing the Messaging Mission on Multiple Machines

Note: The in-lab demonstration of this mission is due *Tuesday* March 13th, during lab hours.

2.2.1 Preparing for In-Lab Demonstration

In preparation for in-lab demonstration, work with your lab partner to use both of your machines. Use one machine acting as the shoreside community and the other to launch the two vehicles.

2.2.2 In-Lab Demonstration

The in-lab demonstration will proceed with the TA launching a shoreside community and each lab partner launching a single vehicle. Successful demonstration is defined by (a) both vehicles sending and receiving their intended messages as prescribed previously in Section 2.1, and (b) both vehicles being rendered on the TA's machine in pMarineViewer.

3 Collaborative Search

3.1 Overall Mission Description and Tips for Getting Started

The goal of this mission is to deploy two marine vehicles collaborating to detect and classify shallow-water hazards. Each vehicle will be able to use a simulated hazard sensor, `uFldHazardSensor`. The autonomy mission, in terms of waypoints, sensor settings, strategies for inter-vehicle communications, and maximizing mission reward is completely up to each team. You are encouraged to work in teams of two.

The Hazard File

The hazard field is a file generated by the `gen_hazards` command line application built as part of the moos-ivp tree. You can generate output by:

```
% gen_hazards --polygon=-150,-75:-150,-400:400,-400:400,-75 --objects=5,hazard --objects=8,benign
Proper polygon specified: -150,-75:-150,-400:400,-400:400,-75
x=64.1,y=-358.20001,type=hazard
x=2.5,y=-153.3,type=hazard
x=145.90001,y=-352.89999,type=hazard
x=267.80002,y=-250,type=hazard
x=37.7,y=-394.80002,type=hazard
x=-118.7,y=-155.3,type=benign
x=222,y=-312.39999,type=benign
x=31.7,y=-365.30002,type=benign
x=76.8,y=-332.89999,type=benign
x=-78.1,y=-381,type=benign
x=242.60001,y=-340.89999,type=benign
x=-63.8,y=-156.2,type=benign
x=223.2,y=-255.40001,type=benign
```

You may redirect the above output to a file, by appending `"> file.txt"` to the above command-line invocation.

The Baseline Mission

A baseline mission for this lab is available. It is advised that you begin the construction of your `lab09two` mission (this assignment) by starting with the baseline mission:

```
% cp -rp moos-ivp/ivp/missions/lab09two_baseline  moos-ivp-jsmith/missions/lab09two
```

This mission launches (`"./launch.sh 10"` a shoreside vehicle and two vehicles, *archie* and *betty*. Deploying archie first will start a lawnmower pattern mission. Upon archie's return, betty will traverse a set of waypoints with a different sensor setting.

This mission does not contain any provisions for keeping track of sensor data, or path re-planning. This is up to you. See the discussion in Section 3.2 regarding `pHandleSensorData` and `pHazardPath`.

3.2 Application Modules Important to this Lab

The following applications are important to this lab, including two that you will need to write yourself:

The uFldHazardSensor Application

This tool is your simulated sensor. It runs in the shoreside MOOS community. During in-lab demonstrations, it will be launched and configured by the TAs, but you will need to have your own shoreside community configured during your own development and testing. It is described in the uField Toolbox documentation found on the course website. The baseline mission provided for this lab also makes use of this application.

The uFldMessageHandler Application

This application runs on each of the vehicles. It is used for unparsing inter-vehicle messages. You should be fairly familiar with this now as it was the focus of Section 2 in the first part of this lab.

The uFldNodeComms Application

This application runs on the shoreside computer. It is used for routing messages between vehicles. You should be fairly familiar with this now as it was the focus of Section 2 in the first part of this lab.

The pHandleSensorData Application

You are free to implement your solution however you see fit. However, it is likely that you will need an application that simply manages the output of your sensor information, updates it with new sensor information from subsequent passes and other vehicles, and reports it to the shoreside when the mission is complete. The recommended name for such an app or similar is `pHandleSensorData`. This will help us understand what is going on in your implementation when/if we need to help you, and when grading your handed in assignment.

Note: One function of the above app may be simply to ensure that `UHZ_SENSOR_REQUEST` is being published and bridged to the shoreside. Recall that this message must be received from the vehicle regularly in order for the simulated sensor to generate reports to that vehicle.

The pHazardPath Application

You are free to implement your solution however you see fit. However, it is likely that you will need an application that reasons about your vehicle's waypoints. The recommended name for such an app or similar is `pHazardPath`. This will help us understand what is going on in your implementation when/if we need to help you, and when grading your handed in assignment.

3.3 Ground Rules

The following ground rules are relevant to this lab.

3.3.1 Operation Area

The simulated hazard field will be within a polygon with the following four vertices:

- (-150, -75)
- (-150, -400)
- (400, -400)
- (400, -75)

The region was chosen to be rectilinear to simplify the lawnmower pattern generation for a waypoint behavior. See the documentation for this behavior. A pattern may be given to the behavior simply by specifying the characteristics of the lawnmower pattern without having to explicitly determine all the points.

3.3.2 Time and Top Speed

You mission will will have a maximum duration of 150 minutes, or 9000 seconds. The plan is to test at 30x real time in the labs. So we should be able to see each team perform its full mission in 5 minutes of real time.

The top vehicle speed for each vehicle will be 2.0 meters per second. This will be monitored on the shoreside. If a vehicle is noted to go over the maximum speed a penalty may be invoked in the form of cutting off access to the sensor simulator, or inter-vehicle messaging, or both.

3.3.3 Sensor Configuration Frequency

You will be allowed to reset your sensor configuration once every 20 minutes, per vehicle. Attempts to switch configurations more frequently will simply be ignored. (Currently this is not enforced in the uFldHazardSensor app, but it will be enforced by next week).

Your sensor configuration options will be:

- sensor_config = width=10, exp=6, class=0.93
- sensor_config = width=25, exp=4, class=0.80
- sensor_config = width=50, exp=2, class=0.60

See the documentation for `uFldHazardSensor` and the class notes on what this entails in terms of the ROC curve performance of the sensor, and how the vehicle may request a particular configuration.

3.3.4 Scoring Metrics

The scoring metrics (as discussed in today's lecture material) are as follows:

- +10: Hazard guessed correctly as a hazard
- -50: Hazard guessed incorrectly as benign
- -75: Hazard not reported at all
- -25: Benign guessed incorrectly as a hazard
- +5: Benign guessed correctly as benign

- 0: Benign not reported at all

Your job is to report, before the end of the mission time, a series of messages of the following form:

```
HAZARD_REPORT = x=45,y=88,hazard=true,label=04    // objects believed to be a hazard
HAZARD_REPORT = x=15,y=18,hazard=false,label=16   // objects believed to be benign
```

Messages reported late are ignored. Duplicate and conflicting messages will be treated as if nothing at all was reported for that object. You must ensure this variable is bridged to the shoreside community with an entry to `uFldNodeBroker`.

3.3.5 Inter-Vehicle Communications

In this lab there is no limit to the frequency or bandwidth in inter-vehicle communications. This will likely change in later labs. However, in this lab your vehicles will not be allowed to communicate at ranges greater than 100 meters. This is enforced in the `uFldNodeComms` configuration. This module is on the shoreside and will be configured by the TAs during testing, but you should configure your own `uFldNodeComms` module for this restriction during your development.

3.3.6 Collision Avoidance

We have not yet covered the issue of inter-vehicle collision avoidance, or means for ensuring operation without hitting land. You will not be penalized for either type of collision in this lab.

3.3.7 Depth

Depth is not a factor in this sensor simulator, so you can assume the simulation of a surface vehicle for this lab. Your helm need not be configured for depth operations.

3.4 Due Date and In-Lab Testing

This part of the lab is due next Thursday, March 15th, in lab. Handing in your assignment will work a bit differently in this lab. Handing in your work will consist of three parts:

- Demonstrating your mission to a TA, with the TA providing the shoreside.
- Uploading your source code.
- Uploading a short writeup (at most two pages, one page is fine) describing the autonomy strategy used, and why. It's also good to discuss known limitations of your approach.

Your assignment will not be considered submitted until you have completed all three. The write-up should be in PDF format included in the top-level directory of your handed in code, in a file named lab9-writeup.pdf.

A final note: We will be returning to this problem of collaborative search in labs later in the semester. *The single most important component of this lab is that you have a working system by the March 15th lab. There are many ways to optimize and strategize your autonomy approaches in*

this lab, and we do eventually want to explore those ideas as much as possible. But, in this lab, it is most important to have at least a brute force naive working system handed in on time and able to be demonstrated in next Thursday afternoon's lab.

4 Help in Determining Your Machine's IP Address

In Linux, the IP address may address may be found by the following means (among perhaps many ways), from the command line:

```
$ ifconfig eth1
```

This is assuming that `eth1` is your network interface. Other possibilities are `eth0`, `wlan0`, and so on, depending on your particular computer. (Type `ifconfig` with no argument to see a list of interfaces). If you want to use grep to prune out some of the verbiage, try the following on the command line:

```
$ ifconfig eth1 | grep 'inet addr:'| grep -v '127.0.0.1' | cut -d: -f2 | awk '{ print $1}
```

On OS X, try the following

```
$ networksetup -getinfo <interface>
```

where `interface` is either `"Wi-Fi"` on OS X Lion, or `"Airport"` on pre-Lion machines, or `"Ethernet"` if you're connected by an ethernet cable. The IP address may also be found by using the System Preferences GUI interface under the Network section. Select the active interface (the one with the green button at the top of the list). Select Advanced. Then select TCP/IP. The IP address should be then visible.

2.S998 Marine Autonomy, Sensing and Communications

Spring 2012