MOOS-ROS bridge and MOOS/ROS autonomy/processing code overview

August 17, 2018

Erin Fischell

efischell@whoi.edu

**Purpose**

This document describes the MOOS/ROS autonomy/processing code structure for the MARINER project.

**Structure**

Code for the MARINER project exists in two trees- mariner-src and missions-mariner. All compiled and source code is located in mariner-src, and missions code is located in missions-mariner. Both are git repositories, contact Erin Fischell ([efischell@whoi.edu](mailto:efischell@whoi.edu)) to request access (provide your git username).

**Dependencies:**

1. moos-ivp, located at <http://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php?n=Site.Download>

2. all dependencies located in mariner-src/DEPENDENCIES

If you are running Ubuntu 16.04, running the dependency script will install dependencies. If not, you may need to do so by hand.

**Installing and Building:**

Install all dependencies, request/get access to both repos, and check out mariner-src:

> cd

> mkdir mariner-workspace

> cd mariner-workspace

> git clone https://github.com/efischell/missions-mariner.git

> git clone https://github.com/efischell/mariner-src.git

Next, install moos-ivp and dependencies:

> svn co https://oceanai.mit.edu/svn/moos-ivp-aro/releases/moos-ivp-17.7 moos-ivp

> cd moos-ivp

> ./build-moos.sh

> ./build-ivp.sh

> cd ../mariner-workspace/mariner-src/

> sudo ./dependencies.sh

Add moos-ivp bin etc. to path:

> sudo emacs .bashrc

Add moos-ivp/bin to your path in the ~/.bashrc file:

> emacs .bashrc

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

export PATH=~/mariner-workspace/moos-vp/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib:/usr/include:~/local/bin:/usr/local

export IVP\_BEHAVIOR\_DIRS=~/mariner-workspace/moos-ivp/lib/:~/mariner-workspace/mariner-src/lib

source /opt/ros/kinetic/setup.bash

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Build mariner-src code:

> ./build.sh

Run simulation in missions-mariner:

> cd mariner-workspace/missions-mariner/auv/darter

> ./simulation\_launch.sh

You should see a simulated AUV running in pMarineViewer, and windows should pop up for the bridge, pHelmIvP, etc.

**How to use the missions structure: plugs and automatic run**

missions-mariner contains a structure that automatically takes variables and plugs them into all moos and ros app configuration files, then starts ros nodes, moos apps, and the bridge between them using a single script. This section provides an overview of where to put new variables etc.

In missions-mariner there are several directories: cruise, auv, global\_defaults, logs, sensors, and topside. This structure is designed to make it possible to easily switch a single vehicle to have different sensors and cruises in runtime and simulation.

General: .rplug are plugs for ros apps, .plug are plugs for moos apps, .meta files describe nodes/applications that need to be run, .yaml files control configuration/variables of nodes and apps.

*Cruise directory:*

Create a folder within missions-mariner/cruise/ for each physical location the AUVs will be deployed. Choose which cruise you want to run using the cruise\_config.sh script:

> cd missions-mariner

> ./cruise\_config.sh ashumet\_pond

Each cruise directory contains the following:

cruise.yaml: configuration variables for the specific cruise, including lat/long, opbox, default vehicle behaviors, gps period information, soundspeed and water depth information.

cruise.def: this is automatically populated from cruise.yaml for input into moos

datum.plug: tells moos what lat/long to use for origin

mariner\_missionmanage.plug: contains config for ros node mariner\_missionmanage

cruisebridge.xml: xml file with vars for bridging ros/moos

moosros.rplug: plug for moosros bridge

bridge.moos: moos for moosros bridge

bhv\_ops\_safety.plug,bhv\_avoid\_collision.plug: behavior-specific cruise plugs for moos-ivp

pMarineViewer.plug: config for pMarineViewer

data/modemidlookup.txt: vehicle names/modemids. If your vehicle is not in this list, it will not show up in pMarineViewer

data/\*.info: information (lat/long on corners) for .tif file used by pMarineViewer

Put variables in the cruise.yaml and cruise directory plugs in the cruise folder that will change with each specific cruise. E.g. variables that vary with environment/location, such as vehicle behavior and environment parameters.

*sensors:*

Sensor-specific plugs/rplugs.

*auv:*

The auv directory contains folders ‘auv\_plugs’ and a directory for each specific auv.

auv/auv.yaml: contains defaults for all AUVs. These variables will be overridden by values in a specific auv allconfig.yaml or in a cruise.yaml.

auv/auv\_plugs/\*: plugs and rplugs for apps/nodes that apply to all AUVs (e.g. behavior plugs that might run on any AUV, loggers, interface plugs, simulation apps, etc.

auv/darter/allconfig.yaml: auto-generated file with all configuration parameters currently used with ros/moos plugs. Do not edit this file!

auv/darter/moos.meta: moos application to run

auv/darter/ros.meta: ros application to run

auv/darter/all.yaml: top-level config for the AUV- will override all other defautls, put vehicle-specific config here.

auv/darter/mission.yaml: mission configuration for the specific auv, including mode definitions

auv/darter/last.moos: .moos file run by pAntler (auto-built each time you run)

auv/darter/lastros.launch: last ros config to be launched (auto-built each time you run)

auv/darter/bhv.meta: behaviors to include for moos-ivp

auv/darter/current.bhv: bhv configs run by moos-ivp (auto-built each time you run)

auv/darter/simulation\_launch.sh: script to run simulation (sim sensors/auv)

auv/darter/runtime\_launch.sh: script to run runtime (real sensors/auv)

*Adding flags:*

Flags may be added to missions-mariner/flags.txt to provide more options on what to run when/where.

*What does simulation\_launch.sh do?* simulation\_launch.sh links to missions-mariner/.scripts/simulation\_launch.sh, which in turn calls missions-mariner/.scripts/launch\_yaml.sh. launch\_yaml.sh plugs all variables from .yaml files into .rplug and .plug confgs for ros and moos, turns moos.meta and ros.meta into last.moos and lastros.launch files, then executes moos apps using pAntler and ros nodes using roslaunch. The end result is to launch a full simulation with bridged moos/ros variables and both moos and ros processes using the common set of configuration variables for the appropriate cruise, sensors, and AUV found in the .yaml files. A single set of variables is defined and placed in allconfig.yaml and allconfig.ydef so that the user can see what was run, and the launch files are also available for view. This saves multiple definition, and should make it easy to add cruise, auvs, and sensor configurations and to switch between them. The main difference between runtime and simulation is which variables etc. are bridged, and that uSim\* moos apps are run in simulation to emulate vehicle motion.

**Adding ROS applications in mariner-src**

Creating a new ROS package (python):

To create a new python ROS package, run the following:

> cd ros/src/

> catkin\_create\_pkg yourpackagename rospy --rosdistro kinetic

Write your ros code, yourpackagename/src/the\_file.py. Next, make the .py executable:

> chmod u+x the\_file.py

Re-run the build script.

Next:

> source ros/devel/setup.bash

> rosrun yourrospackage yourfile.py

To add the ROS node to the automatic run of simulation or runtime, add a “.rplug” file to the appropriate location in the missions-mariner structure (e.g. if it is specific to any AUV, to auv/auv\_plugs, if it is specific to the splitbeam sonar sensor sensors/splitbeam/, if it is specific to the ashumet\_pond cruise to cruise/ashumet\_pond/). Then add the path to the filelist in ros.meta in the auv/darter directory (replacing darter with the auv you want it to run on).

Example: cruise/ashumet\_pond/mariner\_missionmanage.rplug:

# ros plug for mariner\_missionmanage missionmanage node

node:

name: "missionmanage" #required

pkg: "mariner\_missionmanage" #required: package

type: "missionmanage.py" #required: executable

viewmode: "xterm" # not required, xterm or quiet

# param vals: can be optional or required

param:

start\_state: $(DEPLOY\_MISSION)

switch\_state: "LOITER"

Example: auv/darter/ros.meta:

# meta file for launching ros apps

# list ros config files here!

# Use yaml list format

filelist:

- cruise/current/mariner\_missionmanage.rplug

- sensors/splitbeam/splitbeam\_parser.rplug

- cruise/current/moosros.rplug

**Adding MOOS applications**

> mkdir moos-src/pYourApp/

Create the moos application, including a CmakeLists.txt. Follow an existing application as a template, see <http://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=Helm.MOOSOverview> for more information.

To add the MOOS app to the automatic run, add a “.rplug” to the appropriate directory, then add the application to the auv/darter/moos.meta file.

Example moos.plug:

ProcessConfig = uSimMarine

{

AppTick = 4

CommsTick = 4

prefix = NAV

start\_x = $(LAUNCH\_X)

start\_y = $(LAUNCH\_Y)

start\_heading = $(LAUNCH\_HEADING)

start\_speed = 0

start\_depth = $(LAUNCH\_DEPTH)

start\_pos = x=$(LAUNCH\_X), y=$(LAUNCH\_Y), speed=0, heading=$(LAUNCH\_HEADING), depth=$(LAUNCH\_DEPTH)

force\_x = 0

force\_y = 0

force\_theta = 0

force\_vector = 0,0 // heading, magnitude

buoyancy\_rate = 0.025 // meters/sec

max\_acceleration = 0.2 // meters/sec^2

max\_deceleration = 0.5 // meters/sec^2

max\_depth\_rate = 0.5 // meters/sec

max\_depth\_rate\_speed = 2.0 // meters/sec

sim\_pause = false // or {true}

dual\_state = false // or {true}

thrust\_reflect = false // or {true}

thrust\_factor = 20 // range [0,inf)

turn\_rate = 70 // range [0,100]

thrust\_map = 0:0, 20:0.50, 40:1.0, 60:1.5, 80:2.0, 100:2.5

#ifdef TRIM\_TOLERANCE

trim\_tolerance = $(TRIM\_TOLERANCE)

#endif

#ifdef MAX\_TRIM\_DELAY

max\_trim\_delay = $(MAX\_TRIM\_DELAY)

#endif

}

Example moos.meta:

// laboratory for autonomous marine sensing systems

// massachusetts institute of technology

//

// mission configuration meta-moos file

// for the node whose folder this file resides in

//

// use simulation\_launch.sh and runtime\_launch.sh in this

// directory to launch the respective simulation and vehicle runtime

// missions

//

// questions about this setup should be directed to t. schneider: tes@mit.edu

////////////////////////////////////////////////////////////

// definitions

////////////////////////////////////////////////////////////

#include allconfig.ydef

////////////////////////////////////////////////////////////

// inheritance

////////////////////////////////////////////////////////////

NoNetwork = true

ServerHost = $(HOST)

ServerPort = $(PORT)

Community = $(VNAME)

MOOSTimeWarp = 1

#include cruise/current/datum.plug

@ANTLER Run = MOOSDB @ NewConsole = false

#include auv/auv\_plugs/pMarinePID.plug

#include auv/auv\_plugs/pHelmIvP.plug

#ifdef simulation

#include auv/auv\_plugs/uSimMarine.plug

#include cruise/current/pMarineViewer.plug

#include auv/auv\_plugs/uProcessWatch.plug

#include auv/auv\_plugs/pNodeReporter.plug

#endif

#include auv/auv\_plugs/uXMS.plug

#include auv/auv\_plugs/pLogger.plug

#ifdef runtime

#endif

**Adding vehicles**

1. Add a vehicle directory in missions-mariner/auv/, copying structure from auv/darter/
2. Add a line in missions-mariner/cruise/data/modemidlookup.txt for new vehicle
3. Modify vars as needed
4. Test in sim

**Adding cruise**

1. Add directory in missions-mariner/cruise, copying structure from cruise/ashumet\_pond/
2. Create a .tif from google earth using instructions here: <http://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=Tools.PMarineViewer> (background image section)
3. Choose op box and default behaviors
4. Test in sim

**MOOS-ROS Bridge:**

Code is based on

**Next steps:**

1. Bridging of nav variables
2. Improved simulation modules: drop-in for RECON using MOOS that goes MOOS-ROS-MOOS (currently a single MOOS community is doing all vehicle sim, which doesn’t test bridging)
3. Development of ROS applications for auto-processing of sonar data
4. runtime configuration
5. flags for warp, etc. (not yet implemented)