Lab #15
Behavior Construction for Autonomous Front Estimation

2.S998 Unmanned Marine Vehicle Autonomy, Sensing and Communications

Contents

1 Overview and Objectives

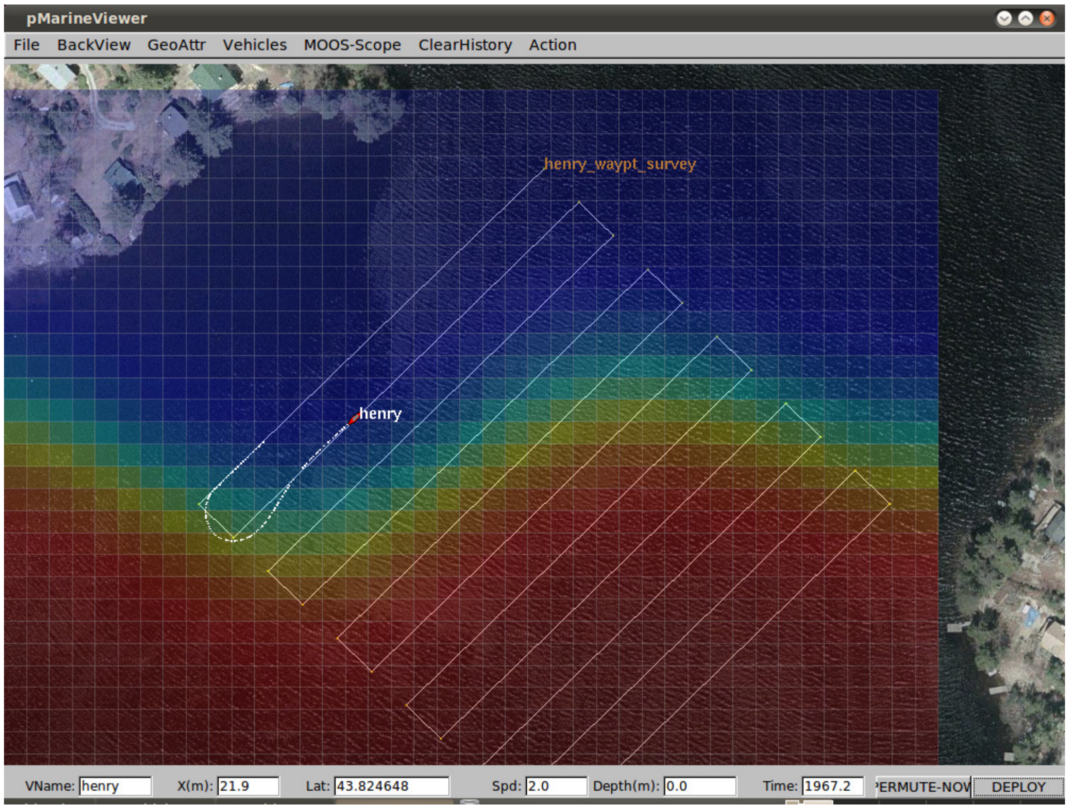

pMarineViewer
File BackView GeoAttr Vehicles MOOS-Scope ClearHistory Action
henry_waypt_survey
henry
VName: henry X(m): 21.9 Lat: 43.824648 Spd: 2.0 Depth(m): 0.0 Time: 1967.2 PERMUTE-NOW DEPLOY

Figure 1: **Ocean Front Estimation:** An autonomous vehicle with a temperature sensor is used to estimate the dynamic properties of an ocean front, separating two water bodies of difdferent temperatures, indicated by the color shading. The front is dynamic, and the objective is to estimate the parameters in an analytical model of the temperature versus time and location

In the previous labs we have focused on remote sensing techniques for detecting, classifying and locating (DLT) seabed hazards. Another common application of autonomous vehicles involves the measurement of oceanographic properties using physical, chemical or biological sensors mounted on the vehicles. For this problem the sensors provide point measurements at the locaion of the AUV. A common problem in coastal oceanography is the spatial and temporal characterization of a front separating water masses of different temperature, as shown in Fig. 1. Only in a few pathological cases the front will be stationary in space and time, and in general the measurements will have to support the estimation of the dynamic properties of the front. This problem is complicated by the *space-time aliasing* inherent to the sampling of a dynamic field by a point sensor with finite mobility. In other word, an AUV with a temperature sensor does not provide a *synoptic* measurement of the temperature field, and a change observed between two measurements may not be directly assocoiated with a spatial or a temporal change.

To break this ambiguity one must in general apply a parametric model of the oceanographic phenomenon, and then design a survey which provides a reliable estimate of the parameters. This approach first of all requires that the model makes sense physically, and the most robust approach is to choose a model which is consistent with the physics of the problem. A suite of such oceanographic models exist, but their use is beyond the scope of this class. Instead we will use an analytical model for the temperature field, but the principle of the oceanographic sampling problem are the same. Thus the survey selected for the measurements by the AUV must be chosen in such a way that the measurements provide an estimate which is unambiguous in space and time. This in general is a complex problem with no unique solution, but some general guidelines exist for the survey:

- The survey must focus on the areas that provide the most information about the model parameters, in general translating into a strategy of concentrating the survey in areas of strong gradients.
- While focusing most effort in areas of strong variability, the geographical region of interest must be adequately covered.
- The survey should be designed to optimally uncouple the parameters, i.e. avoid patterns which provide information that is ambiguous in the parameters.

1.1 Structure of the Lab and Goals

This lab with stretch over four lab sessions. In the first session, you will use a scripted lawnmower survey to collect the data for the parameter estimation, and use it to develop an understanding of the coupling between the model parameters and the effects of changes in the survey design.

In the next part, the goal will be to write a couple of intermediate simple behaviors. In the final part you will, based on the experience gained in the first parts, design your own IvP behavior for estimating the model parameters in the shortest amount of time.

You will be provided a MOOS module for performing the parameter estimation using simulated annealing, but you have to select the control parameters for the algorithm, most importantly those controlling the 'cooling' schedule. You will be able to modify the front model as you see fit to test your behavior.

During the last lab session, we will run a blind test of your algorithm and behavior, where we control the front simulator and your vehicle will have to perform its survey and determine a parameter estimate. The grading will be based on a score that is inversely proportional to the parameter estimation error and the time for the survey and the parameter estimation.

1.2 Preliminaries

Before jumping into the assignments following, you may want to familiarize yourself with the template configuration. The shoreside and vehicle configuration you should use as template for the frontal estimation problem is available in the class repository:

```
  moos-ivp-12.2mit/ivp/missions/u6_alpha
```

To execute the template survey mission and the shoreside community, use the command:

```
> ./launch.sh --warp=15 --cool=10 --angle=0
```

where the switches represent:

warp Time warp factor as usual. You should be able to do at least 10 here.

cool The cooling factor k applied to the Boltzman probability in the simulated annealing, $\exp(-E/kT)$.

angle The angle of the survey legs. 0 yields vertical survey legs, ± 90 horizontal legs.

2 Mechanics of Writing Behaviors

Before building a the primary behavior for this lab, we will build to intermediate simple behaviors. The first behavior, the "pulse" behavior, simply generates a visual artifact viewable in pMarineViewer to confirm the behavior function. The second behavior, the "zigleg" behavior, builds on the first behavior to periodically influence the heading of the vehicle in a partial zig-zag leg.

In the end your two behaviors should be runnable with output similar to the two clips below on the course wiki page:

```
http://oceanai.mit.edu/2.S998/clips/clip_151_bhv_rpulse.mp4
http://oceanai.mit.edu/2.S998/clips/clip_152_bhv_zleg.mp4
```

2.1 A First Behavior - the Pulse Behavior

Although the primary purpose of writing a behavior is to generate output (IvP functions) to influence the trajectory of a vehicle, our first behavior will simply post a visual artifact to the MOOSDB at certain points in time. The goals of this exercise are to gain familiarity with:

- Generation of a new behavior class
- Adding a behavior to the build system
- Obtaining information from the information buffer into your behavior
- Overloading general behavior functions to suit your needs
- Handling the setting of behavior parameters set by the user

General Description of the Pulse Behavior

The simple idea behind the `Pulse` behavior is for the behavior to post a visual artifact to the MOOSDB five seconds after the behavior notices that vehicle reaches a waypoint achieved by a sister waypoint behavior. The idea is shown in the Figure 2 below.

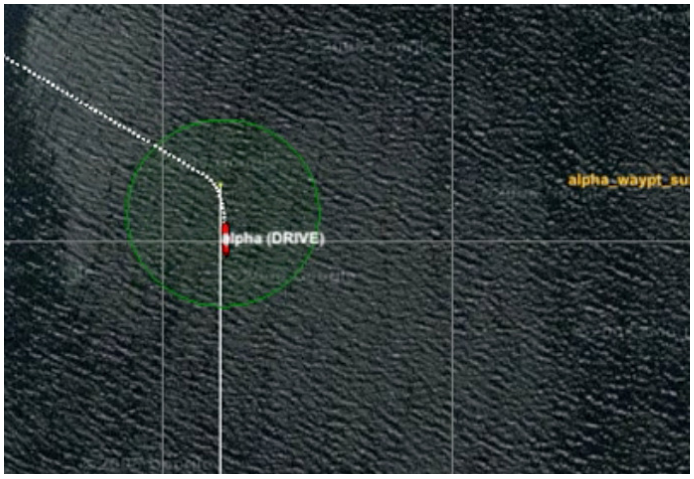


Figure 2: **The Pulse Behavior:** The behavior simply generates a range pulse after it observes that a waypoint has been achieved. The range pulse is a posting to the variable footnotesize RANGE_PULSE.

The Pulse behavior is not configured with waypoints itself, but uses information published by the waypoint behavior to detect a change in waypoint status. Each time the "next" waypoint changes, the behavior posts a range pulse. Nothing more.

2.1.1 Step 1: Make a new behavior that does nothing for now

The first step is to clone the BHV_SimpleWaypoint behavior in your moos-ivp-jsmith tree and use this as a starting point for the Pulse behavior

```
$ cd moos-ivp-jsmith/src/lib_behaviors-test
$ cp BHV_SimpleWaypoint.h   BHV_Pulse.h
$ cp BHV_SimpleWaypoint.cpp BHV_Pulse.cpp
```

Edit these files to change all the class names to BHV_Pulse and feel free to remove any of the simple-waypoint functionality to start with a bare-bones behavior.

Make sure you also edit the boiler-plate code at the bottom of the behavior header file with the line beginning with IVP_EXPORT_FUNCTION.

2.1.2 Step 2: Add the behavior to the build system

To add the behavior to build system, edit the lib_behaviors-test/CMakeLists.txt file. Follow the example for BHV_SimpleWaypoint.

Verify that the new behavior is built after re-building the tree. There should be a dynamic/shared library file in moos-ivp-jsmith/libs after the build.

Verify that the new behavior is runnable with the helm. Make a quick copy of your favorite simple mission like the alpha mission (ivp/mission/s1_alpha in the in the moos-ivp tree), and add this behavior with no configuration parameters. Confirm that the helm launches. The helm will abort if it cannot find this behavior. You can tell the helm about the location of the behavior in one of two ways. First you can use a shell environment variable. Set the following in your .cshrc file:

```
setenv IVP_BEHAVIOR_DIRS /Users/janesmith/moos-ivp-janesmith/lib'
```

or the equivalent in your .bashrc file for bash users. You can also specifiy the behavior location in the pHelmIvP configuration block of your .moos file:

```
ProcessConfig = pHelmIvP
{
  AppTick    = 4
  CommsTick  = 4

  Behaviors  = alder.bhv
  Domain     = course:0:359:360
  Domain     = speed:0:4:21

  IVP_BEHAVIOR_DIR = /Users/janesmith/moos-ivp-janesmith/lib
}
```

2.1.3 Step 3: Edit your behavior to get needed info from the InfoBuffer

There are three key pieces of information your behavior will need to know and update frequently. The vehicle's own x,y position, and the waypoint index to which the waypoint behavior is currently transiting toward. The latter information is published by the waypoint behavior in a simple report of the form:

```
WPT_INDEX = 2
```

To ensure the above MOOS variable is subscribed for by the helm and thus in the helm info buffer, you will need to explicitly declare that your behavior needs it by using the `addInfoVars()` function in the behavior constructor as described in the Lecture 14 notes.

You will need also to get the current time from the information buffer. Use the `getBufferCurrTime()` function as described in the Lecture 14 notes.

You will need the waypoint index to know and note when it has changed. You need the time so you can mark the time when it changed and generate the pulse five seconds later. And you will need the vehicle x,y position for creating the range pulse when the time comes. You may or may not want to store these as member variables of your behavior class. By storing them as member variables, you then have the option of having a separate little class function for updating info from the info buffer on each behavior iteration. This is just a style suggestion.

2.1.4 Step 4: Overloading the onRunState() function

Implement your behavior's `onRunState()` function to produce a range pulse each time the behavior notes that the waypoint index has increment. Remember, the waypoint index is information posted by a separate behavior, the waypoint behavior, in a posting to the MOOSDB. Have your behavior make its posting five seconds after hitting each waypoint.

The range pulse is a particular message that the pMarineViewer application knows how to parse and render. A typical posting looks like:

```
VIEW_RANGE_PULSE  =  x=151.11,y=-43.44,radius=20,duration=4,label=pulse,
                     edge_color=yellow,fill_color=yellow,time=16010.31,edge_size=1
```

(Perhaps try poking the above to the MOOSDB with `uPokeDB` while pMarineViewer is running, to get an idea of how it works and what it looks like.)

The pulse is essentially a ring with expanding radius that expands to a specified value (`radius`), over a specified time (`duration`). It is centered at a given x,y position and specified edge color, fill color, and edge width. In our behavior we want the center to be the vehicle's current position.

It is recommended that a `VIEW_RANGE_PULSE` message be generated by first creating a `XYRangePulse` instance, setting its parameters, and getting a serialized string version, as below:

```
#include "XYRangePulse.h"

XYRangePulse pulse;
pulse.set_x(m_osx);
pulse.set_y(m_osy);
pulse.set_label("bhv_pulse");
pulse.set_rad(m_range);
pulse.set_duration(m_pulse_duration);
pulse.set_time(m_curr_time);
pulse.set_color("edge", "yellow");
pulse.set_color("fill", "yellow");

string spec = pulse.get_spec();
postMessage("VIEW_RANGE_PULSE", spec);
```

2.1.5 Step 5: Handle user behavior preferences

Although not strictly essential in this simple behavior, almost all behaviors need to handle parameter preferences specified by the user in the `.bhv` file. In the `Pulse` behavior, you should handle two particular parameters: the radius of the range pulse and the duration of the range pulse. Your behavior should be able to handle the below configuration (namely lines 5 and 6):

```
1  Behavior=BHV_Pulse
2  {
3    name = pulse
4
5    pulse_range    = 20
6    pulse_duration = 4
7  }
```

To do this, overload the `setParam()` function in your behavior. You should be able to just follow the example of the `BHV_SimpleWaypoint` behavior in your tree.

2.1.6 Step 6: Demonstrating your behavior

To demonstrate your behavior, start with a copy of the Alpha mission in the moos-ivp tree (moos-ivp/ivp/missions/s1_alpha), and just add your behavior to run under the same conditions as the main waypoint behavior. Your end result should look something like:

http://oceanai.mit.edu/2.S998/clips/clip_151_bhv_rpulse.mp4

2.2 A Second Behavior - the ZigLeg Behavior

The `ZigLeg` behavior is an extension of the `Pulse` behavior. Rather than merely post a range pulse after hitting a waypoint, the `ZigLeg` behavior will briefly produce an objective function to influence the vehicle trajectory. Like the `Pulse` behavior, the `ZigLeg` behavior looks for the point in time where a sister waypoint behavior has reached a waypoint. Five seconds after hitting a waypoint, the `ZigLeg` behavior will produce an objective function with a heading preference some number of degrees (by default 45) from the vehicle's present heading. It will do this only for a brief period of time (by default 10 seconds). The idea is shown in the Figure 3 below.

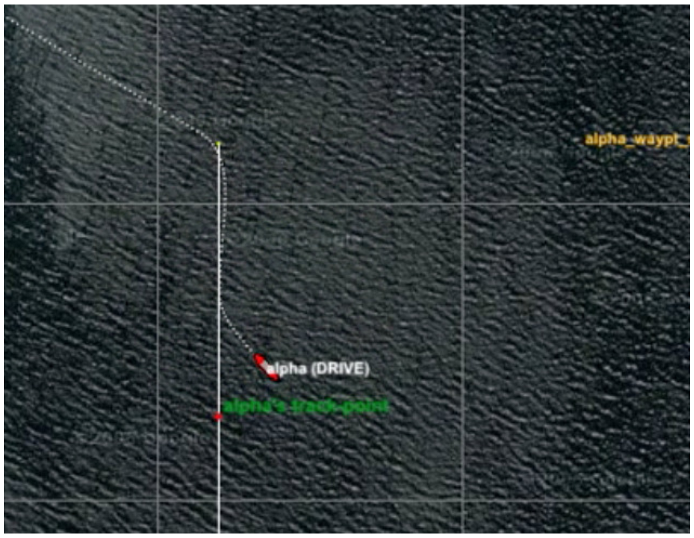


Figure 3: **The ZigLeg Behavior:** The behavior generates a brief heading preference (in the form of an IvP objective function) shortly after noticing the arrival at a waypoint.

As with the `Pulse` behavior, the `ZigLeg` behavior is not configured with waypoints itself, but uses information published by the waypoint behavior to detect a change in waypoint status. Each time the "next" waypoint changes, the behavior posts a range pulse and briefly generates an objective function. The idea can be seen in the below short video clip:

http://oceanai.mit.edu/2.S998/clips/clip_151_bhv_rpulse.mp4

2.2.1 Step 1: Make a new behavior that does nothing for now

Same as with the `Pulse` behavior, Section 2.1.1.

2.2.2 Step 2: Add the behavior to the build system

Same as with the `Pulse` behavior, Section 2.1.2.

2.2.3 Step 3: Edit your behavior to get needed info from the InfoBuffer

Same as with the `Pulse` behavior, Section 2.1.3, except you will also need the vehicle's present heading. When the behavior *does* produce an objective function over heading, it will be as an offset to the present heading.

2.2.4 Step 4: Overloading the onRunState() function

Same as with the `Pulse` behavior, Section 2.1.4, except in this behavior you will additionally produce an objective function over vehicle heading, at the very moment the range pulse is generated. Although the pulse is generated only once, the IvP function will be generated continuously beginning when the pulse is made, continuing for some number of seconds. The number of seconds and the heading offset should by default be 10 seconds and 45 degrees respectively. But these values should be configurable in the behavior file and thus handled in your behavior's `setParam()` function as described below.

Regarding the syntax needed to create an objective function, most likely you will be using the `ZAIC_PEAK` tool to do this. For more information, see the Lecture 14 class notes, and consult the `BHV_SimpleWaypoint` behavior for an example.

Two additional important notes:

- Before returning the objective function in the `onRunState()` function, you will need to set its priority weight. This step is unfortunately missing in the `BHV_SimpleWaypoint` behavior. You can do this with the following two lines:

```
if(ipf)
   ipf->setPWT(m_priority_wt);
```

- Your `ZigLeg` behavior will be running alongside the `Waypoint` behavior. For the `ZigLeg` behavior to exert its influence, you will need to give it a higher priority weight in the `.bhv` configuration file.

2.2.5 Step 5: Handle user behavior preferences

In addition to the two parameters handled in the `Pulse` behavior, your `ZigLeg` behavior should handle two further parameters: the heading offset, and the duration for which the behavior produces the objective function on each waypoint arrival event.

Your behavior should be able to handle the below configuration (namely lines 5-8)

```
Behavior=BHV_Pulse
{
  name = pulse

  pulse_range    = 20
  pulse_duration = 4
  zig_duration   = 10
  zig_angle      = 45
}
```

As before, do this by overloading the `setParam()` function in your behavior.

2.2.6 Step 6: Demonstrating your behavior

To demonstrate your behavior, start with a copy of the Alpha mission in the moos-ivp tree (moos-ivp/ivp/missions/s1_alpha), and just add your behavior to run under the same conditions as the main waypoint behavior. Your end result should look something like:

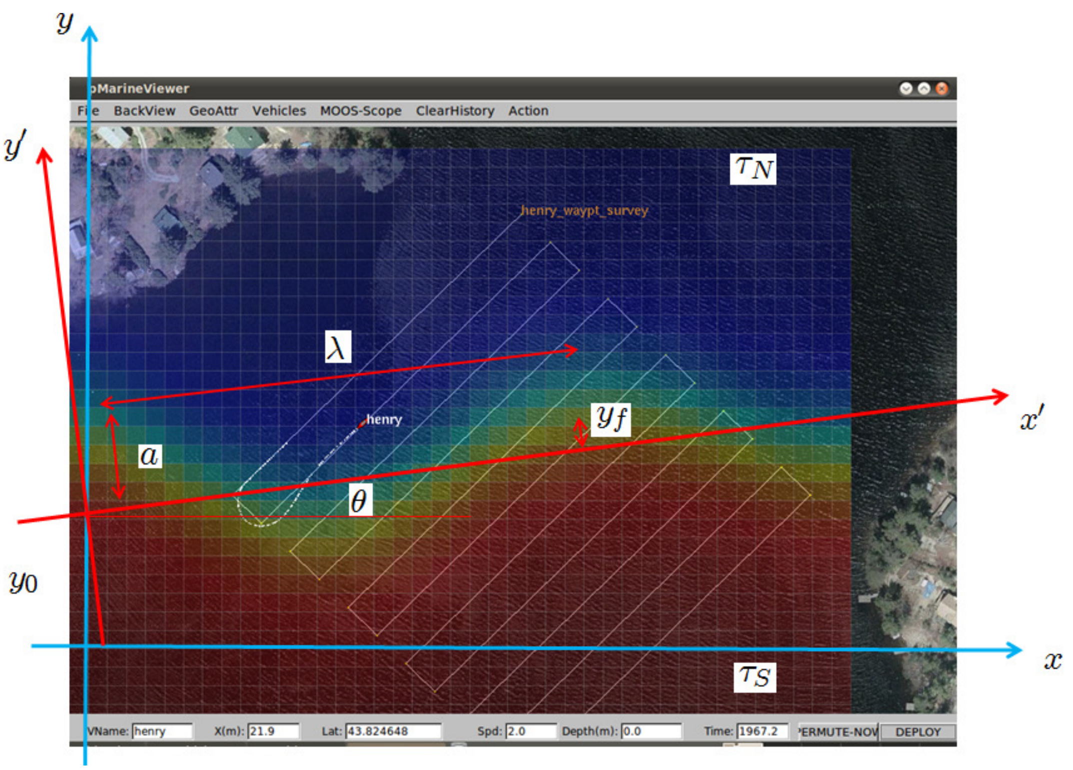
`http://oceanai.mit.edu/2.S998/clips/clip_152_bhv_zleg.mp4`

2.3 Assignment

The assignment for this section is to (a) demonstrate your two working behaviors to a TA or instructor, and (b) demonstrate that your two behaviors accept parameter modifications as described in Sections 2.1.5 and 2.2.5. The due date for this is by the end of lab on the 24th (next Tuesday).

3 Front Estimation

3.1 Front Model



y
y'
pMarineViewer
File BackView GeoAttr Vehicles MOOS-Scope ClearHistory Action
τ_N
henry_waypt_survey
λ
henry
y_f
x'
a
θ
y_0
x
τ_S
VName: henry X(m): 21.9 Lat: 43.824648 Spd: 2.0 Depth(m): 0.0 Time: 1967.2 PERMUTE-NOW DEPLOY

Figure 4: **Ocean Front Estimation:** The temperature field is parameterized by an analytical model with 9 parameters. In addition to the geometrical parameters shown here, the parameters include the temporal period T, the length scale of the frontal gradient, β, and the propagation decay α

The temperature field is modeled by an analytical function, defined as follows. The coordinate system aligned with the front, shown in Fig. 4 is defines as

$$\begin{aligned} x' &= x\cos\theta + (y - y_0)\sin\theta && (1) \\ y' &= (y - y_0)\cos\theta - x\sin\theta. && (2) \end{aligned}$$

The position of the center of the front in the front coordinate system is

$$y_f = a\exp(-x'/\alpha)\sin(kx' - \omega t), \tag{3}$$

where $k = 2\pi/\lambda$ is the wavenumber and $\omega = 2\pi/T$ is the radial frequency. The temperature field is then given by

$$\tau = \tau_0 + d\tau\tan^{-1}(y' - y_f)/\beta, \tag{4}$$

where

$$\begin{aligned} \tau_0 &= (\tau_N + \tau_S)/2 && (5) \\ d\tau &= (\tau_N - \tau_S)/\pi. && (6) \end{aligned}$$

The sensor model used by the vehicles is `uFldCTDSensor` which simulates the front and takes virtual measurements of the associated temperature field using the class `CFrontSim`. The sensor simulator is operated in the shoreside community. A vehicle requests the measurement by publishing the variable `UCTD_SENSOR_REQUEST` to the MOOSDB with the value `vname=my_auv`. In the template configuration the measurement request are scheduled by the process `uTimerScript`, with the sampling period set iin the plug `plug_uTimerScript.moos`. When you develop your behavior you may use the scheduler or directly issue the requests from your behavior. The measurement is returned in the MOOS variable `UCTD_SENSOR_REPORT` with the content

```
vname=my_auv,utc=123456789.0,x=123.45,y=345.67,temp=22.34
```

You control the dynamics of the frontal simulator by setting the ground truth parameters in the plug `plug_uFLDCTDSensor.moos`:

```
//---------------------------------------------------
// uFldCTDSensor configuration block from plugin

ProcessConfig = uFldCTDSensor
{
  AppTick   = 3
  CommsTick = 3

   // Configuring Model of Dynamic Front
  xmin  = 0;
  xmax  = 500;
  ymin  = -400;
  ymax  =  0;

  offset = -90;              // y_0
  angle  = 5;                // front angle theta
  amplitude = 20;            // spatial amplitude a
  period = 200;              // temporal period T
  wavelength = 200;          // spatial wavelength lambda
  alpha = 400;               // spatial 1/e length scale alpha
  beta = 20;                 // length scale of frontal gradient beta
  temperature_north = 20;    // temperature North of front
  temperature_south = 25;    // temperature South of front
  sigma = 0.01;              // standard deviation of gaussian sensor noise
}
```

3.2 Parameter Estimation

On your vehicle you will run the parameter estimation process *pFrontEstimate* which subscribes to the sensor report. It must be configured with the minimum and maximum of all the frontal model parameters in the plug *plug_pFrontEstimate.moos*:

```
ProcessConfig = pFrontEstimate
{
  AppTick    = 4
  CommsTick  = 4

  vname = $(VNAME)
  temperature_factor = $(COOL_FAC)
  cooling_steps = $(COOL_STEPS)
  min_offset = -120;
  max_offset = -60;
  min_angle = -5;
  max_angle = 10;
  min_amplitude = 0;
  max_amplitude = 50;
  min_period = 150;
  max_period = 250;
  min_wavelength = 100;
  max_wavelength = 300;
  min_alpha = 399;
  max_alpha = 401;
  min_beta = 10;
  max_beta = 30;
  min_T_N = 15;
  max_T_N = 25;
  min_T_S = 20;
  max_T_S = 30;
  concurrent = true           // Flag controlling whether the
                              // annealing is performed concurrently
                              // with survey.
}
```

The cooling parameters are set in the launch script or on the command line. Note the parameter *concurrent* which is used to control whether the annealing is performed concurrently with the survey. This feature can be used to save time, but it has to be used with care, in particular if combined with fast cooling (cooling_steps small), because the parameters may 'freeze' before the survey reaches the areas which provide the most information.

Note also that you may 'fix' one or more variables by simply setting the min and max values equal to the true value. This is useful for determining the utility of a survey strategy for determining a specific parameter or the coupling between a couple of parameters, as you will be ask do do in the assignments.

The measurement collection and the parameter estimation is initiated when the survey flag is set in the MOOSDB:

```
SURVEY_UNDERWAY = true
```

which must be set by the survey behavior. You can see how this is done with the *active_flag* in the meta file for the vehicle helm, *meta_vehicle.bhv*:

```
//-----------------------------------------------------
// Helm Behavior file

initialize   DEPLOY  = true
initialize   RETURN  = false
initialize   STATION_KEEP = false
initialize   SURVEY  = true
initialize   AVOID   = true
initialize   SURVEY_UNDERWAY = false
```

```
set MODE = ACTIVE {
  DEPLOY = true
} INACTIVE

set MODE = RETURNING {
  MODE = ACTIVE
  RETURN = true
}

set MODE = SURVEYING {
  MODE = ACTIVE
  SURVEY = true
  RETURN = false
}
//----------------------------------------------
Behavior = BHV_Waypoint
{
  name      = waypt_survey
  pwt       = 100
  condition = MODE==SURVEYING
  perpetual = true
  updates   = SURVEY_UPDATES
  activeflag   = SURVEY_UNDERWAY = true
  inactiveflag = SURVEY_UNDERWAY = false
  endflag   = RETURN = true

//     cycleflag = SURVEY = false
//        repeat = 1
          lead = 8
   lead_damper = 1
         speed = 2.0   // meters per second
        radius = 8.0
        points = format=lawnmower, label=dudley_survey, x=$(SURVEY_X), y=$(SURVEY_Y), \
                 width=$(WIDTH), height=$(HEIGHT), lane_width=$(LANE_WIDTH),          \
                 rows=north-south, degs=$(DEGREES)
  visual_hints = nextpt_color=red, nextpt_lcolor=khaki
  visual_hints = vertex_color=yellow, line_color=white
  visual_hints = vertex_size=2, edge_size=1
}
```

When you write your own behavior in the second part of the assignment you will have to configure it in a similar way.

3.3 Assignment Section A

In the first part of the assignment, we will focus on developing a fundamental understanding of the front model parameterization and the role of the survey pattern in isolating or coupling the various parameters. Also, we are using a simulated annealing algorithm for the parameter estimation, and we have to make sure we choose the cooling schedule properly to arrive at a good parameter estimate. We will try to achieve that throgh a sequence of excersises using the template configuration in *missions/u6_alpha*. Your lab report should include a description of what you are doing in each component, and what you are learning.

The lab is open-ended and unstructured, and you should use your own intuition for this 'learning phase':

- Note that there is no unique solution, so use your common sense in your analysis.
- The parameter settings in the following execises are only suggestions. If you see something interesting, choose your own settings.
- Also, do not base your discussion on a single survey run, the score may vary significantly due to the stochastic nature of the estimator.. You can do this by simply re-deploying the vehicle.

3.3.1 Annealing Schedule

For this exercise we will initially fix some of the parameters, and make the front quasi-stationary:

```
alpha = 500
beta = 20
angle = 5
period = 1e6
```

Run the default survey with survey with *–angle=0*, changing the cooling factor.

Next, open up the search interval for the parameter *alpha* to the interval $[300, 800]$. Do you observe a difference in performance? Explain.

Also, try to run the annealer after all measurements are collected, by stting *concurrent = false*.

3.3.2 Parameter Sensitivity

Fix all parameters except one, the wavelength, run several surveys andd make a plot of the score versus estimated value.

Repeat the excersise with two parameters unknown, the wavelength and the period.

3.3.3 Dynamic Front

Now add dynamics by setting *period=200* with search interval $[100, 300]$. Discuss the results.

3.3.4 Survey Pattern

Repeat the survey with fixed with *–angle=270* and *angle=225.* Also, try changing the survey lane width. Discuss your findings. You may want to focus on wavelength and period, fixing all other parameters. Which survey angle do you think best separates the two parameters? Is your expectation confirmed by the survey?

3.3.5 Multiparameter Estimation

Open up the search interval for all parameters.

3.4 Assignment Section B

3.4.1 Adaptive Sampling Behavior

In the second part of the assignment you should use your experience from the survey exercises to develop your own strategy for sampling the field and estimate the parameters as accurately as possible in the shortest amount of time, and build it into a new IvPHelm behavior, adapting to the measured temperature field.

Your behavior must perform both the search for the front and then survey it in an optimal manner. You may either write a single behavior doing both phases or you may set up the search and the mapping phases as separate modes. The objective is to get the smallest parameter error in the shortest amount of time.

3.4.2 Collaborative Sampling

We are not asking you to perform the survey with two collaborating vehicles, but we may do that on the river. As the last part of the assignment, we ask you to speculate on how you would design a collaborative behavior strategy for reaching the objective.