How to use the *"aquaticus1.0"* mission?[☆,☆☆]

Mohamed Saad Ibn Seddik[a,1]

[a] *MIT, 77 Massachusetts Avenue, Cambrige 02139, MA*

Abstract

The *"aquaticus1.0"* mission is a base mission to prove the concept of the *AQUATICUS* project where humans and robots collaborate in teams to achieve a common goal in a marine environment. This mission is the first example where teams of humans and robots are playing a simple capture the flag, each team is composed of one human and one robotic teammate. The robot has a set of pre-programmed behaviors that are triggered by voice commands from the human over RF link. This document explains how to launch the mission in both simulation and real-life.

Keywords: marine robotics, field robotics, USV, autonomous vehicles

1. Prerequisites

1.1. Basic requirement

To be able to run the "aquaticus1.0" mission, please make sure that:

1. The latest version of *moos-ivp* tree is downloaded and compiled;
2. The latest version of *moos-ivp-colregs* tree is downloaded and compiled[2];
3. The latest version of *moos-ivp-aquaticus* tree is downloaded and compiled.

It is recommended that all the folders of the list are located on the same level for easy discovery. If not, special configuration might be needed.

1.2. Advanced tools

1.2.1. SSH keys

It is recommended to have the own SSH keys deployed on the targets to accelerate the login into the targets and minimize the time spend entering the username/password.

[☆] The *"aquaticus1.0"* mission is available under the tree *moos-ivp-aquaticus*.

[☆☆] This document is targeted to the users of the mission. It should contain all the necessary information to run the mission flawlessly.

Email address: msis@mit.edu (Mohamed Saad Ibn Seddik)

[1] Post-doctoral Associate at the Mechanical Department, MIT.

[2] The *moos-ivp-colregs* is required to have the vehicle behave in compliance with the COLREGS regulation.

SSH keys can be copied using a one line command:

```
ssh-copy-id user@target
```

The previous command will automatically copy the SSH keys from the local computer to the remote *target* and will allow the *user* to login without prompting login screen.

1.2.2. fabric

fabric is a python tool used by system administrators to manage large numbers of servers through SSH without having to log in to each and every server and launching a script or different commands.

One can easily see the similarity between the world of cloud computing and field robotics where different systems need to be configured and launched multiple times on usually heterogeneous systems to accomplish different goals.

For this reason, the use of *fabric* is recommended but not necessary. In the case of this mission, the fabric script allows to:

1. Update the directories with the latest version of the code from trunk;
2. Build the source code on all targets;
3. Launch the mission on all the targets simultaneously;
4. Retrieve the logs from all the targets to be studied later.

When SSH keys are set-up of the targets, all the previous commands can be run in parallel saving precious time.

On most recent OS (Mac and Linux), Python is pre-installed, therefore making the installation of *fabric* a one step process. In order to install *fabric*, please follow the instructions in the official page[3].

2. Launching the mission

In this section we suppose that the current directory is *missions/aquaticus1.0* under the *moos-ivp-aquaticus* tree.

2.1. Shoreside

We call *shoreside* the main computer that will ran the view of the vehicles and the terrain but also run some crucial applications for the mission, for instance managing tags and flags etc.

To launch the *shoreside*, the directory must be changed:

```
cd shoreside/
```

A script is available under that directory to smoothly launch the view and other apps:

[3] http://www.fabfile.org/installing.html

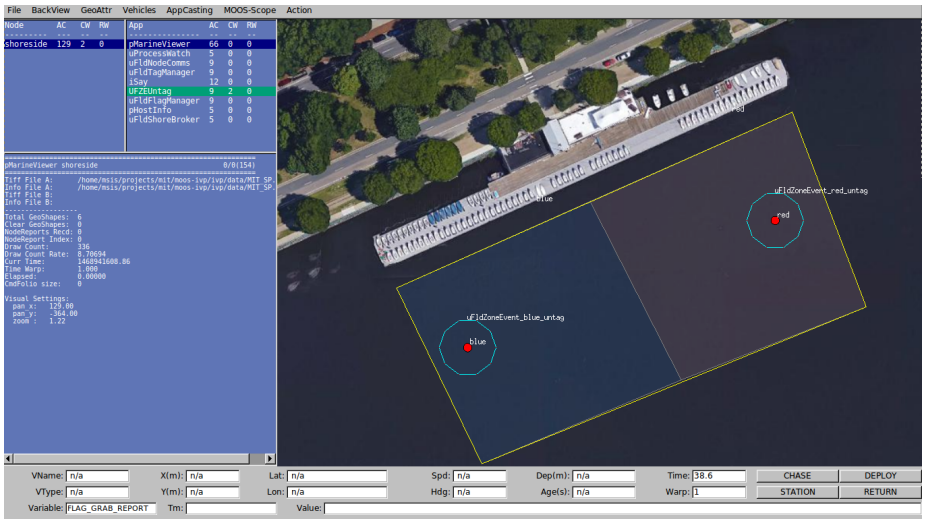

Figure 1: Shoreside view from pMarineViewer

```
./launch_shoreside.sh [SWITCHES]
 --shore-port= , set up a shore listening port. (Default is 9300)
 --shore-ip=   , set up a shore listening IP. (Default is localhost)
 --just_make, -j
 --help, -h
```

When this script is launched, it will set up and launch all the required applications and you should see a view similar to Figure 1 on page 3.

Whether the mission is running in simulation or on the real vehicles, the shoreside stays unchanged and is supposed to behave the same in both cases.

Once done, pushing the *q* key on the terminal will kill all the applications that have been launched.

Logs of the mission with the corresponding date and time of the launch are saved under *LOG_SHORESIDE_DD_MO_YYYY_____HH_MM_SS*.

2.2. MOKAIs

We call *MOKAI* the vehicle that the human will be operating. When used, the *MOKAI* does not need to have visual of the map. To communicate with the *shoreside* and other vehicles, either other humans or robotic teammates, an link is required. The link is either over a local network, when running on simulation, or RF when running on different platforms.

Like the *shoreside*, *MOKAI* has a script that automates the configuration and launches the required applications located under the directory:

```
cd mokai/
```

```
./launch_mokai.sh [SWITCHES]
--blue, -b        : Blue team
--red, -r         : Red team
--w-evan, -e      : Evan as a teammate.
--w-felix, -f     : Felix as a teammate.
--w-gus, -g       : Gus as a teammate.
--semi-sim, -ss   : Semi-autonomous simulation (w/ joysticks)
--sim, -s         : Full simulation
--voice-on, -von  : Voice recognition on
--voice-off, -voff: Voice recognition off
--just_build, -j
--help, -h
```

2.2.1. Switches:

1. *-b* and *-r* switches allow you to select a team for the *MOKAI*, either blue or red resp.
2. *-e*, *-f*, and *-g*[4] switches are used to select a robotic teammate. This required to properly set up the voice commands that will be processed by the voice recognition software and later forwarded to the selected teammate.
3. *-ss* switch is used to simulate the vehicle but keep the control to a human operator through a joystick input.
4. -s switch is used to simulate the vehicle fully, behavior and controls, and does not require any human input or joystick.
5. *-von* and *-voff* switches resp. enable and disable the speech recognition and microphone input. When the later is used, a terminal window shows up, Figure 2 on page 5, and will simulate the output from the voice recognition application.

2.2.2. Examples

Simulation. To launch a simulated *MOKAI* using a joystick and voice recognition off and be on the red team with Gus as a teammate:

```
./launch_mokai.sh -ss -voff -r -g
```

Real vehicle. To launch on a real *MOKAI* using voice recognition on and be on the blue team with Evan as a teammate:

```
./launch_mokai.sh -von -b -e
```

2.3. M200s

We call *M200* the autonomous vehicle that will be teammate with the human. To communicate with the *shoreside* and other vehicles, either other robots or human teammates, an data link is required. The data is either shared over

[4] Evan, Felix, and Gus are the names of the vehicles that are being used. Each one has its own static IP address and this is how the script makes the difference between the vehicles.

```
Cue                 VarName                      VarValue
-----------         ---------------              --------------
no                  SPEECH_RECOGNITION_SENTENCE  "NO"
yes                 SPEECH_RECOGNITION_SENTENCE  "YES"
grab                SPEECH_RECOGNITION_SENTENCE  "GRAB"
tag                 SPEECH_RECOGNITION_SENTENCE  "TAG"
evan_attack         SPEECH_RECOGNITION_SENTENCE  "evan ATTACK"
evan_defend         SPEECH_RECOGNITION_SENTENCE  "evan DEFEND"
evan_intercept      SPEECH_RECOGNITION_SENTENCE  "evan INTERCEPT"
evan_cover          SPEECH_RECOGNITION_SENTENCE  "evan COVER"
evan_station        SPEECH_RECOGNITION_SENTENCE  "evan STATION"
evan_follow         SPEECH_RECOGNITION_SENTENCE  "evan FOLLOW"
evan_return         SPEECH_RECOGNITION_SENTENCE  "evan RETURN"
evan_deploy         SPEECH_RECOGNITION_SENTENCE  "evan DEPLOY"

>
```

Figure 2: Text input to simulate voice recognition.

a local network, when running on simulation, or RF when running on different platforms.

Like the *shoreside*, *M200* has a script that automates the configuration and launches the required applications located under the directory:

```
cd m200/
```

```
./launch_m200.sh [SWITCHES]
  --evan, -e      : Evan vehicle only.
  --felix, -f     : Felix vehicle only.
  --gus, -g       : Gus vehicle only.
  --blue, -b      : Blue team.
  --red, -r       : Red team.
  --sim, -s       : Simulation mode.
  --start-x=      : Start from x position (requires x y a).
  --start-y=      : Start from y position (requires x y a).
  --start-a=      : Start from angle (requires x y a).
  --just_build, -j
  --help, -h
```

2.3.1. Switches:

1. *-b* and *-r* switches allow you to select a team for the *M200*, either blue or red resp.
2. *-e*, *-f*, and *-g* switches are used to select the robot. This required to properly set up the interface between the payload computer and the front-seat computer to drive the vehicle.
3. *-s* switch is used to simulate the robot, therefore disabling the interface to the front-seat computer.
4. *–start-x=*, *–start-y=*, and *–start-a=* are used to specify starting point different from the default one.

2.3.2. Examples

Simulation. To launch a simulated *M200* and have it be on the red team with Gus front-seat computer:

```
./launch_m200.sh -s -r -g
```

Real vehicle. To launch on a real *M200* and have it be on the blue team with Evan front-seat computer:

```
./launch_m200.sh -b -e
```

3. Advanced usage

In this section is explained an advanced method to use all the above scripts using only one command that will connect, configure and launch the relevant applications all in parallel in both the *shoreside*, the *MOKAI*s, and the *M200*s.

As introduced in 1.2.2, *fabric* is a tool that enables automation of scripts and commands over SSH to distant targets. A *fabric* script has been written to update, build, and launch the scripts in all operating vehicles in parallel. It is located in:

```
aquaticus1.0/fabfile.py
```

As the script is configured to be always run in parallel[5], it is highly recommended to have SSH keys already deployed on the targets.

To list all the commands that are defined in the script, the following command is entered at the same location as the *fabfile.py* script:

```
fab -l
```

3.1. Example usage

Updating the trees. For instance, to get the latest version of both *moos-ivp* and *moos-ivp-aquaticus* and build both trees, two options are selected that will be executed sequentially in order but parallel on the targets:

```
fab update_all build_all
```

Running a mission. To run a mission on real vehicles, it is possible to launch the all the targets using one simple command:

```
fab launch_all
```

AppendixA. ./launch_simulation.sh

Nothing to see for now. Coming soon.

AppendixB. fabric.py

Nothing to see for now. Coming soon.

[5] This option can disabled by changing the variable *env.parallel=False* in the script.