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>

January 28, 2008

***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.

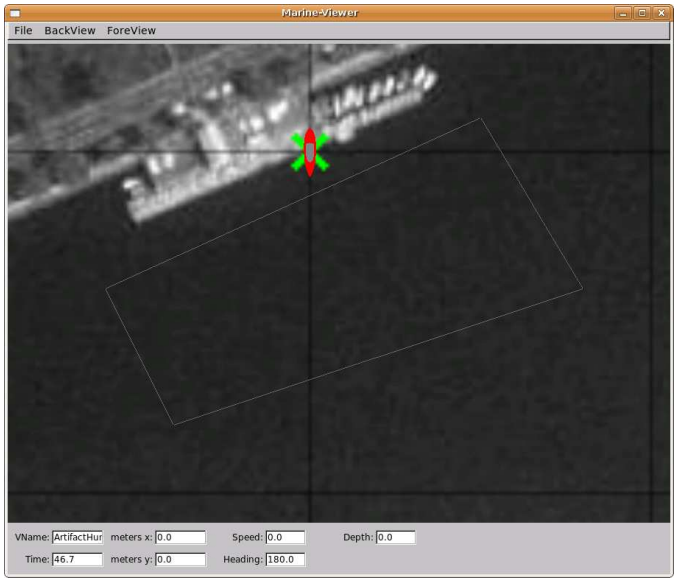

Fig. 1. A geographic area (a convex polygon) defined as a search area.

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 MOOSApp 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

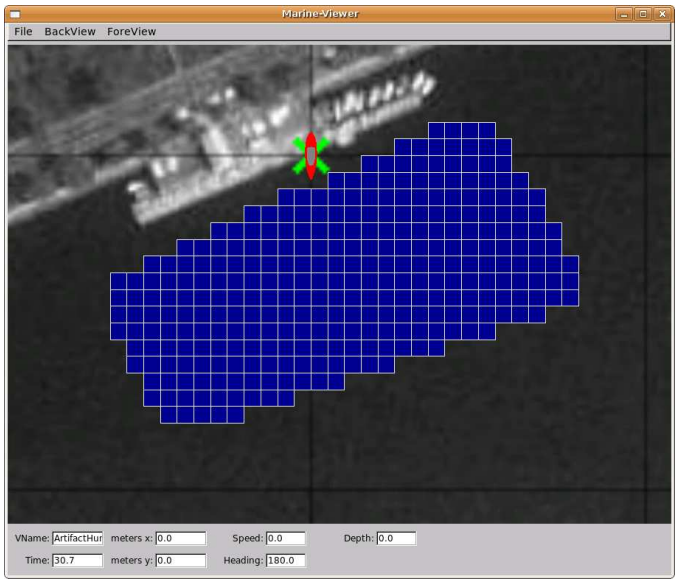

Fig. 2. A search grid defined over a search area.

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.

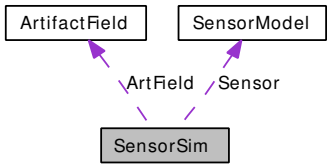

Fig. 3. A class diagram for pSensorSim

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 labelled with the community name. An example 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.

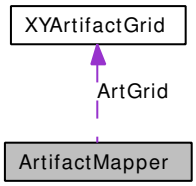

Fig. 4. A class diagram for pArtifactMapper

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 = 0
}
```

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. If this is a new detection, a GRID_DELTA_LOCAL 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.

GRID_DELTA: These delta's come from other mapping processes on other platforms. The contents of this update are placed into the local grid structure.

ARTIFACTMAP_EXPORT: Any process can set this value to a string to ask pArtifactMapper to write its current artifact field state to a file. pArtifactMapper attempts to write out the data to a file with the name specified in the string (e.g. ./data_output). The format of this file has three header lines (Grid label, grid configuration string, and "Index, Value, Utility") followed by strings of the format index,value,utility. This output is used by the scoring utility to calculate the score on a given artifact field.

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_LOCAL: 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.

IV. BHV_SEARCHGRID

THIS SECTION IS CURRENTLY UNDER DEVELOPMENT.

The bhv_SearchGrid used in the example mission (See Section V) is Mike Benjamin's BHV_SearchGrid. It uses a simple heuristic that tries to visit each cell for a specified period of time (approximately 7 seconds). It does not communicate with other vehicles, but when used in conjunction with other behaviors (such as ConstantHeading and ConstantSpeed), it explores the search grid adequately.

Andrew Shafer is currently developing BHV_SearchArtifact to allow inter-vehicle communication. The behavior (and pSensorSim) should also allow different sensor models and change its behavior based on these models.

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 us-

ing two vehicles. One will use a pre-generated lawnmower pattern and the other will use the search artifact behavior. HUNTER1 will follow the pregenerated lawnmower pattern while HUNTER2 will search the grid.

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. Save the string in a file, prefix the string with "polygon = ".

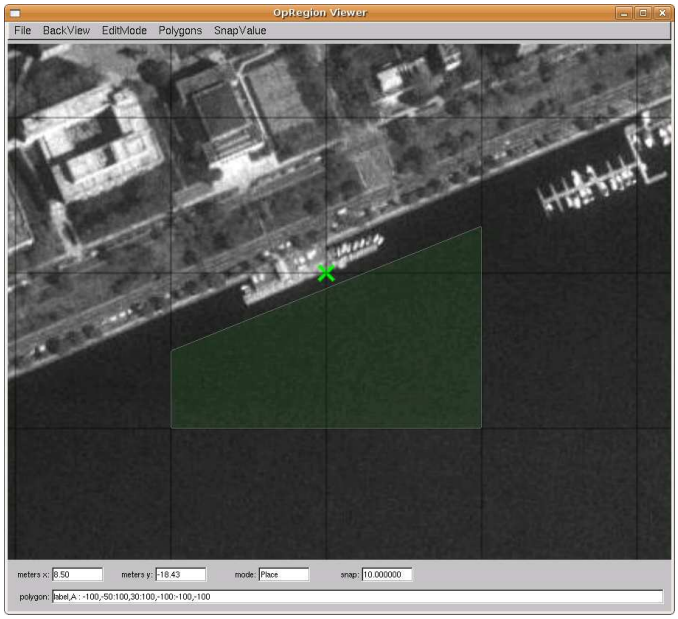

Fig. 5. Defining the search area in polyview

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

```
artfieldgenerator label,ArtGrid:-100,-50:100,30:100,-100:
-100,-100 .25 75 > mines.art
```

This generates some random artifacts and stores them in mines.art:

```
head -4 mines.art
ARTIFACT = X=-60.75,Y=-70.5
ARTIFACT = X=-71.75,Y=-82
ARTIFACT = X=6,Y=-95.25
ARTIFACT = X=-30.5,Y=-89.75
```

The next setup task is to create the MOOS mission file. See Appendix V for working examples for both vehicles and the viewer. The relevant portions are printed below:

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

   ArtifactFile = mines.art
   Sensor = FixedRadius
   Sensor_Radius = 10
}

//----------------------------------------
// pArtifactMapper config block
ProcessConfig = pArtifactMapper
{
   AppTick   = 4
   CommsTick = 4
   GridPoly = label,ArtGrid:-100,-50:100,30:
    100,-100:-100,-100
   GridSize = 5.0
   GridInit = .5
}
```

We also need to configure the helm to search over the search area. For the first vehicle (HUNTER1), we want it to run a lawnmower pattern on the area. generatelawnmower can create this pattern for us (pLawnmower can be used during runtime for dynamic path creation).

```
generatelawnmower poly 45 10 .25

cat poly_seglists
label,ArtGrid : -99.75,-50:-100,-50.25:-100,-78.5:-52.5,-31:
-5.5,-12.25:-93.25,-100:-65,-100:41.75,6.75:88.75,25.5:
-36.75,-100:-8.5,-100:100,8.5:100,-19.75:19.75,-100:
48.25,-100:100,-48.25:100,-76.5:76.5,-100
```

Now put this list of points in the behavior file for the point sequence (cut off the label,ArtGrid portion). See Appendix VI for the full behavior.

We also put the original polygon in the behavior file for the "searching" vehicle (HUNTER2). It's important to append this line with the same parameters used to configure pArtifactMapper. From above, we would add "@5.0,5.0@.5" to the end of the polygon configuration string in HUNTER2's behavior file.

2) Launch: After getting setup for the mission, we invoke it with `./startall.sh` (see Appendix VII). The display should look like Fig. 6. The blue grid is the search grid, the light blue dots are artifacts, and the circle around the kayak is the detection radius.

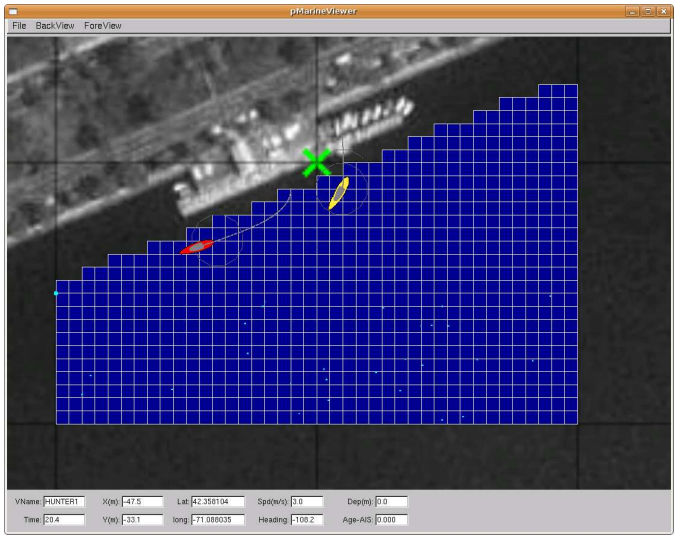

Fig. 6. pMarineViewer at the beginning of an artifact search mission. The blue grid is the search grid. The bright-blue dots are artifacts. The circle around the kayaks is the 10m detection radius.

HUNTER1 will now loop through the points defined in the lawn-mower pattern while HUNTER2 executes the grid searching behavior. The grid will change to red when it detects an artifact in that cell. See Fig. 7

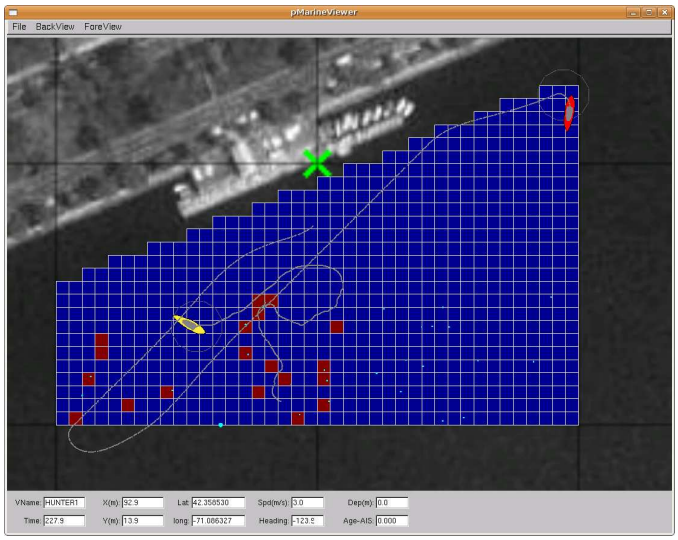

Fig. 7. The various search components working together in the middle of a mission.

To analyze the results of the survey, we have to export the artifact grid to a file using pArtifactMapper. Using any MOOS database poking tool (such as uMS), set ARTIFACTMAP_EXPORT to "./hunter1output" or "./hunter2output" accordingly.

Create a new file for the scoring metrics:

```
hit = 10
miss = -1000
falsealarm = -10
corrrej = 10
threshold = .75
```

To analyze the score for that log file, use scoreartfield.

```
user@machine$ scoreartfield hunter2output mines.art
score results1
Loaded ArtGrid: 747 elements.
Loaded ArtField: 50 artifacts.
Loaded Metrics: 10 -1000 -10 10 0.75
user@machine$ cat results1
Score: 7230
Hits: 41
Misses: 0
FalseAlarms: 12
CorrRejs: 694
HitPoint: 10
MissPoint: -1000
FalseAlarmPoint: -10
CorrRejPoint: 10
Threshold: 1
```

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. 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
GENERATELAWNMOWER

generatelawnmower is a command-line tool for generating lawnmower patterns for covering a polygon

To use generatelawnmower:

`generatelawnmower x y angle radius clockwise input_file [snap]` or `generatelawnmower input_file [snap value]` or `generatelawnmower input_file angle radius [snap value] (generates full patterns)`

x and y The x and y locations for the initial position to start from.

angle The direction the paths should lie, with 0 being ”north” and increasing to 360 clockwise.

radius Half the distance between the rows.

clockwise Determines the direction of the first turn, that is, to the right or the left.

input_file A list of polygons to generate patterns for. Uses the MBUtil readPolysFromFile, so polygon lines should be prefixed with “polygon = ”. This value is also used as the header for the output files, _polys _seglists. These two files have a 1-to-1 coorespondence.

snap Value to “snap” the artifacts to. E.g., .25 will allow artifacts to take on values x.00, x.25, x.50, x.75. An empty snap or snap of 0.0 does not snap the value

In the last usage example, this version will generate a full lawnmower pattern (hence no initial position or turn direction). All other versions generate a pattern from a single interior point to the rest of the polygon.

The output of generatelawnmower is written to two files and some diagnostic data to standard out.

APPENDIX III
PLAWNMOWER

pLawnmower is a MOOSApp that can create lawnmower patterns on the fly.

A. MOOSApp pLawnmower

1) Configuration: pLawnmower does not read in any configuration parameters.

2) MOOS Variables:

a) Subscribes:

LAWNMOWER_POLYGON: A LAWNMOWER_POLYGON string looks like this, arguments in any order: poly=polystring#x=blah#y=blah#ang=blah#radius=blah [#clockwise=truefalse][#snap=val][#full=truefalse]; x, y, clockwise are not necessary when using full output x, y = initial position ang = [0,360], N = 0, clockwise positive radius = 1/2 distance between rows clockwise is first turn, defaults to true; snap is snap value, defaults to no snapping;

b) Publishes:

LAWNMOWER_SEGLIST: This is seglist containins the points that were requested. The label is the same as the one passed in by the polygon string (it is a good idea to check this to make sure the seglist matches the polygon).

VIEW_POLYGON: Plots the polygon that was just requested.

VIEW_SEGLIST: Plots the seglist that was just created.

APPENDIX IV
SCOREARTFIELD

scoreartfield is a command-line tool for scoring the output of pArtifactMapper.

To use scoreartfield:

```
scoreartfield artmapper.file artifactlocs.file
scoringmetric.file output.file
```

artmapper.file: The file that contains the output from pArtifactMapper when it receives ARTIFACTMAP_EXPORT.

artifactlocs.file: The ground truth values of where the artifacts are located. Usually generated from generateartifacts

scoringmetric.file: Holds the scoring metrics to use. In any order, the file must have these lines:

```
hit = 10
miss = -1000
falsealarm = -10
corrrej = 10
threshold = .75
```

The first four lines indicate the score for each of those situations (using standard signal detection terminology). The threshold is the value that the cell’s utility is compared against to determine whether the declaration is “artifact present” or “artifact absent” and is between 0 and 1.

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 V
TUTORIAL MISSIONS

Line break in polygon string is provided for readability only.
File: HUNTER1.moos

```
ServerHost = localhost
ServerPort = 9201
```

```
Simulator  = true

// Community name IS the vehicle name
Community  = HUNTER1

// MIT Sailing Pavilion
LatOrigin  = 42.3584
LongOrigin = -71.08745

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

  Run = MOOSDB @ NewConsole = true
  Run = iMarineSim @ NewConsole = false
  Run = pEchoVar @ NewConsole = false
  Run = pTransponderAIS @ NewConsole = false
  Run = pMOOSBridge @ NewConsole = false
   ~ pMOOSBridge_HUNTER1
  Run = pHelmIvP @ NewConsole = true
  Run = pMarinePID  @ NewConsole = false
  Run = pSensorSim  @ NewConsole = true
  Run = pArtifactMapper @ NewConsole = true
}

//------------------------------------------
// iMarineSim config block

ProcessConfig = iMarineSim
{
  AppTick = 10
  CommsTick = 10

  MAX_TRANS_VEL = 5.0
  MAX_ROT_VEL   = 0.6
  START_X       = -10
  START_Y       = -10
  START_SPEED   = 0
  START_HEADING = 180

  //FLOAT_RATE    = 0.5
}

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

ProcessConfig = pEchoVar
{
  AppTick     = 20
  CommsTick   = 20

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


//------------------------------------------
// pTransponderAIS config block
//
// Note: Outgoing processed every other cycle..
//       Incoming processed every cycle.

ProcessConfig = pTransponderAIS
{
  AppTick = 8
  CommsTick = 8

  VESSEL_TYPE   = KAYAK
}

//------------------------------------------
// pMOOSBridge config block

ProcessConfig = pMOOSBridge_HUNTER1
{
  AppTick    = 10
  CommsTick  = 10

  // SHARE = [VAR] -> to-community @ to-host:to-port [VAR]

  SHARE = [AIS_REPORT_LOCAL]   ->
   shipside @ localhost:9123 [AIS_REPORT]
  SHARE = [STATION_CIRCLE]     ->
   shipside @ localhost:9123 [STATION_CIRCLE]
  SHARE = [RSTATION_CIRCLE]    ->
   shipside @ localhost:9123 [STATION_CIRCLE]
  SHARE = [VIEW_POLYGON]       ->
   shipside @ localhost:9123 [VIEW_POLYGON]
  SHARE = [VIEW_POINT]         ->
   shipside @ localhost:9123 [VIEW_POINT]
  SHARE = [RVIEW_POINT]        ->
   shipside @ localhost:9123 [VIEW_POINT]
  SHARE = [PROC_WATCH_SUMMARY] ->
   shipside @ localhost:9123 [PROC_WATCH_SUMMARY_HUNTER1]
  SHARE = [PROC_WATCH_EVENT]   ->
   shipside @ localhost:9123 [PROC_WATCH_EVENT_HUNTER1]

  SHARE = [GRID_CONFIG]        ->
   shipside @ localhost:9123 [GRID_CONFIG]
  SHARE = [GRID_DELTA_LOCAL]   ->
   shipside @ localhost:9123 [GRID_DELTA]
  SHARE = [GRID_DELTA_LOCAL]   ->
   HUNTER1 @ localhost:9201 [GRID_DELTA]
  SHARE = [GRID_DELTA_LOCAL]   ->
   HUNTER2 @ localhost:9202 [GRID_DELTA]

  SHARE = [AIS_REPORT_LOCAL] ->
   HUNTER1   @ localhost:9201 [AIS_REPORT]
  SHARE = [AIS_REPORT_LOCAL] ->
   HUNTER2   @ localhost:9202 [AIS_REPORT]
}

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

ProcessConfig = pHelmIvP
{
  AppTick    = 2
  CommsTick  = 2

  Behaviors  = hunter1.bhv
  Verbose    = verbose
  Domain     = course:0:359:360
  Domain     = speed:0:5:21

  ACTIVE_START = true
}

//------------------------------------------
// pMarine config block

ProcessConfig = pMarinePID
{
  AppTick    = 10
  CommsTick  = 10

  Verbose    = true

  ACTIVE_START  = true

  // 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


  DEPTH_CONTROL = true

  //Pitch PID controller
  PITCH_PID_KP                  = 0.4
  PITCH_PID_KD                  = 0.8
  PITCH_PID_KI                  = 0.0
```

```
  PITCH_PID_INTEGRAL_LIMIT      = 0

 //ZPID controller
  Z_TO_PITCH_PID_KP             = 0.12
  Z_TO_PITCH_PID_KD             = 0.0
  Z_TO_PITCH_PID_KI             = 0.004
  Z_TO_PITCH_PID_INTEGRAL_LIMIT = 0.05

  // Maximums
  MAXRUDDER   = 100
  MAXTHRUST   = 100
  MAXELEVATOR = 14
  MAXPITCH    = 30

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

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

   ArtifactFile = mines.art
   Sensor = FixedRadius
   Sensor_Radius = 10
}

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

   GridPoly = label,ArtGrid:-100,-50:100,30:
    100,-100:-100,-100
   GridSize = 5.0
   GridInit = .5
}

//--------------------------------------------
// 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_HOME @ "true"
   CustomKey = 7 : RETURN_HOME @ "false"
}
```

File: HUNTER2.moos

```
ServerHost = localhost
ServerPort = 9202
Simulator  = true

// Community name IS the vehicle name
Community  = HUNTER2

// MIT Sailing Pavilion
LatOrigin  = 42.3584
LongOrigin = -71.08745

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

  Run = MOOSDB @ NewConsole = true
  Run = iMarineSim @ NewConsole = false
  Run = pEchoVar @ NewConsole = false
  Run = pTransponderAIS @ NewConsole = false
  Run = pMOOSBridge @ NewConsole = false
   ~ pMOOSBridge_HUNTER2
  Run = pHelmIvP @ NewConsole = true
  Run = pMarinePID  @ NewConsole = false
  Run = pSensorSim  @ NewConsole = true
  Run = pArtifactMapper @ NewConsole = true
}

//--------------------------------------------
// iMarineSim config block

ProcessConfig = iMarineSim
{
  AppTick = 10
  CommsTick = 10

  MAX_TRANS_VEL = 5.0
  MAX_ROT_VEL   = 0.6
  START_X       = 10
  START_Y       = 10
  START_SPEED   = 0
  START_HEADING = 180

  //FLOAT_RATE    = 0.5
}

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

ProcessConfig = pEchoVar
{
  AppTick     = 20
  CommsTick   = 20

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


//--------------------------------------------
// pTransponderAIS config block
//
// Note: Outgoing processed every other cycle..
//       Incoming processed every cycle.

ProcessConfig = pTransponderAIS
{
  AppTick = 8
  CommsTick = 8

  VESSEL_TYPE   = KAYAK
}

//--------------------------------------------
// pMOOSBridge config block

ProcessConfig = pMOOSBridge_HUNTER2
{
  AppTick    = 10
  CommsTick  = 10

  // SHARE = [VAR] -> to-community @ to-host:to-port [VAR]

  SHARE = [AIS_REPORT_LOCAL]   ->
   shipside @ localhost:9123 [AIS_REPORT]
  SHARE = [STATION_CIRCLE]     ->
   shipside @ localhost:9123 [STATION_CIRCLE]
  SHARE = [RSTATION_CIRCLE]    ->
   shipside @ localhost:9123 [STATION_CIRCLE]
  SHARE = [VIEW_POLYGON]       ->
   shipside @ localhost:9123 [VIEW_POLYGON]
  SHARE = [VIEW_POINT]         ->
   shipside @ localhost:9123 [VIEW_POINT]
  SHARE = [RVIEW_POINT]        ->
   shipside @ localhost:9123 [VIEW_POINT]
  SHARE = [PROC_WATCH_SUMMARY] ->
   shipside @ localhost:9123 [PROC_WATCH_SUMMARY_HUNTER1]
  SHARE = [PROC_WATCH_EVENT]   ->
   shipside @ localhost:9123 [PROC_WATCH_EVENT_HUNTER1]
```

```
  SHARE = [GRID_CONFIG]        ->
   shipside @ localhost:9123 [GRID_CONFIG]
  SHARE = [GRID_DELTA_LOCAL]   ->
   shipside @ localhost:9123 [GRID_DELTA]
  SHARE = [GRID_DELTA_LOCAL]   ->
   HUNTER1 @ localhost:9201 [GRID_DELTA]
  SHARE = [GRID_DELTA_LOCAL]   ->
   HUNTER2 @ localhost:9202 [GRID_DELTA]

  SHARE = [AIS_REPORT_LOCAL] ->
   HUNTER1   @ localhost:9201 [AIS_REPORT]
  SHARE = [AIS_REPORT_LOCAL] ->
   HUNTER2   @ localhost:9202 [AIS_REPORT]
}

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

ProcessConfig = pHelmIvP
{
  AppTick    = 2
  CommsTick  = 2

  Behaviors  = hunter2.bhv
  Verbose    = verbose
  Domain     = course:0:359:360
  Domain     = speed:0:5:21

  ACTIVE_START = true
}

//----------------------------------------
// pMarine config block

ProcessConfig = pMarinePID
{
  AppTick    = 10
  CommsTick  = 10

  Verbose    = true

  ACTIVE_START  = true

  // 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


  DEPTH_CONTROL = true

  //Pitch PID controller
  PITCH_PID_KP                  = 0.4
  PITCH_PID_KD                  = 0.8
  PITCH_PID_KI                  = 0.0
  PITCH_PID_INTEGRAL_LIMIT      = 0

 //ZPID controller
  Z_TO_PITCH_PID_KP             = 0.12
  Z_TO_PITCH_PID_KD             = 0.0
  Z_TO_PITCH_PID_KI             = 0.004
  Z_TO_PITCH_PID_INTEGRAL_LIMIT = 0.05

  // Maximums
  MAXRUDDER   = 100
  MAXTHRUST   = 100
  MAXELEVATOR = 14
  MAXPITCH    = 30

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

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

   ArtifactFile = mines.art
   Sensor = FixedRadius
   Sensor_Radius = 10
}

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

   GridPoly = label,ArtGrid:-100,-50:100,30:
    100,-100:-100,-100
   GridSize = 5.0
   GridInit = .5
}
```

File: VIEWER.moos

```
ServerHost = localhost
ServerPort = 9123
Simulator  = true

// Community name IS the vehicle name
Community  = SHIPSIDE

// MIT Sailing Pavilion
LatOrigin  = 42.3584
LongOrigin = -71.08745

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

  Run = MOOSDB @ NewConsole = true
  Run = pMarineViewer @ NewConsole = true
  Run = iRemote @ NewConsole = true
}

//----------------------------------------
// pMarineViewer config block
ProcessConfig = pMarineViewer
{
  AppTick     =  6
  CommsTick   =  6
}

//----------------------------------------
// 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_HOME @ "true"
   CustomKey = 7 : RETURN_HOME @ "false"
}
```

APPENDIX VI
TUTORIAL BEHAVIOR

Line breaks in point list are provided for readability only.

File: hunter1.bhv

```
initialize   DEPLOY = true
initialize   RETURN = false
```

```
//--------------------------------------------
Behavior = BHV_Waypoint
{
  name      = bhv_waypt
  pwt       = 100
  condition = DEPLOY = true
  condition = RETURN = false
  repeat    = 10
  order     = normal
  lead      = 15.0

  speed     = 3.0
  radius    = 2.0
  nm_radius = 10.0
  points    = -100,-50:-100,-50:-100,-78:-53,-31:
  -5,-12:-93,-100:-65,-100:42,7:89,26:-37,-100:
  -8,-100:100,8:100,-20:20,-100:48,-100:100,-48:
  100,-76:76,-100

}
```

File: hunter2.bhv

```
initialize   DEPLOY = true
initialize   RETURN = false
initialize   ONSTATION = true

//--------------------------------------------
Behavior = BHV_SearchGrid
{
  name         = bhv_search_grid
  pwt          = 40
  time_horizon = 60
  precision    = high
  //log_ipf    = false
  searchgrid   = label,ArtGrid:-100,-50:100,30:
   100,-100:-100,-100@5.0,5.0@0.5
  timutif      = normal, 2, 2, 2, 100
  condition    = (DEPLOY=true) and (RETURN!=true)
  condition    = (ONSTATION=true)
}
```

APPENDIX VII
STARTUP SCRIPT

File: startall.sh

```
#/bin/sh
pAntler VIEWER.moos &
sleep 1;
pAntler HUNTER1.moos &
sleep 1;
pAntler HUNTER2.moos &
sleep 1;
```