uFldObstacleSim: Simulating Obstacles

June 2020
July 2021 - minor additions

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Overview

The `uFldObstacleSim` application is a tool for simulating obstacles, and obstacle sensor output. Ground truth position and size of obstacles are read from a pre-generated *obstacle file*. The simulation generates information in one of two modes, in one of two manners:

- Obstacle polygons are simply published and shared to all vehicles, or
- Simulated sensor data, similar to Lidar points on the obstacle, are published and shared to the vehicles. Downstream apps are then left with the job of inferring the obstacle from the sensor data.

When this simulator is operating in the first mode, it publishes information to the variable GIVEN_OBSTACLE. In the second mode, it publishes the information to the variable TRACKED_FEATURE.

The simulator also supports two different modes of obstacle generation:

- By default, the obstacles are simply read in from the obstacle file and never change.
- A second mode is supported where the obstacles are periodically regenerated with the same configuration parameters used for creating the original obstacle file.

A key parameter in regeneration is the obstacle region, a polygon within which all randomly generated obstacles are guaranteed to reside within. During a simulation, we want to take care that obstacle regeneration does not take place while any vehicles are in the obstacle region. Otherwise an obstacle could be generated in a position currently occupied by, or just in front of a vehicle. The ensuing unavoidable collision would skew the test results. The obstacle simulator therefore will monitor vehicle positions and only regenerate obstacles when all vehicles are a safe distance from the obstacle region. The vehicle locations are known to the simulator from the NODE_REPORT messages received from the vehicles.

The setup overview is depicted in Figure 1:

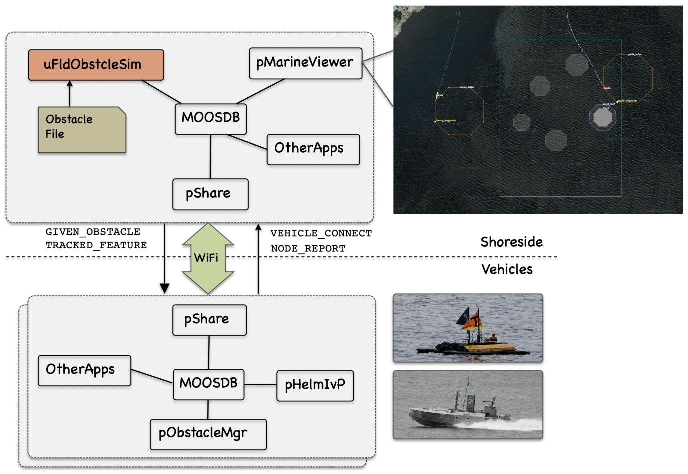

Figure 1: **The Obstacle Simulator:** The obstacle simulator resides on the shoreside and ingests and maintains ground truth obstacle state. Information is fed to each connected vehicle. Typically the primary consumer on the vehicle side is the obstacle manager, pObstacleMgr.

2 A Quick Start Guide to Using uFldObstacleSim

To get started, we (a) point you to an example mission using uFldObstacleSim, (b) lay out the minimum uFldObstacleSim configuration components, i.e., those which do not have default values, and (c) discuss the structure of a simple obstacle file representing the ground-truth set of obstacles used by the simulator.

2.1 A Working Example Mission - the Bo Alpha Mission

The example mission is referred to as the Bo Alpha example mission and may be found and launched in the moos-ivp distribution with:

```
$ cd moos-ivp/ivp/missions/m34_bo_alpha
$ ./launch.sh 10
```

This launches the simulation with time warp 10. The time warp may be adjusted to suit your preference and is bounded above by your computer's processing capability. After launching and hitting the deploy button, you should see something similar to Figure 2.

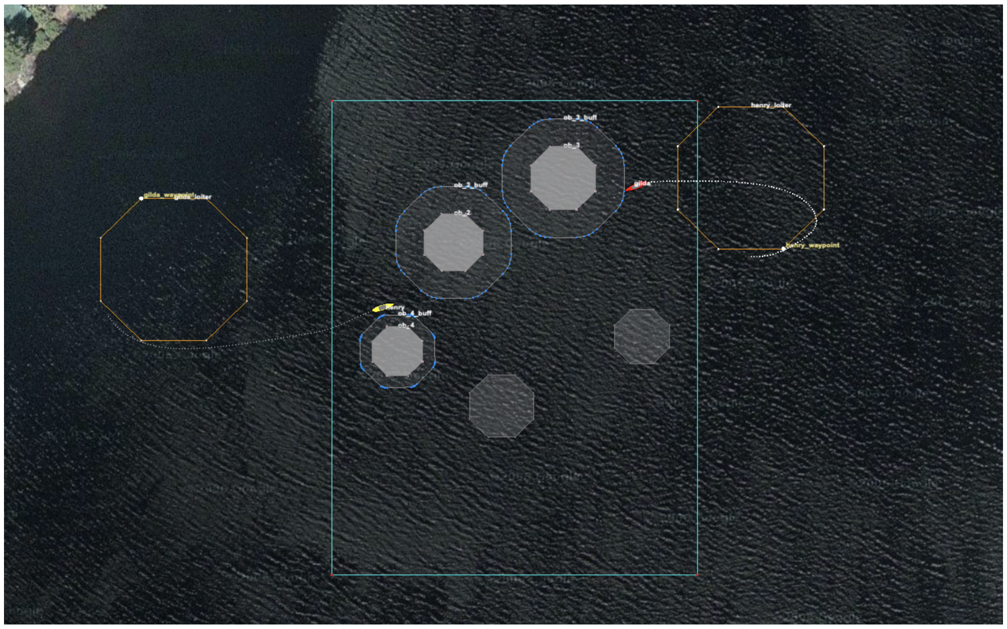

Figure 2: **The Bo Alpha Mission:** The Bo Alpha example mission involves a two vehicles each traversing East-West through an obstacle field created by the simulator. The helm on both vehicles is performing obstacle avoidance and COLREGS based collision avoidance with the other vehicle. The true obstacle is the inner polygon. When the vehicle is actively avoiding the obstacle, the true obstacle is rendered more opaque, and the buffer region around the obstacle is rendered.
Video:(1:01): https://vimeo.com/424523746

There are a few notable components of this simulation that comprise the nature of the mission:

- The uFldObstacleSim: Running on the shoreside, distributes knowledge of the obstacles to each vehicle.
- The obstacles.txt file read in by the uFldObstacleSim obstacle simulator.
- The gen_obstacles command-line app that generated the obstacles.txt file, used by the simulator. This app chooses random obstacle locations within a polygon and ensures a given minimum spacing between obstacles.

- pObstacleMgr app running on each vehicle that receives the obstacle knowledge from the obstacle simulator and maintains a set of known obstacles. It will also generate obstacle alerts which result in spawned obstacle avoidance behaviors in the helm.

It may also be worth considering, in Figure 2, where all the visual artifacts shown in the pMarineViewer snapshot are coming from:

- The uFldObstacleSim app is publishing the large square rectangle, which is the region from which the random obstacles were originally chosen from. It is also publishing all five ground-truth obstacles. The three north-west obstacles have other things being rendered over them, but the two south-east obstacles are the ones published by uFldObstacleSim.
- The Avoid Obstacle behavior in the helm also publishes the obstacle polygon when the behavior is active, but it publishes a request to render the polygon with more opacity. Thus the three north-west polygons show that at least one of the vehicles is actively avoiding it. The behavior also published the buffer polygon around the ground truth obstacle, with less opacity. And of course the behavior is also publishing the loiter polygon to the far east and west to which each is transiting through the obstacle field.
- Not shown in the figure, but when the two vehicles are close to each other and also actively engaging in collision avoidance, each vehicle's collision avoidance behavior will render a line between vehicles, with color varying with the behavior's priority weight.

2.2 A Bare-Bones Example uFldObstacleSim Configuration

Listing 1 below shows a bare-bones configuration. Line 3 names a obstacle file, containing the ground truth description of objects in the field. This format is described in Section 2.3, and an example obstacle file is in the same directory as the Bo Alpha example mission. There are several more configuration parameters supported by uFldObstacleSim, but they all have default values, accepted in this simple simulation. The other parameter values are discussed later.

Listing 2.1: Example bare-bones configuration of uFldObstacleSim.

```
1  ProcessConfig = uFldObstacleSim
2  {
3    obstacle_file = obstacles.txt
4  }
```

2.3 A Simple Obstacle File

In the Bo Alpha example mission, two vehicles traverse an obstacle field comprised of five obstacles. These obstacles were chosen randomly by the gen_obstacles command line utility. The results were stored in the `obstacles.txt` file read in by the obstacle simulator. This file was named in the obstacle simulator config block in Listing 1 above. The gen_obstacles utility will choose obstacles while ensuring (a) each obstacle is within a given polygon area, and (b) a minimum configurable separation distance between obstacles. This command-line tool is discussed in Section 2.4.

An obstacle file is comprised of:

- The first line, a comment, showing the exact command that generated the file,
- The command line parameters on a separate line,
- The polygon for each obstacle, with a unique label.

The obstacle file for the Bo Alpha mission is given below.

```
# gen_obstacles --poly=30,-20:30,-140:120,-140:120,-20 --amt=5 --min_size=1 --max_size=6 \
   --min_range=20 --meter
region    = pts={30,-20:30,-140:120,-140:120,-20}
min_range = 20
min_size  = 6
max_size  = 10
poly = pts={107,-101:112,-106:112,-113:107,-118:100,-118:95,-113:95,-106:100,-101},label=ob_0
poly = pts={50,-30:53,-33:53,-38:50,-42:45,-42:41,-38:41,-33:45,-30},label=ob_1
poly = pts={82,-65:87,-70:87,-76:82,-81:76,-81:71,-76:71,-70:76,-65},label=ob_2
poly = pts={59,-101:64,-105:64,-112:59,-116:53,-116:48,-112:48,-105:53,-101},label=ob_3
poly = pts={107,-38:112,-43:112,-49:107,-54:100,-54:96,-49:96,-43:100,-38},label=ob_4
```

2.4 Generating an Obstacle File

The `gen_obstacles` command line tool will generate a list of randomly generated obstacle polygons, given a polygon region. The tool will guarantee that the obstacle will completely reside in the region, and will guarantee a minimal separation between obstacles.

For example, the below command will generate the obstacle file shown in the above section, with the initial five obstacles of the Bo Alpha mission. The `--poly` parameter indicates the obstacle *region*. This region must be convex, and the utility will produces as many obstacles as possible up to the amount specified with the `--amt` parameter.

```
$ gen_obstacles --poly=30,-20:30,-140:120,-140:120,-20 --amt=5
  --min_size=1 --max_size=6 --min_range=10 --meter
```

Each obstacle is an octagon with a random radius no smaller than that given by the `--min_size` parameter, and no greater than that given by the `--max_size` parameter. The obstacles will be placed with a guaranteed minimum separation to the other obstacles given by the `--min_range` parameter. Finally, the `--meter` parameter, will round all obstacle vertices to the nearest meter if desired, instead of the default value of `0.1` meters.

3 Dynamic Resetting of Ground Truth Obstacles

The obstacle manager supports an option to periodically reset the ground truth size and location of obstacles. This allows simulations to test against a wider variety of obstacle situations without the need to re-start the simulation. The dynamic reset capability ensures that the reset only occurs only when all known vehicles are outside the obstacle field, to avoid generating a new obstacle too close to (or on) a vehicle and thus creating an unavoidable collision.

3.1 Parameters for Enabling Dynamic Resetting

The dynamic reset occurs on a schedule. After an elapsed period of time, given by the reset_interval parameter, a reset will occur as soon as all the vehicles are outside the obstacle region. All vehicles must be not only outside the obstacle region but also a given distance away from the obstacle region given by the reset_range parameter. Of course, if all vehicles never happen to be outside the obstacle region simultaneously, then the reset will never occur. The mission must be otherwise constructed, as with the Bo Alpha mission, to ensure the vehicles periodically all exit the obstacle region simultaneously.

Here is an example setting:

```
reset_interval = 300
reset_range    = 20
```

The reset_interval units are in seconds, and the reset_range units are in meters. The default value for reset_interval is -1, indicating that resets are disabled. The default value for reset_range is 10 meters.

3.2 MOOS Variable for Enabling Dynamic Resetting

If you would like to reset the obstacle field on demand, by poking a MOOS variable, this can be done. By publishing UFOS_RESET = now, the obstacle simulator will schedule an obstacle reset to occur as soon as possible. As discussed above, the reset will only actually occur as soon as all vehicles have exited the obstacle region, sufficiently far away from the obstacle region given by the reset_range parameter.

3.3 Enabling Dynamic Resetting in the Bo Alpha Mission

Dynamic obstacle resetting is not enabled by default in the Bo Alpha mission. It can be enabled by launching with the -d option on the command line:

```
$ cd moos-ivp/ivp/missions/m34_bo_alpha
$ ./launch.sh 10 -d
```

This additional flag is supported in the launch.sh script to add the following two lines to the uFldObstacleSim configuration block:

```
reset_interval = 100
reset_range    = 10
```

Every 100 seconds, when the two vehicles are at least 10 meters outside the obstacle region, the obstacles will be reset. Figure 3 shows two such random obstacle configurations, and the associated video link shows the dynamic mission in action.

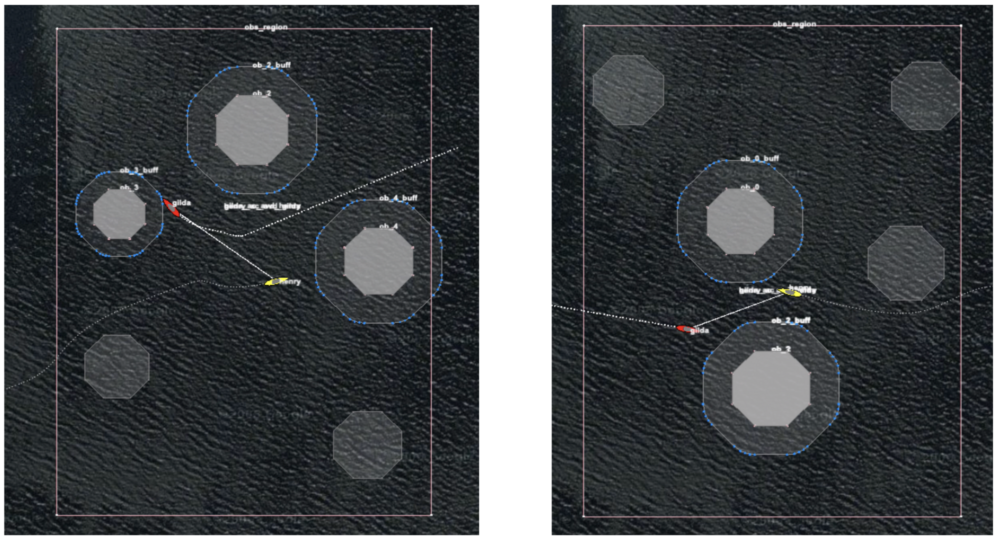

Figure 3: **The Dynamic Bo Alpha Mission:** The Bo Alpha is launched with dynamic obstacle reset enabled. Roughly each time the vehicles reach their east west loiter positions, the obstacle simulator reset the obstacles. This produces a different challenge each time the vehicles traverse through the obstacle field.
`Video:(0:54): https://vimeo.com/425156397`

4 Simulating Sensor Data from Ground Truth Obstacles

The obstacle simulator supports a second mode of distributing obstacle information to vehicles. In the default mode discussed so far, the ground truth obstacle position and location are sent directly to the vehicles in the `GIVEN_OBSTACLE` MOOS variable. For the vehicle there is no guesswork, and obstacle avoidance is conducted with perfect knowledge of the obstacles.

In the second mode, the *points* mode, the obstacle simulator generates a steady stream of random points inside the ground truth obstacles. It then sends these points to the vehicle in the MOOS variable `TRACKED_FEATURE`. Here is an example publication:

```
TRACKED_FEATURE = x=100,y=-49,key=ob_4
```

The message contains both the point location, and a unique ID associated with the ground truth obstacle. So this simulatad sensor data is still pretty artificially simplistic - there are no false points because all points generated by the simulator do reside within the ground truth obstacle polygon. And by including a key, clustering of points is already done and perfect.

4.1 Enabling the Points Sensor Data Mode

The *points* mode is disabled by default. It is enabled by setting the `post_points` configuration parameter to `true`:

```
post_points = true
rate_points = 5
```

The `rate_points` parameter sets the number of points generated, per obstacle, per iteration of the simulator. The default value is 5. When the points mode is enabled, the obstacle simulator does not post the ground truth obstacle information to GIVEN_OBSTACLE. However, it is still posted to KNOWN_OBSTACLE. Typically uField sharing is configured to *not* share KNOWN_OBSTACLE to the vehicles. However it may still be useful for other apps in the shoreside community to have access to ground truth obstacle information. For example the uFldCollObDetect app runs in the shoreside and monitors vehicles for collisions with obstacles, so it needs access to ground truth obstacle information.

4.2 Generation of Simulated Sensor Points

Simulated sensor points are generated by calculating random line-of-sight points on the edge of the obstacle polygon from the direction of the vehicle. Sensor points are generated per vehicle and shared only with the relevant vehicle as in Figure 4.

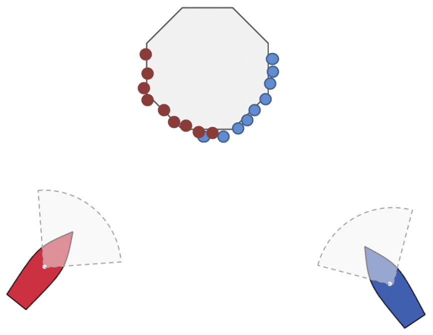

Figure 4: **Vehicle-centered line-of-site sensor:** Sensor points are based on line of site, with different sensor points generated for each vehicle depending on the bearing of the obstacle to the vehicle. This is not currently implemented in the uFldObstacleSim app.

Recall that each point corresponds to two publications. One publication, to the variable TRACKED_FEATURE_ABE, is generated and shared only to the vehicle abe. The other publication is to the variable VIEW_POINT which published locally on the shoreside for the benefit of rendering in a GUI app such as pMarineViewer. The color of the points is based on the color of the vehicle, derived from NODE_REPORT messsages received on the shoreside from each vehicle.

In the event that uFldObstacleSim is run in a headless simulation, i.e., no need for visuals, the publication of VIEW_POINT can be disabled with the configuration post_visuals=false.

5 Obstacle Expiration

There are three notions or reasons why we may want to consider the *expiration* of obstacles.

- The robot or vehicle has moved far away from the obstacle and we no longer care about it, or
- The obstacle never existed to begin with, but perhaps briefly it appeared to exist due to sensor noise, or
- The obstacle is still close and is real, but perhaps a new identifier was mistakenly created for the same obstacle.

Each of these things happen in practice, and the downstream apps that manage and reason about obstacles need to deal with these issues. So the obstacle simulator has the capability to replicate the expiration of an obstacle to enable testing of the downstream apps.

The expiration policy of the two primary obstacle modes of the simulator are discussed here:

- The *points* mode, where the simulator publishes sensor points in the form of the `TRACKED_FEATURE` variable, and
- The *ground-truth* mode where the simulator publishes the ground truth obstacles in the form of the `GIVEN_OBSTACLE` variable.

5.1 Case 1 - Expiration of Sensor Points

The simplest of the two modes is the *points* mode, perhaps because elements in a sensor data stream are usually considered to be ephemeral - as things change, so does the data. In the *points* mode, data is published to the `TRACKED_FEATURE` variable. These postings are regarded as similar to LIDAR points. Although the simulator generates them randomly on or in the obstacle, they at some point will either cease, or evolve position, due to:

- The obstacle goes out of range from the vehicle
- The obstacle moves or drifts, and thus so do the points
- The obstacle stops producing points because the simulator deletes the obstacle
- The obstacle stops producing points because in the real world the obstacle actually didn't exist but was perhaps produced by a wave or some other noise

Figure 5 conveys the lifespan of sensor points on an obstacle as a vehicle approaches, passes and leaves behind an obstacle.

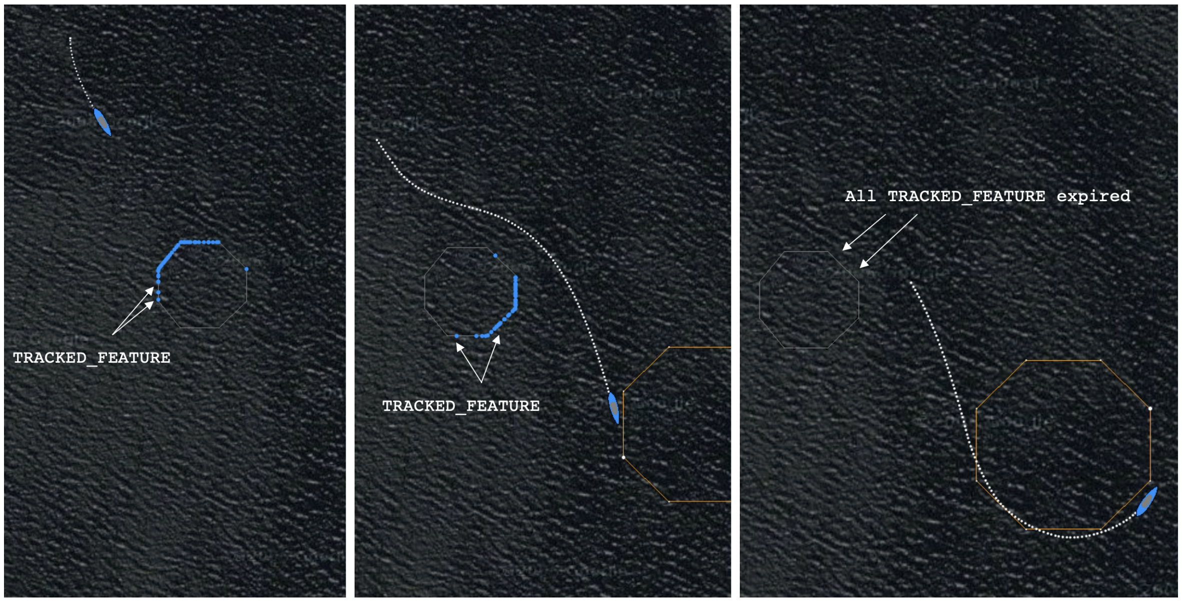

Figure 5: **Expiration of Sensor Points:** Sensor points are generated on the edge of the obstacle as the vehicle comes in range (left). As long as the vehicle remains in range, regardless of the relative bearing from the vehicle to the obstacle, sensor points on the line of site to the obstacle will continue to be generated (middle). As the vehicle open range to the obstacle the sensor points will cease to be generated (right).

5.2 Case 2 - Expiration of Ground Truth Obstacles

In *ground-truth* mode, where the simulator is publishing the actual obstacle polygon rather than simulated sensor points, the simulator publishes the polygon information in the form of:

- `VIEW_POLYGON`: for consumption by a GUI app like `pMarineViewer`.
- `KNOWN_OBSTACLE`: for consumption by other shoreside apps that need access to ground truth obstacles such as `uFldCollObDetect`.
- `GIVEN_OBSTACLE`: for sharing to the vehicles and consumption by the obstacle manager `pObstacleMgr`.

In each of these cases, there is a need to "forget" about an obstacle after time. So each obstacle message contains a `duration` field, in seconds. While the obstacle is relevant, the obstacle simulator will periodically publish to these variables, each time with same duration. The consumers presumably reset their duration clocks each time a new message is received. The obstacle simulator will publish only periodically, but often enough such that the obstacle will endure indefinitely if the periodic refresh publications continue.

The `refresh_interval` determines how often `uFldObstacleSim` re-publishes the current ground-truth obstacles. A duration is associated with each obstacle, so typically the refresh should occur before an obstacle expires (if the goal is persistence). The duration is set with two parameters, `min_duration` and `max_duration`. A random duration will be chosen between these two values.

The default settings for `refresh_interval`, `min_duration`, and `max_duration` are all `-1`. When `refresh_interval` is not set, the obstacle simulator will never refresh (republish) the obstacle postings unless one or more vertices change. When the obstacle duration is not set, they we never expire in any of the consumer apps unless they are cleared or erased via other methods.

6 Configuration Parameters of uFldObstacleSim

The following parameters are defined for uFldObstacleSim. For some parameters, more detailed description are provided in other sections. Parameters having default values are indicated so.

Listing 6.2: Configuration Parameters for uFldObstacleSim.

obstacle_file:	A file with obstacle position and size location. Sections 2.3 and 2.4.
poly_vert_color:	Color of obstacle polygon vertices. The default is "gray50".
poly_edge_color:	Color of obstacle polygon edges. The default is "gray50".
poly_fill_color:	Color of obstacle polygon interior. The default is "white".
poly_label_color:	Color of obstacle polygon labels. The default is "invisible".
poly_vert_size:	Size of rendered obstacle polygon vertices. The default is 1.
poly_edge_size:	Size of obstacle polygon edges. The default is 1.
poly_transparency:	Transparency of rendered obstacle polygons. The default is 0.15.
draw_region:	If true, draw the obstacle region. The default is true.
region_edge_color:	Color of obstacle polygon edges. The default is "gray50".
region_edge_color:	Color of obstacle region edges. The default is "white".
post_points:	If true, sensor points are generated rather than ground truth obstacle polygons. The value false. Section 4.1.
rate_points:	When post_points is true, this parameter sets the rate of point generation. The default is 5 points, per obstacle, per iteration. Section 4.1.
min_duration:	If non-negative, set a random duration for each obstacle no lower than this value. The default is -1. Section 5.2.
post_visuals:	If true, visual posts to VIEW_POINT and VIEW_POLYGON are generated. The default is true. Section 4.2.
refresh_interval:	If non-negative, publications to GIVEN_OBSTACLE will be reposted evern N seconds where N is the value of this parameter. The default is -1. Section 5.2.
reset_interval:	Time, in seconds, between automatic reseting of obstacle locations. The default is -1, indicating disabled resetting. Section 3.1.
reset_range:	Distance, in meters, that all vehicles need to be outside the obstacle region in order for an obstacle reset to be allowed. The default is 10 meters. Section 3.1.
reuse_ids:	If false, each time the obstacle field is reset, a unique set of obstacle IDs, i.e., labels, will be generated for the newly generated obstacles. The default is true. Section 3.1.
sensor_range:	The range to an obstacle at which simulated LIDAR point will be generated. The default is 50. Section 4.2.

6.1 An Example MOOS Configuration Block

To see an example MOOS configuration block, enter the following from the command-line:

```
$ uFldObstacleSim --example or -e
```

This will show the output shown in Listing 3 below.

Listing 6.3: Example configuration for `uFldObstacleSim`.

```
===============================================================
uFldObstacleSim Example MOOS Configuration
==============================================================

ProcessConfig = uFldObstacleSim
{
  AppTick   = 4
  CommsTick = 4

  obstacle_file    = obstacles.txt
  poly_vert_color  = color    (default is gray50)
  poly_edge_color  = color    (default is gray50)
  poly_fill_color  = color    (default is white)
  poly_label_color = color    (default is invisible)

  poly_vert_size    = 1       (default is 1)
  poly_edge_size    = 1       (default is 1)
  poly_transparency = 0.15    (default is 0.15)

  region_edge_color = color   (default is gray50)
  region_vert_color = color   (default is white )

 draw_region       = true    (default is true)
 region_edge_color = color   (default is gray50)
 region_vert_color = color   (default is white)

 post_points      = true     (default is false)
 rate_points      = 5        (default is 5)
 point_size       = 5        (default is 2)

 min_duration     = 10       (default is -1)
 max_duration     = 15       (default is -1)
 refresh_interval = 8        (default is -1)

 reset_interval   = -1       (default is -1)
 reset_range      = 10       (default is 10)

 reuse_ids        = true     (default is true)
 sensor_range     = 50       (default is 50)

 app_logging  = true  // {true or file} By default disabled

 post_visuals = true  // {true or false} By default true
}
```

7 Publications and Subscriptions for uFldObstacleSim

The interface for uFldObstacleSim, in terms of publications and subscriptions, is described below. This same information may also be obtained from the terminal with:

```
$ uFldObstacleSim --interface or -i
```

7.1 Variables Published by uFldObstacleSim

The primary output of uFldObstcleSim to the MOOSDB is posting of sensor reports, visual cues for the sensor reports, and visual cues for the hazard objects themselves.

- APPCAST: Contains an appcast report identical to the terminal output. Appcasts are posted only after an appcast request is received from an appcast viewing utility. Section 8
- GIVEN_OBSTACLE: A ground truth obstacle, same as, KNOWN_OBSTACLE, but published separately. Typically this variable will be shared to the the vehicles.
- KNOWN_OBSTACLE: A ground truth obstacle, same as, GIVEN_OBSTACLE, but published separately. Typically shoreside apps will register for this variable for knowledge of ground truth obstacles.
- TRACKED_FEATURE: A single point related to a sensor measurement related to a ground truth obstacle, and key associated with that obstacle.
- VIEW_POLYGON: A visual artifact for rendering a a ground truth obstacle polygon.

Example postings:

```
KNOWN_OBSTACLE  = pts={48,-77:52,-80:52,-86:48,-89:43,-89:39,-86:39,-80:43,-77},  \
                  label=ob_4,duration=5
GIVEN_OBSTACLE  = pts={48,-77:52,-80:52,-86:48,-89:43,-89:39,-86:39,-80:43,-77},  \
                  label=ob_4,duration=5
TRACKED_FEATURE = x=100,y=-49,key=ob_4
VIEW_POLYGON    = pts={48,-77:52,-80:52,-86:48,-89:43,-89:39,-86:39,-80:43,-77},  \
                  label=ob_4,label_color=invisible,edge_color=gray50,vertex_color=gray50, \
                  fill_color=white,vertex_size=1,edge_size=1,fill_transparency=0.15
```

7.2 Variables Subscribed for by uFldObstcleSim

The uFldObstcleSim application will subscribe for the following MOOS variables:

- APPCAST_REQ: A request to generate and post a new apppcast report, with reporting criteria, and expiration.
- VEHICLE_CONNECT: A message from the vehicle indicating its presence in the simulation. This will prompt a (re)publication and sharing of ground truth obstacles to the vehicle.
- UFOS_RESET: A request to the simulator to reset the obstacle field immediately, or as soon as all vehicles have safely cleared the obstacle field. 3.2.
- NODE_REPORT: A report on a vehicle location and status.

Example postings:

```
UFOS_RESET = true
VEHICLE_CONNECT = true
```

Command Line Usage of uFldObstcleSim

The uFldObstcleSim application is typically launched as a part of a batch of processes by pAntler, but may also be launched from the command line by the user. To see command-line options enter the following from the command-line:

```
$ uFldObstacleSim --help or -h
```

This will show the output shown in Listing 4 below.

Listing 7.4: Command line usage for the UFldObstacleSim *tool.*

```
==========================================================
Usage: uFldObstcleSim file.moos [OPTIONS]
==========================================================

Options:
  --alias=<ProcessName>
      Launch uFldObstcleSim with the given process
      name rather than uFldObstcleSim.
  --example, -e
      Display example MOOS configuration block.
  --help, -h
      Display this help message.
  --interface, -i
      Display MOOS publications and subscriptions.
  --version,-v
      Display release version of uFldObstcleSim.
  --verbose=<setting>
      Set verbosity. true or false (default)

Note: If argv[2] does not otherwise match a known option,
      then it will be interpreted as a run alias. This is
      to support pAntler launching conventions.
```

8 Terminal and AppCast Output

The UFldObstacleSim application produces some useful information to the terminal and identical content through appcasting. An example is shown in Listing 5 below. On line 2, the name of the local community, typically the shoreside community, is listed on the left. On the right, "0/0(204) indicates there are no configuration or run warnings, and the current iteration of UFldObstacleSim is 204. Lines 5-9 show the obstacle file configuration. Lines 10-12 indicate whether ground truth obstacles or simulate sensor points will be generated. Lines 13-15 indicate the range of random durations associated with each obstacle. Lines 16-18 indicate whether the simulator will periodically reset the obstacle locations and if so, how often.

Lines 20-28 reveal the current state of the simulator. Lines 21-22 show how many times the polygon obstacles have been posted as viewable polygons and how many times as given polygons respectively. Lines 23-28 show the state with respect to possibly resetting the obstacles. Line 24 shows the distance between the obstacle region to the closest vehicle. The zero in this case indicates that one or more vehicles are inside the obstacle region. Lines 25-28 show the progress toward potentially resetting the obstacles.

Lines 31-38 show key information related to each obstacle. Column 3 will show total simulated sensor points published per obstacle, if in the *points* mode. Column 4 will be used when given obstacles are published, and incremented each time they are published.

Listing 8.5: Example `UFldObstacleSim` *console output.*

```
 1  ===================================================================
 2  uFldObstacleSim shoreside                                 0/0(204)
 3  ===================================================================
 4  ================================
 5  Config (Obstacles)
 6    Obstacles: 5
 7    MinRange:  20
 8    MinSize:   6
 9    MaxSize:   10
10  Config (Points)
11    Post Points:   false
12    Rate Points:   5
13  Config (Duration)
14    Min Duration: 400.0
15    Max Duration: 500.0
16  Config (Reset)
17    Reset Range:  10
18    Reset_Interv: -1
19  ================================
20  State
21    Viewables Posted: 1
22    Obstacles Posted: 2
23  State (resetting)
24    Min Poly Range: 0
25    Reset Pending:  false
26    Newly Exited :  false
27    Reset Tstamp :  23867167818
28    Reset Total  :  0
29
30
31  Obs   Obs       Points     Given
32  Key   Duration  Published  Published
33  ----  --------  ---------  ---------
34  ob_0  468.1     0          2
35  ob_1  470.1     0          2
36  ob_2  478.2     0          2
37  ob_3  451.7     0          2
38  ob_4  445.3     0          2
```