iBellhop / lib_bellhop Quick Start

T. Schneider tes@mit.edu
Laboratory for Autonomous Marine Sensing Systems
MIT / WHOI Joint Program in Oceanography & Ocean Engineering

This document is available at http://gobysoft.com/dl/iBellhop_quick_start.pdf.

1 BELLHOP

Michael Porter's ray tracing code BELLHOP can be downloaded (with some documentation) as part of the Acoustics Toolbox (AT) from http://oalib.hlsresearch.com/Rays/index.html. I have also mirrored a reasonably new copy of BELLHOP in the moos-ivp-local repository in `moos-ivp-local/src/thirdparty/at`. Most of AT is built as part of the normal CMake build system in moos-ivp-local and placed in moos-ivp-local/bin.

2 Google Protocol Buffers TextFormat representation of the ENV file

BELLHOP reads in an ENVFIL (read "environmental file") which is now a file ending in .env with a particularly impossible to decipher format. Basically the file contains parameters about the modeled environment (parameters for the surface, water column, bottom), acoustics (sources and receivers) and rays specifics (number of beams, tracing box, beam angles). Thus, I have reinterpreted the environmental file as a Google Protocol Buffers (protobuf: http://code.google.com/p/protobuf/) object (bellhop::protobuf::Environment) containing other objects (Surface, Bottom, Water Column, etc.) which have various parameters and values. See `moos-ivp-local/src-shared/tes/lib_bellhop/environment.proto` for the protobuf representation.

To represent the Environment object on disk as a file, I use the TextFormat human readable serialization of the bellhop::protobuf::Environment message. This "protobuf configuration" file (*.pb.cfg) is a mostly complete representation of the parameters in the ENVFIL. Thus, the simplest thing you can do with lib_bellhop is read an Environment .pb.cfg file and write an ENVFIL that can be used to run BELLHOP on. This is useful in its own right since ENVFILs are so hard to read (and thus generate). The tool *protobuf2bellhop_env* (in moos-ivp-local) does precisely this:

```
cd moos-ivp-local/src-shared/tes/iBellhop
protobuf2bellhop_env example.pb.cfg > example.env
bellhop.exe example
```

protobuf2bellhop_env reads in example.pb.cfg, which at the time of this writing contained:

```
title: "DSOP Base TL"            # Title of this environmental
                                 # file for your use (opt)
                                 # (default="Environmental File")
freq: 500  # Hertz (opt) (default=25000)
output {  # Parameters governing the output of BELLHOP (opt)
    type: INCOHERENT_PRESSURE  # Type of ray trace to perform
                                 # and file to output(ARR, RAY, or
                                 # SHD) (ARRIVAL_TIMES, EIGENRAYS,
                                 # RAYS, COHERENT_PRESSURE,
                                 # INCOHERENT_PRESSURE,
                                 # SEMICOHERENT_PRESSURE) (opt)
                                 # (default=INCOHERENT_PRESSURE)
}
surface {  # Parameters that model the sea surface (opt)
    medium {  #  (opt)
        type: VACUUM   # The type of the medium (VACUUM, RIGID,
                # HALF_SPACE) (opt)
    }
}
water_column {  # Parameters that model the water column (opt)
    interpolation_type: CUBIC_SPLINE  # The method used to
                                        # interpolate the discrete
                                        # sound speed profile (SSP)
                                        # points (CUBIC_SPLINE,
                                        # C_LINEAR, N2_LINEAR)
                                        # (opt)
                                        # (default=CUBIC_SPLINE)
    use_attenuation: true  # Enable Thorpe volume attenuation
                             # (opt) (default=true)

    sample { depth: 1.2000000e+03 cp: 1.4830304e+03 }
    sample { depth: 1.3000000e+03 cp: 1.4834762e+03 }
    sample { depth: 3.0000000e+01 cp: 1.5380951e+03 }
    sample { depth: 5.0000000e+01 cp: 1.5383277e+03 }
    sample { depth: 7.5000000e+01 cp: 1.5384324e+03 }
    sample { depth: 1.0000000e+02 cp: 1.5378874e+03 }
    sample { depth: 1.2500000e+02 cp: 1.5355741e+03 }
    sample { depth: 1.5000000e+02 cp: 1.5321705e+03 }
    sample { depth: 2.0000000e+02 cp: 1.5243158e+03 }
    sample { depth: 2.5000000e+02 cp: 1.5170844e+03 }
    sample { depth: 0.0000000e+00 cp: 1.5377889e+03 }
    sample { depth: 1.0000000e+01 cp: 1.5378686e+03 }
    sample { depth: 2.0000000e+01 cp: 1.5379703e+03 }
    sample { depth: 3.0000000e+02 cp: 1.5113237e+03 }
    sample { depth: 4.0000000e+02 cp: 1.4998603e+03 }
    sample { depth: 5.0000000e+02 cp: 1.4908948e+03 }
    sample { depth: 6.0000000e+02 cp: 1.4862527e+03 }
    sample { depth: 7.0000000e+02 cp: 1.4842065e+03 }
```

```
    sample { depth: 8.0000000e+02 cp: 1.4834365e+03 }
    sample { depth: 9.0000000e+02 cp: 1.4826757e+03 }
    sample { depth: 1.0000000e+03 cp: 1.4825752e+03 }
    sample { depth: 1.1000000e+03 cp: 1.4826209e+03 }
    sample { depth: 1.4000000e+03 cp: 1.4843964e+03 }
    sample { depth: 1.5000000e+03 cp: 1.4852395e+03 }
    sample { depth: 1.7500000e+03 cp: 1.4881166e+03 }
    sample { depth: 2.0000000e+03 cp: 1.4910494e+03 }
    sample { depth: 2.5000000e+03 cp: 1.4981750e+03 }
    sample { depth: 3.0000000e+03 cp: 1.5062671e+03 }
    sample { depth: 3.5000000e+03 cp: 1.5147061e+03 }
    sample { depth: 4.0000000e+03 cp: 1.5234204e+03 }
    sample { depth: 4.5000000e+03 cp: 1.5323953e+03 }
    sample { depth: 5.0000000e+03 cp: 1.5416240e+03 }
    sample { depth: 5.5000000e+03 cp: 1.5508048e+03 }
}
bottom {  # Parameters that model the sea floor (opt)
    medium {  #  (opt)
        type: HALF_SPACE  # The type of the medium (VACUUM, RIGID,
                # HALF_SPACE) (opt)
        attenuation {  # Attenuation parameters of the medium;
                       # only used for type == HALF_SPACE (opt)
            units: DB_PER_WAVELENGTH
                   # Must use same units for surface and bottom
                   # attenuation (DB_PER_M_KHZ, PARAMETER_LOSS,
                   # DB_PER_M, NEPERS_PER_M, Q_FACTOR,
                   # DB_PER_WAVELENGTH) (opt)
            value: 0.5  #  (opt)
        }
        cp: 1700   # The compressional speed (m/s) of the medium; only
              # used for type == HALF_SPACE (opt)
        cs: 200  # The shear speed (m/s) of the medium; only used
              # for type == HALF_SPACE (opt)
        density: 2.0  # Density in g/cm^3; only used for type ==
                   # HALF_SPACE (opt)
        depth: 5500  # The depth of this medium interface (0 =
                 # surface) (opt)
    }
}
sources {  # Parameters that determine acoustic "source(s)"
               # in the ray trace (opt)
    number_in_depth: 1  # The number of sources vertically;
                        # sources are all assumed to be at range
                        # == 0 (opt)
    first {  # The position of the shallowest source (opt)
        depth: 200  #  (opt)
    }
}
receivers {  # Parameters that determine the acoustic
                 # "receiver(s)" in the ray trace (opt)
    number_in_depth: 301   # The number of receivers vertically
```

```
                        # (opt)
    number_in_range: 1000  # The number of receivers horizontally
                        # (opt)
    first {  # The position of the shallowest source at the
               # closest range (opt)
        depth: 2000  #  (opt)
        range: 0  #  (opt)
    }
    last {  # The position of the deepest source at the
              # furthest range (opt)
        depth: 5000  #  (opt)
        range: 50000  #  (opt)
    }
}
beams {  # Parameters that govern the beams used in the ray
             # trace (opt)
    approximation_type: GAUSSIAN  # The method used to approximate the
                            # finite element shape used in the ray
                            # trace (GEOMETRIC, CARTESIAN,
                            # RAY_CENTERED, GAUSSIAN) (opt)
    theta_min: -60  # The smallest ray angle used in the ray
                      # trace. theta=0 is horizontal. (opt)
                      # (default=-90)
    theta_max: 60  # The largest ray angle. theta=0 is
                     # horizontal (opt) (default=90)
    number: 1000  # The number of rays used (opt) (default=0)
}
adaptive_info {  # Not used in the BELLHOP ENV file, but used
                    # by iBellhop to adaptively populate certain
                    # fields of this message (opt)
    contact: "TGT_1"  # The name of a contact (NAME= field in
                   # NODE_REPORT) used to find  the `source`
                   # position in the ray trace. (opt)
    # ownship: ""  # The name of our vehicle (NAME= field in
                   # NODE_REPORT). If omitted, the Community name
                   # is used. This is used to find the `receiver`
                   # position(s) in the ray trace (opt)
    read_shd: false  # Read the generated SHD file (if
                       # applicable) and parse it into a list of TL
                       # values. Currently used for the
                       # BHV_AcommsDepth (opt) (default=false)
    auto_receiver_ranges: true  # If true, the
                                  # `receiver.first.range` and
                                  # `receiver.last.range` will be
                                  # set automatically based on the
                                  # current trajectory of the
                                  # `ownship` and `contact` based
                                  # on the lastest NODE_REPORT.
                                  # The calculated window will be
                                  # the predicted position from
                                  # now until `look_ahead_seconds`
```

```
                                    # in the future. (opt)
                                    # (default=false)
    look_ahead_seconds: 30  # The number of seconds to look
                              # ahead for when calculated receiver
                              # ranges using `auto_receiver_ranges`
                              # (opt) (default=60)
    auto_source_depth: true  # If true, set
                               # `source.first.depth` based on the
                               # position of `contact` (opt)
                               # (default=false)
}
```

and writes the equivalent example.env that can be read by BELLHOP (bellhop.exe):

```
'DSOP Base TL'
500
1
'SVWT'
0 0 5500
0 1537.79 /
10 1537.87 /
20 1537.97 /
30 1538.1 /
50 1538.33 /
75 1538.43 /
100 1537.89 /
125 1535.57 /
150 1532.17 /
200 1524.32 /
250 1517.08 /
300 1511.32 /
400 1499.86 /
500 1490.89 /
600 1486.25 /
700 1484.21 /
800 1483.44 /
900 1482.68 /
1000 1482.58 /
1100 1482.62 /
1200 1483.03 /
1300 1483.48 /
1400 1484.4 /
1500 1485.24 /
1750 1488.12 /
2000 1491.05 /
2500 1498.17 /
3000 1506.27 /
3500 1514.71 /
4000 1523.42 /
4500 1532.4 /
5000 1541.62 /
```

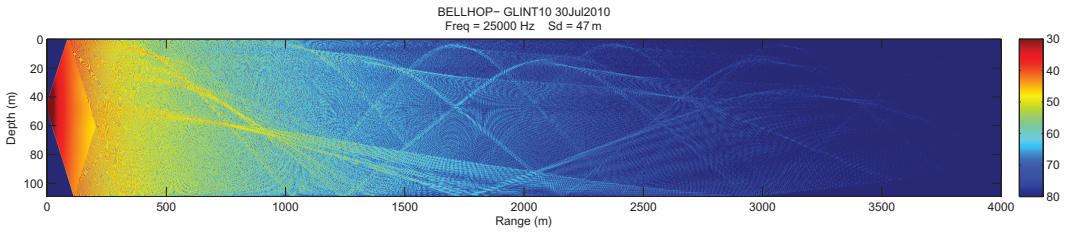

Figure 1: Example SHDFIL (coherent transmission loss) plot (generated from the Protobuf (.pb.cfg) Environmental file passed through lib_bellhop to generate an ENVFIL that was given to BELLHOP.)

```
5500 1550.8 /
'A' 0
5500 1700 200 2 0.5 /
1
200 /
301
2000 5000 /
1000
0 50 /
'IB'
1000
-60 60 /
0 6050 55
```

Because we gave

```
output {
    type: INCOHERENT_PRESSURE
}
```

BELLHOP writes a SHDFIL (shade file) called example.shd. We could plot (see Fig. 1) this in MATLAB by running `plotshd` (add `moos-ivp-local/src/thirdparty/at/Matlab` and subdirectories to the MATLAB path). You can also convert it to ASCII using `toasc.exe`, but you must rename `example.shd` to `SHDFIL` first:

```
cp example.shd SHDFIL
toasc.exe
mv ASCFIL example.asc
```

The resulting example.asc can be read easily by any common programming language. The format is pretty easy to figure out by looking at the MATLAB tools in `moos-ivp-local/src/thirdparty/at/Matlab`.

3 iBellhop

iBellhop takes advantage of the Protocol Buffers C++ representation of the ENVFIL to continuously generate up-to-date ENVFILs during the mission and thus adaptively model

acoustics of interest (targets, acomms, etc.) iBellhop starts each mission with a "background" environment given by a .pb.cfg Environment file. This is usually generated by the "best available" sound speed profile combined with a number of more or less constants (ray tracing parameters, bottom parameters, etc.).

3.1 MOOS File

iBellhop's MOOS file is also a protobuf message (see `moos-ivp-local/src-shared/tes/lib_bellhop/iBellhop_messages.proto`), that primarily contains a bellhop::protobuf::Environment. In `missions-lamss`, we use an #include to grab the appropriate .pb.cfg for a given cruise:

```
// missions-lamss/global_plugs/iBellhop.plug
ProcessConfig = iBellhop
{
  common {
    verbosity: VERBOSITY_GUI  # Verbosity of the terminal
  }

  initial_env {
#include $(MISSION_ROOT)/cruise/current/data/bellhop_env.pb.cfg
  }

  moos_var_request: "BELLHOP_REQUEST"
  moos_var_response: "BELLHOP_RESPONSE"
  output_env_dir: "$(SCRATCH)"
}
```

where `bellhop_env.pb.cfg` is a symbolic link to the currently used pb.cfg background environment. If in doubt of the format of the iBellhop ProcessConfig block (and by extension, the embedded .pb.cfg file), you can always type

```
iBellhop --example_config
```

3.2 Structure

iBellhop is structured as a MOOS Application interface to a C++ library lib_bellhop that handles the object representation of the ENVFIL. Interaction with BELLHOP is done at the filesystem level due to the difficulty of integrating direct calls with legacy Fortran code. Of course, if disk i/o speeds are a concern, one can use a shared memory filesystem (such as /dev/shm provided in Linux). Fig. 2 diagrams this structure using UML.

3.3 Dynamic Runtime Functionality

iBellhop currently supports these dynamic functionalities (diagrammed in Fig. 3):

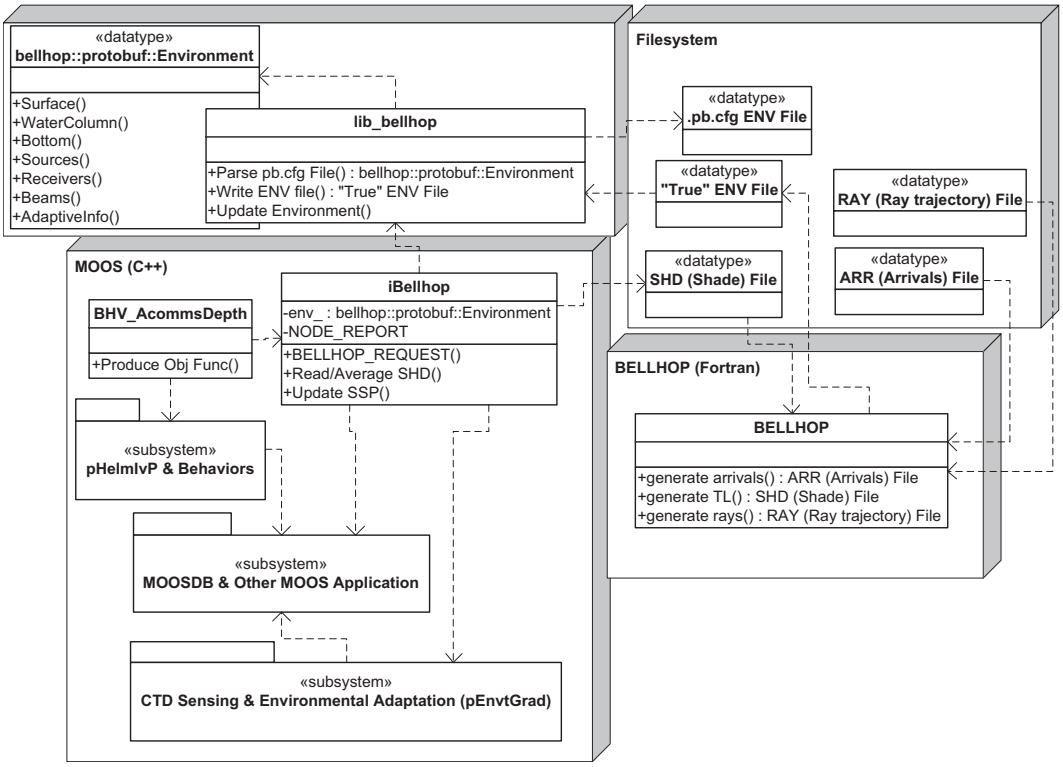

Figure 2: UML Structure diagram of iBellhop and its relation with MOOS, lib_bellhop and BELLHOP

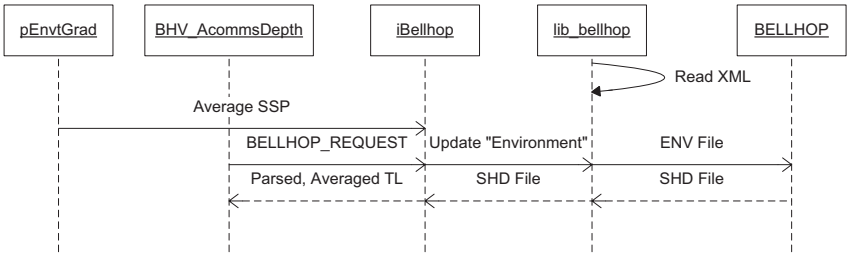

Figure 3: UML Sequence diagram of iBellhop.

- Updates to the sound speed profile from the MOOS message `AVG_SS_VEC` which has the format: "`depth=10,20,30,40,50,c=1510,1505,1500,1500,1500`". `depth` is a list of depths in meters, `c` is the speed of sound at that depth in meters per second. Any previously specified speed values at the depths given are replaced by this update. Depths not given are not changed (if present from the XML file or prior updates via `AVG_SS_VEC`). To disable this feature, do not publish `AVG_SS_VEC` (currently pEnvtGrad publishes `AVG_SS_VEC`).

- Requests for a ray trace via `BELLHOP_REQUEST` (default, a different name can be set in .moos file). `BELLHOP_REQUEST` can contain any valid field that can be given in the .pb.cfg file (with the same syntax). Any parameters that are not given use the XML file's values. In other words, `BELLHOP_REQUEST` overwrites (for that request only) the values given in the .pb.cfg file at launch. The format of `BELLHOP_REQUEST`[1] serialized to a string (for MOOS) looks like:

```
env {
 adaptive_info {
   contact: "TGT_1"
   look_ahead_seconds: 120
   read_shd: true
   auto_receiver_ranges: true
   auto_source_depth: true
  }
  output { type: INCOHERENT_PRESSURE }
}
```

  You can either generate such a string manually, or you can use Google Protocol Buffers and Goby like so:

```
// from moos-ivp-local/src-lamss/tes/lib_behaviors-tes/BHV_AcommsDepth.cpp
#include "goby/moos/libmoos_util/moos_protobuf_helpers.h"
#include "iBellhop_messages.pb.h"

iBellhopRequest request;
bellhop::protobuf::Environment::AdaptiveInfo* adaptive_info =
   request.mutable_env()->mutable_adaptive_info();

adaptive_info->set_contact(collaborator_);
adaptive_info->set_look_ahead_seconds(look_ahead_seconds_);
adaptive_info->set_read_shd(true);
adaptive_info->set_auto_receiver_ranges(true);
adaptive_info->set_auto_source_depth(true);
```

[1] BELLHOP_REQUEST is a TextFormat serialized version of the Google Protocol Buffers Message iBellhopRequest; see `moos-ivp-local/src-shared/tes/lib_bellhop/iBellhop_messages.proto`

```
request.mutable_env()->mutable_output()->set_type(
  bellhop::protobuf::Environment::Output::INCOHERENT_PRESSURE);

std::string serialized;
serialize_for_moos(&serialized, request);

// same as m_Comms.Notify(...)
postMessage("BELLHOP_REQUEST", serialized, "repeatable");
```

You will need to link against two libraries (goby_protobuf, bellhop_tools):

```
// from moos-ivp-local/src-lamss/tes/lib_behaviors-tes/CMakeLists.txt
target_link_libraries(BHV_AcommsDepth
    ...  goby_protobuf bellhop_tools)
```

Following this request, an ENVFIL is written to the directory given by the ProcessConfig parameter `output_env_path`, BELLHOP is run. Finally the generated file name is posted as `{contact}_SHD_FILE` for env.output.type=*_PRESSURE, `{contact}_ARR_FILE` for env.output.type=ARRIVAL_TIMES, `{contact_RAY_FILE` for env.output.type=EIGENRAYS or env.output.type=RAYS. iBellhop also writes the MOOS variable `BELLHOP_RESPONSE`[2] which contains the complete (merged) pb.cfg ENV file for the run, the success or failure of the request, and if env.adaptive_info.read_shd is true, the vector of averaged TL samples. To read the `BELLHOP_RESPONSE`, it is easiest to use code like this:

```
// from moos-ivp-local/src-lamss/tes/lib_behaviors-tes/BHV_AcommsDepth.cpp
#include "goby/moos/libmoos_util/moos_protobuf_helpers.h"
#include "iBellhop_messages.pb.h"
std::string bellhop_in = getBufferStringVal("BELLHOP_RESPONSE", ok);
iBellhopResponse response;
parse_for_moos(bellhop_in, &response);

bool ok = (response.requestor() == "pHelmIvP") &&
          (response.has_avg_tl());
if(!ok) return 0;

std::vector<double> dom, range;

double max_range = 0;
for(int i = 0, n = response.avg_tl().sample_size(); i < n; ++i)
{
   dom.push_back(response.avg_tl().sample(i).depth());
```

[2] BELLHOP_RESPONSE is a TextFormat serialized version of the Google Protocol Buffers Message iBellhopResponse; see `moos-ivp-local/src-shared/tes/lib_bellhop/iBellhop_messages.proto`

```
    double tl = response.avg_tl().sample(i).tl();
   if(tl > max_range && tl != std::numeric_limits<double>::infinity())
       max_range = tl;

    range.push_back(tl);
}
```

4 BHV_AcommsDepth

BHV_AcommsDepth gives a request for an incoherent pressure (transmission loss) calculation for a narrow slice of the area in range where the vehicle expects to be in the next env.adaptive_info.look_ahead_seconds relative to the vehicle it needs to communicate to env.adaptive_info.contact. Based on this transmission loss calculation (made from the return .shd file), the vehicle gives an objective function over depth (where the lowest TL is given the highest utility weight). The objective function is an average over the area (in range) calculated for the next env.adaptive_info.look_ahead_seconds. The features in adaptive_info can grow as we find more ways we want iBellhop to adapt.

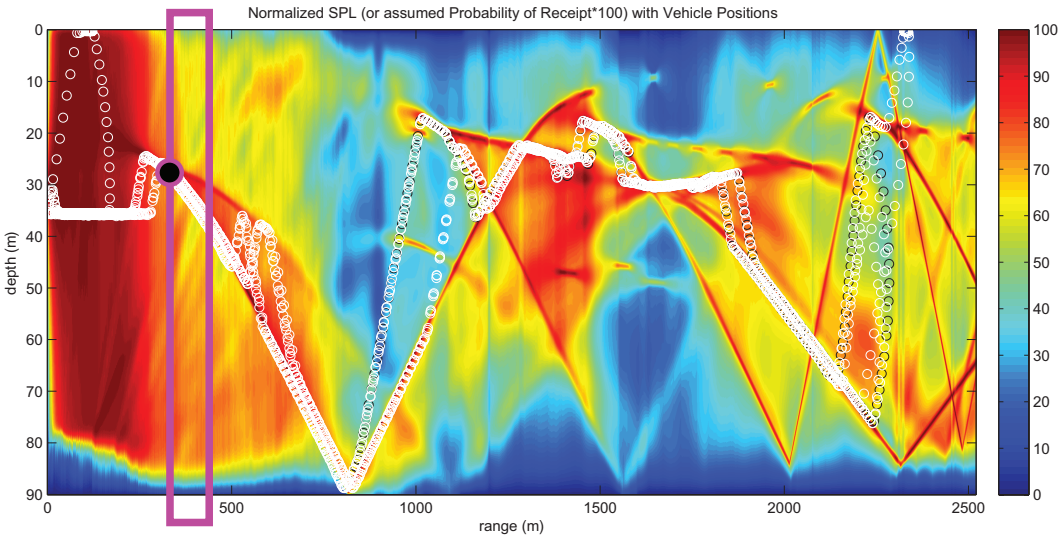

Figure 4: 'Side view" of the AUV during an simple test of an adaptive acoustic communications simulation in shallow water. The circles represent the position of the AUV, and the underlying color plot is a range normalized sound pressure level (SPL) plot. Red regions should have high received signal strength (and by the assumption here, high probability of successful communications) and blue regions have low signal strength. Thus, we expect the AUV to seek regions of high (red) signal strength. Environment from thermistor data taken at 31-May-2009 16:10:24 UTC at 42.5922 ° N, 10.1161 ° E. (The surfacing is caused by the vehicle getting an updated GPS fix). During the mission the entire ray trace shown is not calculated at each update interval (10 seconds), but rather a box of the next expected positions. For example, if the vehicle is at the magenta circle and is moving away from its communicating partner, the magenta box is the region actually calculated by iBellhop / BELLHOP.