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>

September 7, 2007

***Abstract*— This is the abstract.**

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).

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

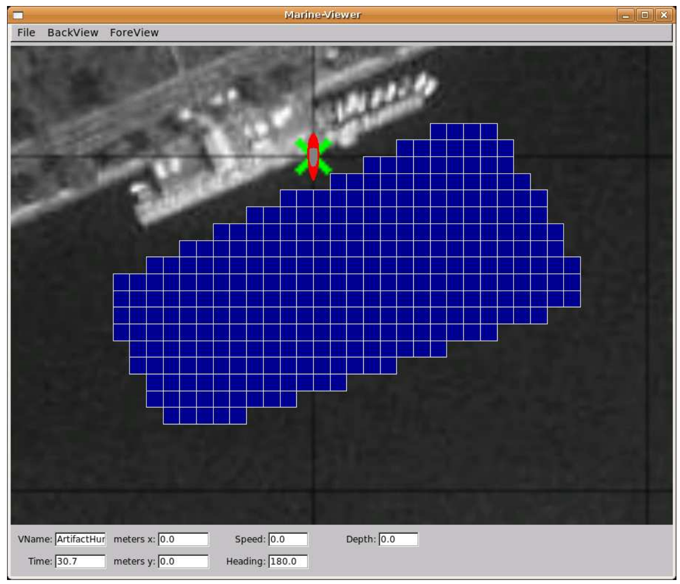

Fig. 2. A search grid defined over a search area.

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. 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. 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 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. 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. pSensorSim

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

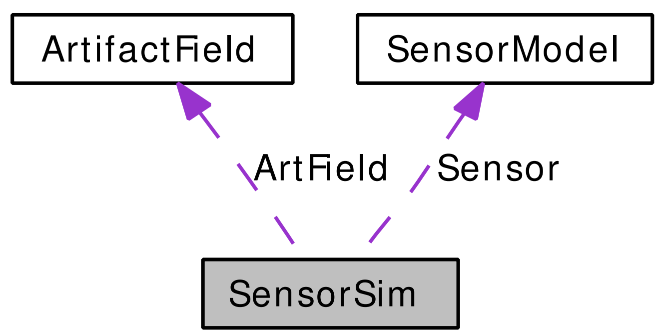

Fig. 3. A class diagram for pSensorSim

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
}
```

EXPLANATION OF PARAMETERS HERE

III. pARTIFACTMAPPER

Combining these two classes and creating a MOOSApp, we get Fig. 4.

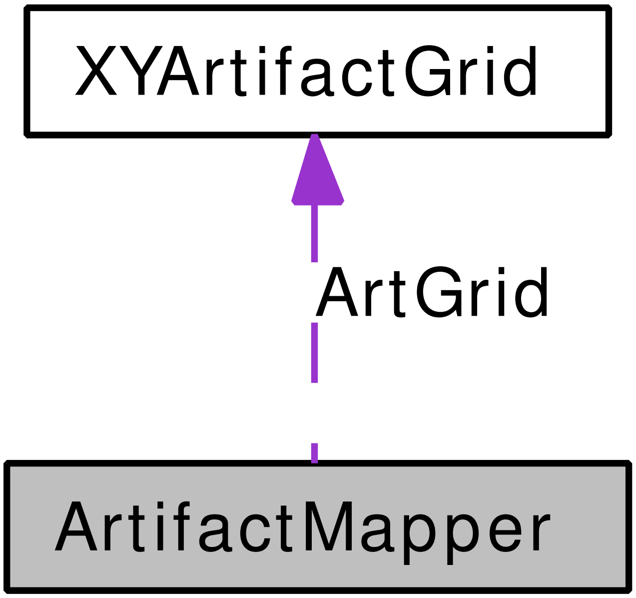

Fig. 4. A class diagram for pArtifactMapper

IV. BHV_SEARCHGRID

V. EXAMPLE MISSIONS

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

Listing A.1 The MOOS File for Examples 1-3

Filename: alpha.moos

```
ServerHost = localhost
ServerPort = 9000
Simulator  = true
Community  = alpha
LatOrigin  = 42.3584
LongOrigin = -71.08745

//------------------------------------------
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 = pMarinePID       @ NewConsole = true
  Run = pMarineViewer    @ NewConsole = true
  Run = pHelmIvP         @ NewConsole = true
  Run = iRemote          @ NewConsole = true
}

//------------------------------------------
ProcessConfig = iMarineSim
{
  AppTick     = 4
  CommsTick     = 4
  MaxTransVel   = 3.0
  MaxRotVel     = 0.6
  StartLon      = 10
  StartLat      = -40
  StartSpeed    = 0
  StartHeading  = 180
}

//------------------------------------------
ProcessConfig = pHelmIvP
{
  AppTick    = 4
  CommsTick  = 4
  Domain     = course:0:359:360
  Domain     = speed:0:3:16

  Behaviors  = foobar.bhv
  VERBOSE    = terse
}

//------------------------------------------
ProcessConfig = pMarinePID
{
  AppTick    = 4
  CommsTick  = 4
  Verbose    = true

  DEPTH_CONTROL = false
  MAXRUDDER     = 100
  MAXTHRUST     = 100

  YAW_PID_KP                = 0.5
  YAW_PID_KD                = 0.0
  YAW_PID_KI                = 0.0
  YAW_PID_INTEGRAL_LIMIT    = 0.07

  SPEED_PID_KP              = 1.0
  SPEED_PID_KD              = 0.0
  SPEED_PID_KI              = 0.0
  SPEED_PID_INTEGRAL_LIMIT  = 0.07
  SPEED_FACTOR              = 20
}

//------------------------------------------
ProcessConfig = iRemote
{
   CustomKey = 1 : HELM_VERBOSE @ "verbose"
   CustomKey = 2 : HELM_VERBOSE @ "terse"
   CustomKey = 3 : HELM_VERBOSE @ "quiet"
```

```
   CustomKey = 4 : DEPLOY       @ "true"
   CustomKey = 5 : RETURN       @ "true"
}

//------------------------------------------
ProcessConfig = pLogger
{
  AppTick       = 20.0
  CommsTick     = 20.0
  File          = alpha
  PATH          = ./datafiles
  SyncLog       = true @ 0.2
  AsyncLog      = true
  FileTimeStamp = true
  Log           = NAV_X     @ 0.1
  Log           = NAV_Y     @ 0.1
  Log           = NAV_Yaw   @ 0.1
  Log           = NAV_Speed @ 0.1
}

//------------------------------------------
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
}

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