A mini HOWTO on AUV simulation

Frédéric Py
fpy@mbari.org

March 28, 2007

1 Some notes about this document

This document describes shortly how to use and install the AUV software architecture under QNX. It is only based on my experience and may not apply to everybody. Feel free to send me comment or corrections.

We will consider here that we try to install the software on a QNX platform already installed and configured named `foobar` and having the `dorado1` user account.

2 Preparing the system

The first thing to do is to modify your `~/.profile` file (on QNX) to define some environment variables used by AUV software. You'll need to add the following lines :

```
export AUV=$HOME/auv/altex/onboard
export AUV_PLAN_DIR=$AUV/missions
export AUV_LOG_DIR=$AUV/logs

# Configurations files for dorado389
export AUV_CONFIG_DIR=$AUV/config/dorado389

export PATH=$AUV/bin:$PATH:$HOME/bin
```

You probably want to source this file if you don't want to logout (it will be loaded each time you login) :

```
$ . ~/.profile
```

Install the `auv` directory on your home directory (`/home/dorado1`). You then need to install the `gctpc` library which ids needed to build the AUV architecture. This package can be downloaded at the following address :

- http://edcftp.cr.usgs.gov/pub/software/gctpc/gctpc.tar.Z

To follow the structure of real AUVs, do the following (some commands may be useless):

```
$ su
password: ********
# mkdir /usr/local/src
# cd /usr/local/src
# gunzip -c /home/dorado1/gctpc.tar.Z | tar xvf -
# cd gctpc/src
# make
```

```
# mkdir /usr/local/lib/gctpc
# cp geolib.a /usr/local/lib/gctpc
# ^D
$
```

During the `make` You may see an error like :

```
make: ranlib: Command not found
make: *** [geolib.a] Error 127
```

This error is not important here and you can simply ignore it.

3 Installing the AUV architecture

To compile the AUV architecture you need first to compile the tools used by it (idlParser and makedepend). To do so do the following :

```
$ cd ~/auv/altex/makedepend
$ make
$ cp makedepend ~/bin
$ cd ~/auv/idl
$ make
$ cp idlParser idlToC++ ~/bin
```

You are now ready to compile the source tree for the AUV :

```
$ cd $AUV
$ make
```

Ther may be errors during compilation try to guess their origins (in particular `makedepend` may crash if you pass him as argument a directory that does not exist and fails if the number of include directories is greater than 10, to check it look at the `INCLUDE` variable on the Makefile which have failed.

If you have finally no error message then you can finish the installation using the command :

```
$ make install
```

You may alsoo check if the directory `latest` exists in the AUV logs directory.

```
$ cd $AUV_LOG_DIR
$ mkdir latest
```

4 Preparing the mission

Some configurations files you need may not be installed. In particular the files needed for the communication with the AMC need to be copied into the configuration directory :

```
$ cd $AUV_CONFIG_DIR
$ cp $AUV/AmcAgent/amcAgent.cfg .
$ cp $AUV/StatePublisher/statePublisher.cfg .
```

The content of `amcAgent.cfg` may be the following :

```
PORT = 8005;
```

where 8005 is the port number used by the socket.

The content of `statePublisher.cfg` looks like this :

```
PublishPeriod = 500;
AMC_PORT = 8002;
AMC_IP = 134.89.12.190;
```

Where `PublishPeriod` is the periodicity to send updates to the AMC, `AMC_PORT` is the port number of the socket used to send updates and `AMC_IP` has to be changed to the IP address wher the AMC agent will be located.

5 Launching the simulation

To launch the simulation you'll need to first check if `Mqueue` is running :

```
$ ps -elf | grep Mqueue
```

If you don't have a line similar to :

```
  148    93 24o  RECV     0    00:01:46 (//1/bin/Mqueue)
```

you need to start `Mqueue` as root :

```
$ su
password: ********
# Mqueue &
# ^D
```

Then you can launch the simulation with this simple command :

```
$ cd $AUV/bin
$ amcAgent -v -sim
```

Welcome to the real world

If you need to launch the architecture on a real AUV you do almost the same thing except for the last command :

```
$ amcAgent
```

Indeed , the meaning o `amcAgent` arguments is the following :

`-v` run in verbose mode.

`-sim` run with the simulation engine.