A Guide to Artifact Searching using MOOSIvP

Andrew Shafer, Michael Benjamin
Dept of Electrical Engineering/Computer Science, MIT
Cambridge MA 02139
<ajshafer@mit.edu>, <mikerb@mit.edu>

October 26, 2007

***Abstract*— Documentation for the MOOS/IvP artifact search system. Includes notes for pSensorSim, pArtifactMapper, artfield-generator, and bhv_SearchGrid. Includes an example mission as a tutorial for users experienced with MOOS/IvP**

I. Introduction

This document describes the use and development of the artifact search system developed as a Master's of Engineering thesis by Andrew Shafer at MIT. This document assumes that the reader has a familiarity with MOOS and IvP and understands how to use those tools (see [6], [1], [2], and [3]).

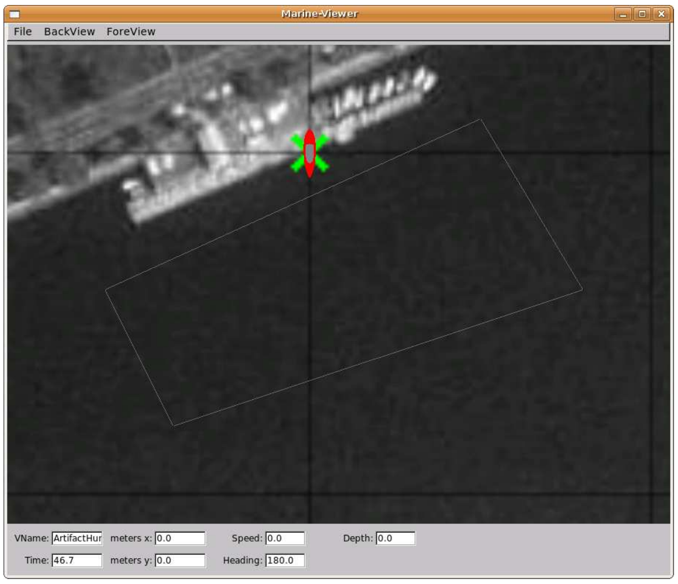
First, a bit of terminology. In this document, an "artifact" is an object of interest. An artifact can be any detectable, identifiable object. In a naval application this would commonly be some type of mine. In naval terminology, "mine-hunting" (or mine-sweeping) usually refers to the process of detecting mines and deactivating or destroying them. "Mine-searching," on the other hand, refers to simply mapping out the locations of detected mines for later deactivation/destruction. Therefore, this project is properly called an artifact searching system, rather than a mine-hunting system.

A "search area" is the geographic region that the user desires to search (see Fig. 1). This area is broken up into uniform, discrete cells that together constitute the "search grid" (see Fig. 2).

To map the search area, several platforms are available. A "platform" is the combination of vehicle type (e.g. autonomous kayak, AUV, human-navigated vessel, etc.) and sensor capabilities (e.g. side-scan sonar, FLIR, MAD, etc.). With a single vehicle trying to cover a search area with a uniform, perfect-detection sensor, a lawn-mower pattern is optimal (see [4] and [5]). With multiple, identical platforms, a straightforward approach is to divide the overall area into similarly-sized, smaller areas and assign a single vehicle to each area. This approach, however, will fail if one of the vehicles breaks down during the operation. It also is not clear that this solution is optimal when multiple, different platforms are used (e.g. A kayak with side-scan sonar and an AUV with FLIR).

The goal of the artifact search system is to develop an algorithm to allow multiple platforms to efficiently search for artifacts in a given search area, respecting constraints on time, vehicle dynamics, and sensor performace.

In the current instantiation of the search system, there are two main MOOS processes and one IvPHelm behavior.



VName: ArtifactHur meters x: 0.0 Speed: 0.0 Depth: 0.0
Time: 46.7 meters y: 0.0 Heading: 180.0

Fig. 1. A geographic area (a convex polygon) defined as a search area.

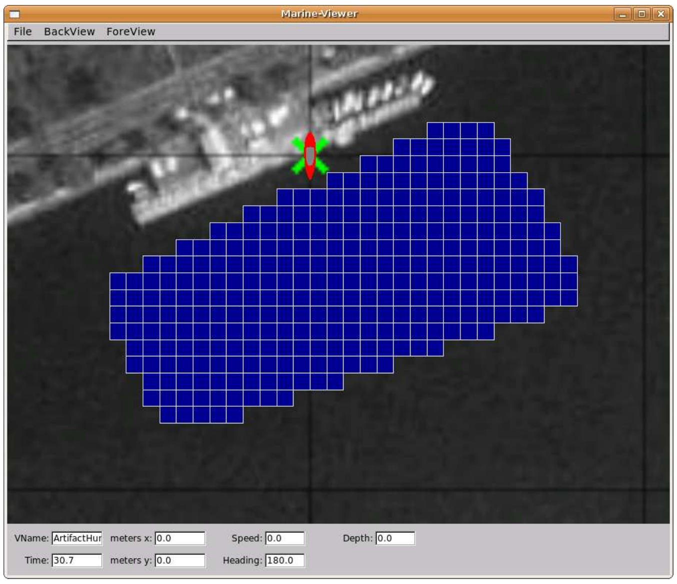
pSensorSim simulates the output of an imaginary sensor in a simulated artifact field. pArtifactMapper takes the output of pSensorSim, fuses it with output from other artifact search platforms in the area, and produces a likelihood map of artifacts in the search region. The IvPHelm behavior, bhv_SearchGrid, provides desired heading and speed information to the helm to optimize the user's utility function (e.g. mapping an entire field with 95% confidence in the least amount of time).

II. pSensorSim

pSensorSim is composed of two C++ classes, ArtifactField and SensorModel. Most users will not directly use these classes and will instead interact with them through the MOOS-App pSensorSim.

A. Class ArtifactField

ArtifactField simulates an artifact field. Internally, it is a vector of strings, where each string represents one artifact. An artifact string consists of a comma separated list of equal-sign delimited variable-value pairs. For example, "Var1=val1,Var2=val2,Var3=val3". This structure makes it



VName: ArtifactHur meters x: 0.0 Speed: 0.0 Depth: 0.0
Time: 30.7 meters y: 0.0 Heading: 180.0

Fig. 2. A search grid defined over a search area.

easy to add new traits to artifacts without having to change much code in other segments.

ArtifactField can return a list of artifacts within a 2D rectangle or circle when the artifact strings contain both “X=xval” and “Y=yval” (e.g “X=10,Y=4.5,TYPE=magnetic”).

Public Member Functions:

- void **addArtifact** (std::string)
  Puts an artifact into the field.
- void **addArtifact** (double, double)
  Constructs the proper string from an x, y *pair.*
- std::string **getArtifact** (int) const
  Returns the artifact at index i.
- int **size** () const
  Returns the number of artifacts in the field.
- std::vector< std::string > **getArtifactbox** (double, double, double, double) const
  Returns a vector of all artifacts within the 2D, X,Y box specified by the parameters.
- std::vector< std::string > **getArtifactcircle** (double, double, double) const
  Returns a vector of all artifacts within the 2D, X,Y circle specified by the parameters.

B. Class SensorModel

SensorModel models the output of a specified sensor on a given ArtifactField. After creating a SensorModel object, the programmer initializes the sensor by calling **setSensorModel** with the name of the model to simulate (currently, only a fixed radius, guaranteed-detection sensor is modeled, “fixedradius”) and setting the detection radius using **setSensorRadius**. The programmer can query the sensor by calling **querySensor** with a query string that is determined by the sensor. For the fixed-radius sensor, the query string should contain the current X and Y values, e.g. “X=4.5,Y=1.3”.

Public Member Functions:

- bool **setSensorModel** (std::string const)
  Currently accepted value is “fixedradius”.
- void **setSensorRadius** (double)
  The sensor radius must be a non-negative value.
- double **getSensorRadius** () const
- std::vector< std::string > **querySensor** (std::string const, **ArtifactField** const &) const
  Parameters: *ArtField is a reference to an artifact field*

Private Member Functions

- std::vector< std::string > **queryFRSensor** (std::string const, **ArtifactField** const &) const
  A private method for querying a fixed-radius sensor.

Private Attributes

- double **dSensorRadius**
  The maxiumum effective sensor radius.
- std::string **sSensorType**
  A string holding the current sensor type.

C. MOOSApp pSensorSim

Combining these two classes and creating a MOOSApp, we get Fig. 3.

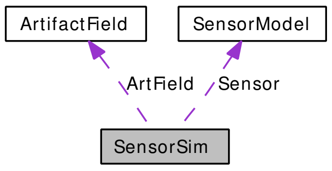


Fig. 3. A class diagram for pSensorSim

1) Configuration: The pAntler configuration block for pSensorSim looks like this:

```
//------------------------------------------
// pSensorSim config block
ProcessConfig = pSensorSim
{
   AppTick    = 4
   CommsTick  = 4

   ArtifactFile = mines.art
   Artifact = X=10,Y=10
   Sensor = FixedRadius
   Sensor_Radius = 10
}
```

ArtifactFile: Optional. This is the path to a file that contains lines of the form “Artifact = artifactstring”. Blank lines are ignored. Useful for adding randomly generated fields using artfieldgenerator. See Appendix I

Artifact: Optional. An artifact string to add to the artifact field.

Sensor: Mandatory. A string containing the sensor type to simulate. Only FixedRadius is currently implemented.

Sensor_Radius: Optional. The effective radius of the FixedRadius sensor. Defaults to 10m, must be non-negative.

2) MOOS Variables:

a) Subscribes:

NAV_X and NAV_Y: Used to determine the current location of the sensor. Uses only the most recently received value.

b) Publishes:

DETECTED_ARTIFACT: A string that contains the output of the sensor evaluated at the current position. For the fixed radius sensor, the output is “X=x_val,Y=y_val,Prob=probability”. pArtifactMapper subscribes to this variable.

VIEW_POLYGON: On each iteration, plots a 12-point hexagon of radius 10-m around the kayak with label “ArtifactHunter”. The string is ”radial:x,y,10,12,0.0,ArtifactHunter”.

VIEW_POINT: Published once on startup for each artifact loaded into the artifact field.

III. pARTIFACTMAPPER

pArtifactMapper implements MOOSApp functionality for the XYArtifactGrid C++ class.

A. Class XYArtifactGrid

XYArtifactGrid is a simple class derived from XYGrid. Using the functionality of XYGrid, this class is able to instantiate a search grid on top of a given search area. Each cell in this search grid has a value member and a utility member. The value member can be set to any double. The utility member is bounded by the minimum and maximum utilities set by the programmer.

B. MOOSApp pArtifactMapper

The simple class diagram for pArtifactMapper is shown in Fig. 4.

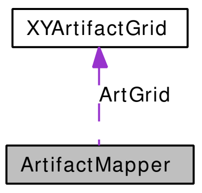


Fig. 4. A class diagram for pArtifactMapper

1) Configuration: The pAntler configuration block for pArtifactMapper looks like this:

```
//------------------------------------------
// pArtifactMapper config block
ProcessConfig = pArtifactMapper
{
   AppTick   = 4
   CommsTick = 4

   GridPoly =  poly:label,A:-60,-40:50,10:80,-40:-40,-80
   GridSize = 5.0
   GridInit = .5
}
```

GridPoly: Mandatory. A valid polygon initialization string (must be convex) that covers the search area.

GridSize: Mandatory. The width/height of each cell in the search grid in meters.

GridInit: Mandatory. The initializing value for each cell.

2) MOOS Variables:

a) Subscribes:

DETECTED_ARTIFACT: The sensor output strings are scanned for their x, y, and probability values. The x, y location is mapped to a cell in the search grid and that cell’s value and utility is updated to the probability. After this, a GRID_DELTA string is published.

ARTIFACTMAP_REFRESH: Any process can set this value to TRUE to ask pArtifactMapper to publish its current artifact field state. This is published as a series of GRID_DELTA updates for each cell. After running the update, ARTIFACTMAP_REFRESH is set to FALSE.

b) Publishes:

GRID_CONFIG: This is published on startup and on connecting to the server. This string is published to get pMarineViewer to plot the search grid and set up its internal XYGrid.

GRID_DELTA: A GRID_DELTA string is published when the current probability in a DETECTED_ARTIFACT cell differs from the detected probability. The string format is the same as that used by pMarineViewer, “label@index,oldval,newval,oldutil,newutil”.

GRID_CONFIG: This is published on startup and on connecting to the server. This string is published to get pMarineViewer to plot the search grid and set up its internal XYGrid.

NOTE: pArtifactMapper currently does not listen for or accept updates from other sources (unless it get published as a DETECTED_ARTIFACT. Listening for other GRID_DELTA updates is a feature that needs to be implemented.

IV. BHV_SEARCHGRID

V. EXAMPLE MISSIONS

A. Tutorial

This example gives a tutorial on how one might go about creating and executing a mission to search for artifacts.

1) Setup: The first step is to define the search area. Using polyview, click on a few points (maintaining a convex polygon) to create the search area and export the polygon string.

We now generate a random artifact field for searching. In the directory you want to run your mission file from:

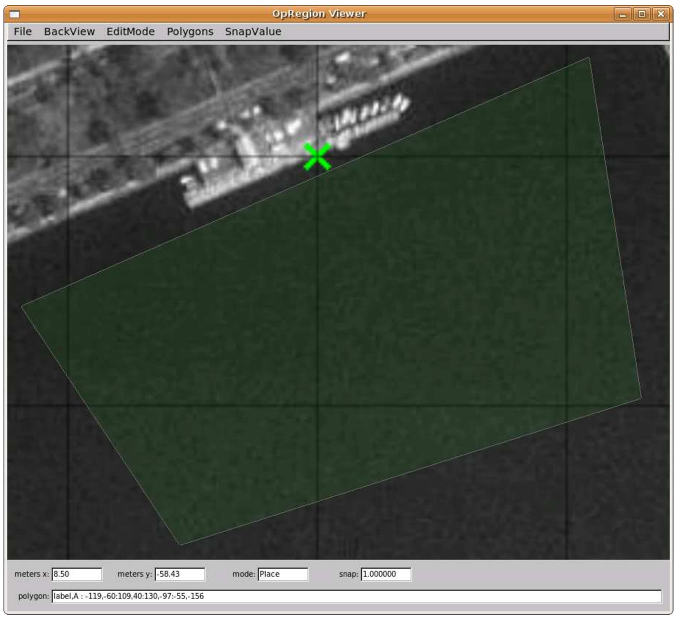
```
artfieldgenerator label,A:-119,-60:109,40:130,-97:-55,-156
.25 25 > mines.art
```

This generates some random artifacts:

```
head -4 mines.art
ARTIFACT = X=92.5,Y=26.25
ARTIFACT = X=59.25,Y=-47.25
ARTIFACT = X=-30.25,Y=-60.25
ARTIFACT = X=88.5,Y=-85.25
```

The next setup task is to create the MOOS mission file. See Appendix II for a working example. The relevant portions are printed below:

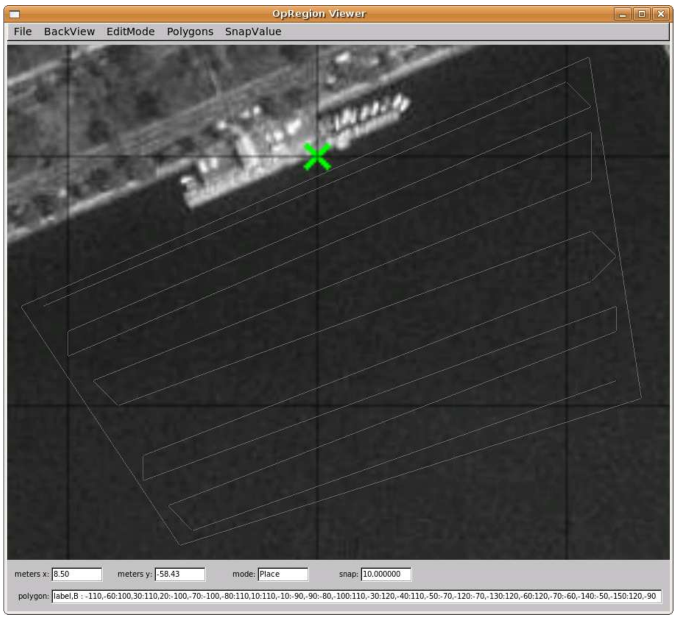
```
//------------------------------------------
// pSensorSim config block
ProcessConfig = pSensorSim
{
   AppTick   = 4
   CommsTick = 4

   ArtifactFile = mines.art
   Artifact = X=10,Y=10
   Sensor = FixedRadius
```



meters x: 8.50 meters y: -58.43 mode: Place snap: 1.000000

polygon: label,A : -119,-60:109,40:130,-97:-55,-156

Fig. 5. Defining the search area in polyview



meters x: 8.50 meters y: -58.43 mode: Place snap: 10.000000

polygon: label,B : -110,-60:100,30:110,20:-100,-70:-100,-80:110,10:110,-10:-90,-90:-80,-100:110,-30:120,-40:110,-50:-70,-120:-70,-130:120,-60:120,-70:-60,-140:-50,-150:120,-90

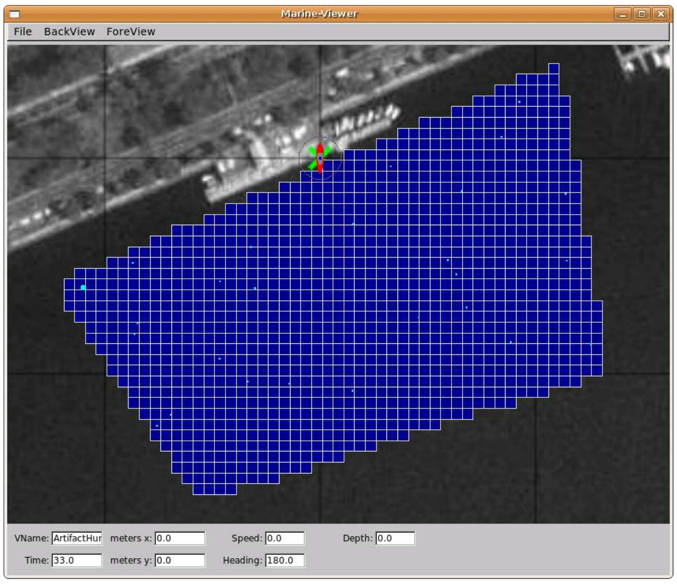
Fig. 6. Defining the lawn-mower pattern in polyview.

```
   Sensor_Radius = 10
}

//------------------------------------------
// pArtifactMapper config block
ProcessConfig = pArtifactMapper
{
   AppTick   = 4
   CommsTick = 4

   GridPoly =  label,A:-119,-60:109,40:130,-97:-55,-156
   GridSize = 5.0
   GridInit = .5
}
```

We also need to configure the helm to run a simple behavior to search over the search area. To do this we first need to create a sequence of points for the vehicle to search over. Load polyview with the previously defined search area (pass polyview a file with a line that reads “Polygon = polystring” to get it to load) and create a new “polygon” (not really a polygon as it will not be convex) whose points are the points in the lawn-mower pattern. See Fig. 6 for an example. Export this sequence of points and put it in the bhv file. See Appendix III for an example behavior.

2) Launch: After getting setup for the mission, we invoke it with `pAntler mission.moos`. The display should look like Fig. 7. The blue grid is the search grid, the light blue dots are artifacts, and the circle around the kayak is the detection radius. To start the helm, in iRemote, set `Deploy = TRUE` (key 4), and then relinquish manual control (‘o’).

The kayak will now loop through the points defined in the lawn-mower pattern and the various displays will update accordingly. See Fig. 8



VName: ArtifactHur meters x: 0.0 Speed: 0.0 Depth: 0.0

Time: 33.0 meters y: 0.0 Heading: 180.0

Fig. 7. pMarineViewer at the beginning of an artifact search mission. The blue grid is the search grid. The bright-blue dots are artifacts. The circle around the kayak is the 10m detection radius.

REFERENCES

[1] Michael R. Benjamin. *Interval Programming: A Multi-Objective Optimization Model for Autonomous Vehicle Control*. PhD thesis, Brown University, Providence, RI, May 2002.

[2] Michael R. Benjamin. Multi-Objective Navigation and Control Using Interval Programming. In *Proceedings of the Multi-Robot Systems Workshop*, NRL, Washington DC, March 2003.

[3] Michael R. Benjamin. The Interval Programming Model for Multi-Objective Decision Making. Technical Report AIM-2004-021, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, September 2004.

[4] H. Choset. Coverage for robotics—a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.

[5] Acar E., H. Choset, Y. Zhang, and M. Schervish. Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabalistic methods. *The International Journal of Robotics Research*, 22(7–8):441–466, July–August 2003.

[6] Paul M. Newman. MOOS - A Mission Oriented Operating Suite.

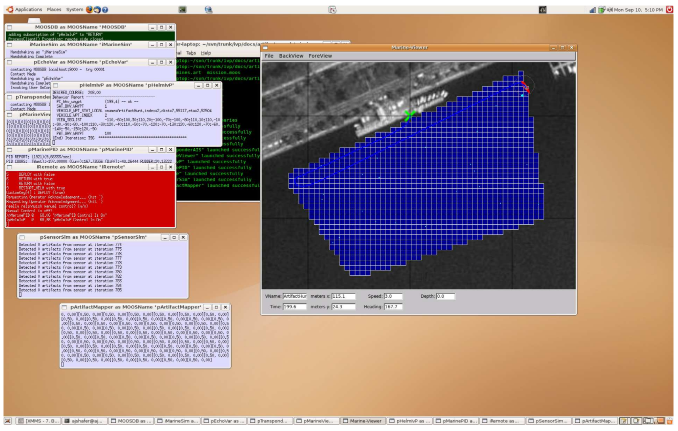


Fig. 8. The various search components working together in the middle of a mission.

Technical Report OE2003-07, MIT Department of Ocean Engineering, 2003.

Appendix I
artfieldgenerator

artfieldgenerator is a command line tool for generating random artifact fields for testing various behaviors and algorithms.

To use artfieldgenerator:

```
artfieldgenerator poly_string step_size num_artifacts
```

poly_string: A valid polygon initialization string (convex). All of the artifacts will be contained within this polygon. Some artifacts, however, may exist outside a search grid, depending on the size of the grid elements.

step_size: Where to step the X, Y values (e.g. .25 generates values in .25 increments)

num_artifacts: Number of unique artifacts to generate

The output of artfield generator is written to standard out, so it is often redirected to a file for later use:

```
artfieldgenerator label,A:-60,-40:50,10:80,-40:-40,-80
.25 25 > mines.art
```

Appendix II
Tutorial Mission

Line break in polygon string is provided for readability only. File: mission.moos

```
// MOOS file

ServerHost = localhost
ServerPort = 9000
Simulator  = true

Community  = ArtifactHunt

LatOrigin  = 42.3584
LongOrigin = -71.08745

//------------------------------------------
// Antler config  block
ProcessConfig = ANTLER
{
  MSBetweenLaunches = 200

  Run = MOOSDB       @ NewConsole = true
  Run = iMarineSim   @ NewConsole = true
  Run = pEchoVar     @ NewConsole = true
  Run = pLogger      @ NewConsole = true
  Run = pTransponderAIS @ NewConsole = true
  Run = pMarineViewer    @ NewConsole = true
  Run = pHelmIvP     @ NewConsole = true
  Run = pMarinePID   @ NewConsole = true
  Run = iRemote      @ NewConsole = true
  Run = pSensorSim   @ NewConsole = true
  Run = pArtifactMapper @ NewConsole = true
}

//------------------------------------------
// pHelmIvP config block

ProcessConfig = pHelmIvP
{
  AppTick    = 2
  CommsTick  = 2

  Behaviors  = artifactsearch.bhv
  Verbose    = verbose
  Domain     = course:0:359:360
  Domain     = speed:0:5:21

  ACTIVE_START = true
}

//------------------------------------------
// pSensorSim config block
ProcessConfig = pSensorSim
{
   AppTick   = 4
   CommsTick = 4

   ArtifactFile = mines.art
   //Artifact = X=10,Y=10
   Sensor = FixedRadius
   Sensor_Radius = 10
}

//--------------------------------------------
// pArtifactMapper config block
ProcessConfig = pArtifactMapper
{
   AppTick   = 4
   CommsTick = 4

   GridPoly =  label,A:-119,-60:109,40:130,
                  -97:-55,-156
   GridSize = 5.0
   GridInit = .5
}

//--------------------------------------------
// pMarineViewer config block
ProcessConfig = pMarineViewer
{
  AppTick     =  4
  CommsTick   =  4

  TIF_FILE = ./Default.tif
}

//--------------------------------------------
// pMarine config block
ProcessConfig = pMarinePID
{
  AppTick    = 10
  CommsTick  = 10

  Verbose    = true

  DEPTH_CONTROL = false

  // Yaw PID controller
  YAW_PID_KP        = 0.5
  YAW_PID_KD        = 0.0
  YAW_PID_KI        = 0.0
  YAW_PID_INTEGRAL_LIMIT = 0.07

  // Speed PID controller
  SPEED_PID_KP        = 1.0
  SPEED_PID_KD        = 0.0
  SPEED_PID_KI        = 0.0
  SPEED_PID_INTEGRAL_LIMIT = 0.07

  // Maximums
  MAXRUDDER  = 100
  MAXTHRUST  = 100

  // A non-zero SPEED_FACTOR overrides
  //     use of SPEED_PID
  // Will set DESIRED_THRUST =
  //     DESIRED_SPEED * SPEED_FACTOR
  SPEED_FACTOR           = 20
}

//--------------------------------------------
// iMarineSim config block
ProcessConfig = iMarineSim
{
  AppTick       = 10
  CommsTick     = 10
  MaxTransVel   = 3.0
  MaxRotVel     = 0.6
  StartLon      = 0
  StartLat      = 0
  StartSpeed    = 0
  StartHeading  = 180
}

//--------------------------------------------
// iRemote configuration block
ProcessConfig = iRemote
{
   CustomKey = 1 : HELM_VERBOSE @ "verbose"
   CustomKey = 2 : HELM_VERBOSE @ "terse"
   CustomKey = 3 : HELM_VERBOSE @ "quiet"
   CustomKey = 4 : DEPLOY @ "true"
   CustomKey = 5 : DEPLOY @ "false"
   CustomKey = 6 : RETURN @ "true"
   CustomKey = 7 : RETURN @ "false"
   CustomKey = 9 : RESTART_HELM @ "true"
}

//--------------------------------------------
// Logger configuration block

ProcessConfig = pLogger
{
  //over loading basic params...
  AppTick   = 20.0
  CommsTick = 20.0

  File      = HW
  PATH      = ../data_from_runs/
  SyncLog   = true @ 0.2
  AsyncLog  = true
  FileTimeStamp = true

  Log       = DESIRED_THRUST @ 0.1
  Log       = DESIRED_RUDDER @ 0.1
  Log       = NAV_X   @ 0.1
  Log       = NAV_Y   @ 0.1
  Log       = NAV_Yaw @ 0.1
  Log       = NAV_Heading @ 0.1
  Log       = NAV_Speed @ 0.1
  Log       = LOOP_WALL @ 0.1
  Log       = LOOP_CPU @ 0.1
  log       = DIST_TO_REGION @ 0.1
  Log       = AIS_REPORT @ 0.1
}

//--------------------------------------------
// pEchoVar configuration block

ProcessConfig = pEchoVar
{
  AppTick     = 5
  CommsTick   = 5

  Echo    = MARINESIM_X       ->  NAV_X
  Echo    = MARINESIM_Y       ->  NAV_Y
  Echo    = MARINESIM_YAW     ->  NAV_YAW
  Echo    = MARINESIM_HEADING ->  NAV_HEADING
  Echo    = MARINESIM_SPEED   ->  NAV_SPEED
}

//--------------------------------------------
// pTransponderAIS config block

ProcessConfig = pTransponderAIS
{
  AppTick   = 2
  CommsTick = 2
  VESSEL_TYPE   = KAYAK
}
```

APPENDIX III
TUTORIAL BEHAVIOR

Line breaks in point list are provided for readability only.
File: artifactsearch.bhv

```
initialize   DEPLOY = true
initialize   RETURN = false

//--------------------------------------------
Behavior = BHV_Waypoint
{
  name      = bhv_waypt
  pwt       = 100
  condition = DEPLOY = true
  condition = RETURN = false
  perpetual = true
  speed     = 3.0
  radius    = 2.0
  nm_radius = 5.0
  points    = -110,-60:100,30:110,20:-100,
   -70:-100,-80:110,10:110,-10:-90,-90:-80,
   -100:110,-30:120,-40:110,-50:-70,-120:-70,
   -130:120,-60:120,-70:-60,-140:-50,
   -150:120,-90
}
```