pObstacleMgr: Managing Vehicle Belief State of Obstacles

June 2020

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

`project-pavlab/appdocs/app_pobstaclemgr`

1 Overview

The pObstacleMgr is designed to reason about obstacles and provide coordinated alerts and updates to the helm for spawning and adjusting obstacle avoidance behaviors. The obstacle manager reasons about *given* obstacles, with prior known locations, loaded at launch time and may include buoys, rocky outcrops, bridge pylons and so on. It also reasons about *sensed* objects that may be derived from LIDAR point clouds, or other sensor sources. The obstacle manager will manage both the mission-loaded and dynamic incoming data to maintain a single list of obstacles. Depending on the robot range to the obstacle, the obstacle manager will produce alerts for coordination with obstacle avoidance behaviors. And in the case of dynamically sensed obstacles, it will continually update the position and shape of the obstacle to previously spawned obstacle avoidance behaviors.

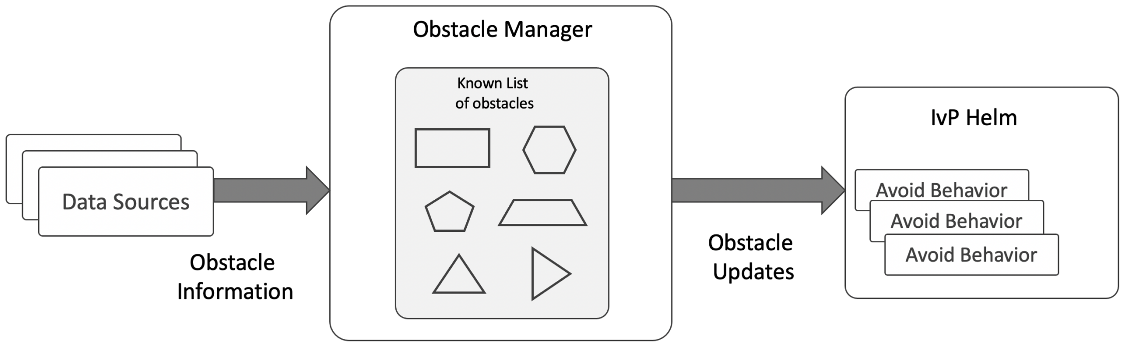

Figure 1: **The Obstacle Manager**: Obstacle information from a variety of sources is received by the obstacle manager, which maintains a list of known obstacles. This is modified with time both in terms of updating the obstacle locations as well as deleting stale obstacles. Continuous updates are posted and consumed by the helm to enable the spawning of avoidance behaviors.

The obstacle manager exists and sits between the sensor stream and the helm for two reasons. (1) This arrangement allows the helm to remain passively postured with respect to behavior spawning of obstacle avoidance behaviors. The helm will spawn behaviors based on incoming events from the obstacle manager. This is implemented in the helm in a manner that is consistent with any other type of behavior or sensor stream. Nothing special was implemented in the helm to handle obstacle information, other than the `AvoidObstacle` behavior. (2) The nature of the obstacle manager may change over time as different sensors, information sources, or outlier rejection algorithms are available. None of these changes will require a change to the helm or its behaviors.

2 Using the Obstacle Manager

To use the obstacle manager there are a few steps and considerations.

- The pObstacleMgr app must be run on the vehicle, by adding it to the Antler block for the vehicle. An example configuration block is given in Section 4.1.
- Obstacle information must be fed to the obstacle manager from either one of three sources. Either it (a) arrives from a sensor-based source as labeled points as decribed in Section 2.1.1, or (b) arrives as convex polygon obstacle mail message in the variable `GIVEN_OBSTACLE`, or (c)

the obstacle size and position information is provided at launch time from a set of obstacles known a priori, listed in the configuration file with the parameter `given_obstacle`, as described in Section 2.1.2.

- The helm is configured to use an obstacle avoidance behavior in a templating mode, allowing new behaviors to spawn based on output from the obstacle manager. See Section 2.2.

2.1 Obstacle Manager Input Sources

The obstacle manager may be populated with obstacle information in one of three methods as shown below. Regardless of the source, all obstacles are maintained as a list of known obstacles by the obstacle manager. And each time there is a change in shape of a known obstacle, an update is posted to the obstacle manager consumers, e.g., the helm.

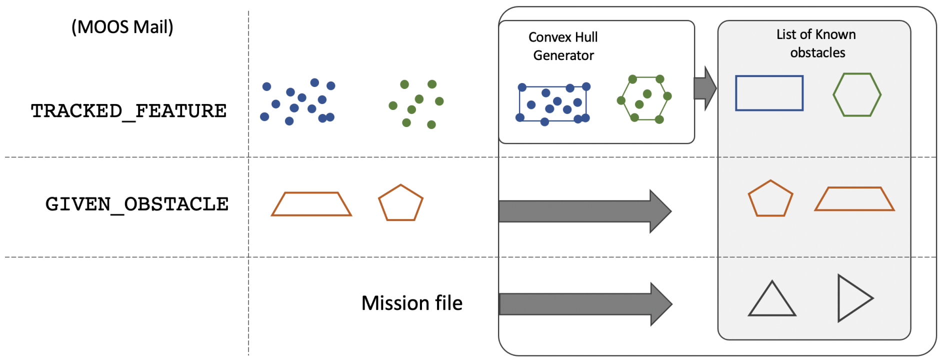

Figure 2: **Obstacle Manager Input**: Currently supported obstacle manager input comes from one of three sources. (1) As individual LIDAR points with grouping information, (2) as polygon obstacles via incoming mail, typically followed by updates for each obstacle, or (3) as polygon obstacles read from a mission configuration file.

In this section, the obstacle manager sources are described in more detail, along with how the obstacle manager processess incoming information.

2.1.1 Tracked Feature Inputs

The obstacle manager is designed to work with one or more other applications producing tracked features. These tracked features may be points generated by a LIDAR, or some other sensor. The obstacle simulator, `uFldObstacleSim`, has a mode that supports generation of simulated LIDAR points. The obstacle manager subscribes for the MOOS variable `TRACKED_FEATURE`, of the form:

```
TRACKED_FEATURE  = x=23.2,y=19.8,label=47
TRACKED_FEATURE  = x=22.9,y=18.2,label=47
```

It is also assumed that the input will arrive with some grouping or clustering algorithm applied to each feature, reflected in the label field in the examples above. The obstacle manager maintains a database in the form of a mapping, keyed on the obstacle label, to a list of features. For each obstacle only the N most recent features (points) are held. The obstacle manager may be configured to ignore incoming features beyond a certain range to the robot. This helps ensure bounded memory growth of the application as longer missions unfold.

For each cluster of points, the obstacle manager maintains a single convex polygon representing each cluster. By default a convex hull polygon of each cluster is maintained as shown in Figure 3.

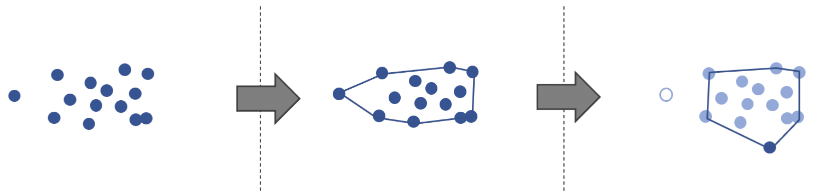

Figure 3: **The Convex Hull Generator**: Each stream of like-labeled points is a cluster from which the generator will maintain a current convex hull. As points age-out, the convex hull may shrink or shift through space. This process naturally accommodates both outlier points, and points associated with a slowly moving obstacle. In the right-hand panel of the figure, the dark blue point shows a newly received point. The white point has aged-out. The lighter blue points are not as old as the new point, but have not yet aged-out.

The stream of points, typically originating from a LIDAR, is presumed to be a constant update stream of points as new information is generated from the LIDAR. The obstacle manager holds enough points in memory to periodically generate a convex hull, but it also guards against unbounded memory by allowing points to drop. A point will be dropped based on one of two criteria. The first is a maximum total number of points held, with oldest points dropped as new points come in, i.e., first-in-first-out. The max amount is applied per-cluster. Currently there is no limit on the number of clusters. This max limit is by default 20 points, but can be set with the `max_pts_per_cluster` parameter as shown below.

```
max_pts_per_cluster = 40   // default is 20
max_age_per_point   = 10   // default is 20 seconds
```

The obstacle manager also will remove points from memory based on the age of the incoming point. By default, the point will be dropped after 20 seconds. The `TRACKED_FEATURE` message does not contain a timestamp. The age of the point is based on the timestamp the obstacle manager applies upon receipt. Dropping points based on age not only serves the purpose of bounding memory growth, it also serves as a kind of outlier rejection. A spurious LIDAR point, that may have come from a wave or some other non-obstacle phenomena, may cause a temporary growth of the convex hull, but it will not last long. Dropping points based on age also allows moving obstacles to have a convex hull that shifts with the motion of the obstacle. Each time the convex hull for an obstacle changes shape or location, an update message is produced by the obstacle manager. If there is an obstacle avoidance behavior currently in existence in the helm, tied to this obstacle, the behavior will be immediately updated, likely resulting in a slight adjustment for the output on that particular behavior.

An Alternative to Convex Hull Clustering

The convex hull is general and preferred when the object is not of a known size or geometry. In certain cases when information about the object size is known a priori, the user may configure the obstacle manager to associate a regular polygon of fixed size. The polygon is centered on the center of mass for all points that have not aged-out, as shown in Figure 4. As before, the polygon will shift with a moving obstacle as new points arrive and older points age-out.

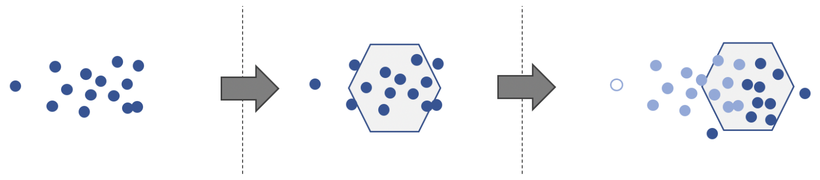

Figure 4: The obstacle manager may also maintain a regular polygon of user configurable radius and number of vertices, centered on the average of points for a given cluster.

To generate obstacle polygons in this manner the `lasso` parameter is used with the following options:

```
lasso = true                // default is false
lasso_points = 6            // default is 6
lasso_radius = 5            // (meters) default is 5
```

The first parameter turns on the lasso option, and the following two set the radius and number of vertices used in the regular polygon.

2.1.2 Configuring Obstacles of Given Location and Size

The obstacle manager may also be configured with obstacles, in polygon form, at given shape and location. This can be done with a configuration parameter in the mission (`.moos` file), or through incoming mail. In either case, the format is the same.

To configure given obstacles through incoming mail, the format is:

```
GIVEN_OBSTACLE = pts={109,-72:113,-76:113,-82:109,-86:103,-86:99},label=23
```

Providing given obstacles through MOOS messages is convenient when using another MOOS app to generate obstacles, perhaps from different sensor input, or perhaps through simulation to stress test the autonomy system. An obstacle is distinct by its label. For any given obstacle, the message may simply arrive once, or it may be steadily updated depending on the source application. Updates are applied immediately and passed on from the obstacle manager with a posting for any consumer of information about this obstacle, e.g., an obstacle avoidance behavior in the helm.

To configure given obstacles in the mission file, the format is:

```
given_obstacle = pts={109,-72:113,-76:113,-82:109,-86:103,-86:99},label=23
```

Providing given obstacles in the mission file may be convenient if there are known obstacles such as buoys, or other items that are consistent with an operation area. Such data can simply be just loaded at mission time.

2.1.3 Obstacle Duration and Memory Management in the Obstacle Manager

The obstacle manager holds information about all known obstacles. A policy for obstacle deletion is needed to guard against unbounded memory growth. In the case of obstacles tied to LIDAR points, there is already a policy for points to age-out after a certain duration. The obstacle manager will remove the obstacle from its list of obstacles when there are no longer any LIDAR points associated with the obstacle. This will occur naturally as the vehicle moves away from an obstacle, beyond sensor range. And if the LIDAR points were a false detection due to a sensor anomaly, or wave, the flow of points related to this false detection will typically soon cease and the obstacle is deleted from the obstacle manager memory.

The case of obstacles arriving from `GIVEN_OBSTACLE` messages is different. The source of these obstacles is not known by the obstacle manager, by design. By default, once the obstacle manager has received information about an obstacle, it is held by the obstacle manager forever. When the obstacle manager is consuming information in this manner, a *duration* is required as part of the mail message. For example:

```
GIVEN_OBSTACLE = pts={109,-72:113,-76:113,-82:109,-86:103,-86:99},label=23,duration=5
```

By default, duration components are mandatory and cannot be any greater than 60 seconds. This amount may be changed with the configuration parameter:

```
given_max_duration=30  // Default is 60 seconds
```

Or it may be disabled completely with:

```
given_max_duration = off
```

This compels the application generating the `GIVEN_OBSTACLE` information to be mindful of obstacle duration, typically updating the information for each obstacle at a rate that is faster than the posted duration. This allows the obstacle manager to delete an obstacle in the *absence* of a recent update.

Currently the only application producing `GIVEN_OBSTACLE` mail messages is the `uFldObstacleSim` application. This method of feeding the obstacle manager may also be used by a future app that has a better method of clustering LIDAR points than the simple convex hull algorithms described earlier that are currently in use by the obstacle manager

2.1.4 Obstacle Manager Actions Upon Deletion of an Obstacle

The intended consumer of obstacle manager output is the helm. When a new obstacle is detected, if it is within the configured range of the vehicle, a new obstacle avoidance behavior is spawned. When the vehicle later goes beyond this range, the avoidance behavior will be deleted by the helm.

Normally, by the time the obstacle manager is about to delete a known obstacle, the helm has probably already deleted the corresponding obstacle avoidance behavior. However, it is possible that the obstacle became known to the obstacle manager due to spurious sensor/LIDAR data. Perhaps this obstacle is very near the vehicle and caused the helm to spawn an obstacle avoidance behavior for this phantom obstacle. In this case the flow of sensor information related to this false obstacle may cease a very short time later, and the obstacle manager will quickly delete the obstacle from its list of known obstacls. Yet the helm may still have the avoidance behavior associated with the false obstacle, and would otherwise continue to have this behavior until the vehicle moves far enough away. This is clearly not desirable since the avoidance behavior for that short-lived false obstacle may constrain the vehicle motion in adverse ways. Instead, we want the vehicle behavior to be removed when the obstacle manager removes the obstacle. For this reason, the obstacle manager will publish the below posting whenever an obstacle is removed from its memory:

```
OBM_RESOLVED = 428
```

The helm obstacle avoidance behavior `BHV_AvoidObstacleV21` or newer, monitors for the above postings. If it notes that the obstacle id for which it is associated has been resolved, the behavior will put in motion the steps to self-delete immediately.

2.2 Configuring the Helm to Handle Obstacle Manager Output

The obstacle manager exists primarily to serve the helm, by posting notifications to the helm that allow the helm to (a) spawn an obstacle avoidance behavior for a new obstacle, and (b) update a previously spawned obstacle avoidance behavior with an updated location or shape of the obstacle.

Configuration on the helm side is straight-forward, requiring only a configuration block for the obstacle avoidance behavior, similar to:

```
//-----------------------------------------------
Behavior=BHV_AvoidObstacleV21
{
  name       = avd_obstacles_
  pwt        = 500
  condition  = DEPLOY = true
  templating = spawn
  updates    = OBSTACLE_ALERT

     allowable_ttc = 5
       buffer_dist = 3
    pwt_outer_dist = 20
    pwt_inner_dist = 10
    completed_dist = 25
}
```

See the documentation for the obstacle avoidance behavior for more details on the above parameters. The important point here is that upon startup of the helm, no obstacle avoidance behavior will be spawned until the helm receives an alert about an obstacle. This alert comes via a posting to the variable `OBSTACLE_ALERT`. Until an alert arrives, this behavior exists in the helm as a *template*, capable of spawning any number of behaviors, one for each obstacle.

When the helm starts, on behalf of the obstacle avoidance behavior template, the helm posts an alert request:

```
OBM_ALERT_REQUEST = alert_range=20, update_var=OBSTACLE_ALERT
```

The alert request uses the value of the updates parameter and the pwt_outer_dist parameter to construct the alert request. The alert_range component of the alert request is automatically set to match the pwt_outer_dist configuration parameter of the behavior. In the behavior, when the obstacle is at, or beyond this range, the priority weight of the behavior becomes zero. This same range value will inform the obstacle manager that (1) until the obstacle is closer than this distance, postings to the OBSTACLE_ALERT should not be made for this obstacle id, and (2) after the obstacle has become farther than this distance, the same said postings should cease. After the vehicle has opened range to the obstacle beyond the completed_dist, the behavior will complete and will be removed from them helm.

Note that if there are say ten instances of this behavior, for ten separate obstacles, they will all receive their updates through the same OBSTACLE_ALERT MOOS variable. Each posting, however, will contain the name of the behavior. For example:

```
OBSTACLE_ALERT = name=avd_obstacles_ob_08#poly=pts={52.2,-32.2:53,-33.02:53,...
OBSTACLE_ALERT = name=avd_obstacles_ob_03#poly=pts={52.82,-115.86:50.64,...
OBSTACLE_ALERT = name=avd_obstacles_ob_02#poly=pts={72.07,-68.93:75.15,...
```

The helm will ensure the updates are only applied to the behavior that matches the name in the update.

3 Implementation of the Obstacle Manager

3.1 Obstacles versus Contacts

The obstacle manager was designed to handle stationary obstacles like buoys or bridge pylons. To handle moving obstacles such as other marine vessels, the IvP Helm uses a similar MOOS application called a contact manager and a collision avoidance behavior based on COLREGS protocol. This is outside the scope of this paper. Our convention is to use the term *obstacle* for objects that are stationary or at best slowly drifting. Obstacles are handled by the obstacle manager and the Avoid Obstacle behavior. We use the term *contact* for moving vessels. Contacts are handled by the contact manager, and the Collision Avoidance behaviors.

3.2 Drifting Obstacles

The obstacle manager is equipped with the ability to handle *drifting* obstacles, e.g., a drifting buoy. In effect, there is not much difference between a truly drifting object and an object with slightly shifting sensor readings. Proper functioning of the obstacle manager depends on proper configuration of the *decay* of sensed points. As the drifting obstacle moves, sensed points at the new location will appear, and sensed points at older locations will become stale and disappear from the obstacle manager memory. By default the number of sensed points is limited in size, per cluster key, to 20 points. This can be changed with the parameter `max_pts_per_cluster`. By default, the points will decay and be removed from memory after 20 seconds. This can be changed wit the parameter `max_age_per_point`.

3.3 Obstacle and Alert Management

The obstacle manager by default will not generate any alerts unless another app has indicated that it would like to receive alerts. This alert request comes in the form of a message:

```
OBM_ALERT_REQUEST = update_var=OBSTACLE_ALERT, alert_range=40, name=avd_obstacle
```

For the current release of the obstacle manager, an alert of a single configuration is supported. Subsequent publications to `OBM_ALERT_REQUEST`, with different values for the alert variable or range, will simply overwrite the previous setting.

3.3.1 Alert Generation and The Alert Range

The obstacle manager will generate alerts about an obstacle to the variable requested in the `OBM_ALERT_REQUEST` message. The first time this alert variable is published, it can be regarded as alert in the sense that the obstacle's existence is new information for whomever is subscribing for these alerts (typically the helm). A couple example postings are shown below.

```
OBSTACLE_ALERT = name=ob_4#poly=pts={48.7,-77.2:52.3,-80.8:52.3,-86:48.7,-89.6:43.5,\
                 -89.6:39.9,-86:39.9,-80.8:43.5,-77.2},label=ob_4

OBSTACLE_ALERT = name=ob_2#poly=pts={62.9,-48.7:67.1,-52.9:67.1,-58.9:62.9,-63.1:56.9,\
                 -63.1:52.7,-58.9:52.7,-52.9:56.9,-48.7},label=ob_2
```

Subsequent alert publications are essentially updates on the obstacle. These subsequent publications are only made if the size or the position of the obstacle changes. For sensed obstacles derived from point data or other dynamic data, alert updates are fairly frequent. For given static obstacles, subsequent alert updates may never happen after the initial alert.
When the robot is beyond the *alert range* to an obstacle, the obstacle manager will no longer generate alerts. Alert publications will resume as soon as the robot returns to within the alert range. For the relatively static given obstacles, the obstacle manager takes care to re-publish an alert for the obstacle when the robot returns within the alert range, even if the size or position of the obstacle has not changed. A vehicle returning within the alert range always needs to be re-alerted as if encountering this obstacle for the very first time.

3.3.2 Resolution of Alerts

The obstacle manager will generate a single alert for each obstacle, thereafter assuming the entity that needed to know about the obstacle has been properly notified. For dynamic obstacles, alerts will continue as the position or shape of the obstacle changes.

The helm may at some point want to delete the, e.g., obstacle avoidance, behavior that registered for the alert, typically when the obstacle has been passed and has reached a range where it is no longer a concern. In the case of the AvoidObstacle behavior, the behavior will be deleted when the obstacle is beyond the completed_dist range, and this range is tied to be equivalent to the requested alert range.

3.3.3 The Ignore Range

For dynamic obstacles, receiving tracked feature point information, normally all such points are received and processed, regardless of range. The ignore_range parameter sets a distance, in meters, beyond which an incoming point will be ignored. This can help reduce management of spurious obstacles at obviously harmless ranges to ownship. By default this value is -1, meaning all points are received and managed. In future releases this range may be automatically tied to the alert range, and ignore regions will also be supported, to reject points on land or outside of the vehicle operation area.

4 Configuration Parameters for pObstacleMgr

The following parameters are defined for pObstacleMgr. A more detailed description is provided in other parts of this section. Parameters having default values are indicated.

Listing 4.1: Configuration Parameters for pObstacleMgr.

alert_range:	The range in meters, between ownship and an obstacle, that an alert will be triggered. Section 3.3.1.
ignore_range:	The range in meters, between an incoming point (tracked feature) and ownship, beyond which the point will be ignored.
lasso:	If true, the polygon associated with each cluster will be firmly set to a circular polygon of a set radius and set number of vertices. The default is false. Section 2.1.1.
lasso_points:	The number of points used in the regular polygon representing the obstacle, if the lasso parameter is set to true. The default is 6, a hexagon. Section 2.1.1.
lasso_radius:	The radius (distance to any vertex) used in the regular polygon representing the obstacle, if the lasso parameter is set to true. The default is 5 meters. Section 2.1.1.
max_age_per_point:	The maximum number of seconds that a point (tracked feature) will be retained in memory. Beyond this number, points will be dropped. The default is 20 seconds. Section 3.2.
max_pts_per_cluster:	The maximum number of points (tracked features) retained in memory per cluster (points having the same label). Beyond this number, oldest points will be dropped. The default is 20 points. Section 3.2.
point_var:	The name of the MOOS variable to looked for tracked features. The default is TRACKED_FEATURE. Section 2.1.1.

obstacles_color:	When the obstacle manager is rendering obstacles (post_view_polys is true), this parameter will set the obstacle color. The default value is blue.
poly_label_thresh:	When the number of obstacle polygons is greater than this parameter, the view polylgons will be published with a label color of invisible, resulting in less work for the pMarineViewer. The default is 25. This is solely to help boost performance of pMarineViewer in extreme uses cases of very large numbers of obstacles and high time warp, and possibly slower machines. It does not affect the function of the obstacle manager in any other way.
poly_shade_thresh:	When the number of obstacle polygons is greater than this parameter, the view polylgons will be published with a shade (fill) color of invisible, resulting in less work for the pMarineViewer. The default is 100. This is solely to help boost performance of pMarineViewer in extreme uses cases of very large numbers of obstacles and high time warp, and possibly slower machines. It does not affect the function of the obstacle manager in any other way.
poly_vertex_thresh:	When the number of obstacle polygons is greater than this parameter, the view polylgons will be published with a vertex size of zero, resulting in less work for the pMarineViewer by only rendering the polygon edges, without the vertices. The default is 150. This is solely to help boost performance of pMarineViewer in extreme uses cases of very large numbers of obstacles and high time warp, and possibly slower machines. It does not affect the function of the obstacle manager in any other way.
post_dist_to_polys:	Legal values are true, false, or close. If true, the distance from ownship to an obstacle is published for each obstacle, to the variable OBM_DIST_TO_OBJ. When set to close, these publications only occur when the obstacle is closer than alert_range. When false, these postings are turned of completely. The default is close.
post_view_polys:	When true, the obstacle polygons are published published by the obstacle manager. Normally this is redundant since the obstacles are also published by the obstacle avoidance behavior and the obstacle simulator. Legal values are true and false. The default is false.

4.1 An Example MOOS Configuration Block

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

```
$ pBasicContactMgr --example or -e
```

This will show the output shown in Listing 2 below.

Listing 4.2: A Simple pObstacleMgr *Example.*

```
=================================================================
pObstacleMgr Example MOOS Configuration
=================================================================

ProcessConfig = pObstacleMgr
{
  AppTick   = 4
  CommsTick = 4

  point_var = TRACKED_FEATURE  // default is TRACKED_FEATURE

  given_obstacle = pts={90.2,-80.4:...:85.4,-80.4},label=ob_23

  post_dist_to_polys = true  // true, false or (close)
  post_view_polys = true     // (true) or false or

  max_pts_per_cluster = 20   // default is 20
  max_age_per_point   = 20   // (secs)  default is 20

  alert_range  = 20          // (meters) default is 20
  ignore_range = -1          // (meters) default is -1, (off)

  lasso = true               // default is false
  lasso_points = 6           // default is 6
  lasso_radius = 5           // (meters) default is 5

  obstacles_color = color    // default is blue

  // To squeeze more viewer effic when large # of obstacles:
  poly_label_thresh = 25     // Set label color=off if amt>25
  poly_shade_thresh = 100    // Set shade color=off if amt>100
  poly_vertex_thresh = 150   // Set vertex size=0 if amt>150
}
```

5 Publications and Subscriptions of pObstacleMgr

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

```
$ pObstacleMgr --interface or -i
```

5.1 Variables Published by pObstacleMgr

The output of pObstacleMgr is:

- 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.
- [ALERT_VAR]: The obstacle manager will publish alerts through a variable specified in an alert configuration, via the incoming OBM_ALERT_REQUEST message. See Section 3.3.1.
- VIEW_POLYGON: The polygon representing the obstacle is posted in this variable. It is re-published when/if the shape or location changes, or upon ownship approaching within range of the obstacle.
- OBM_DIST_TO_OBJ: For each object retained in the object manager's memory, object manager

will continually post the distance from ownship to the obstacle, in meters. This posting can disabled by setting the post_dist_to_polys to false. By default it is enabled.

- OBM_CONNECT: Upon successful launch of the obstacle manager, this variable is posted. It helps coordinate with the obstacle simulator which may be running on the shoreside and may have already published obstacle information. Upon receipt of this posting, the obstacle simulator will refresh the postings.
- OBM_MIN_DIST_EVER: The obstacle manager uses its knowlege of all obstacle locations and ownship location and keeps track of the closest that an obstacle has ever come to ownship. This minimum distance, and the id of the obstacle, are posted to this variable.
- OBM_RESOLVED: When an obstacle is removed from the obstacle manager, a notice is posted containing just the id of the removed obstacle. This may be used by the obstacle avoidance behavior in the helm to let it know that this obstacle no longer exists. Section **??**.

The obstacle manager will also publish to whatever MOOS variables are specified in the obstacle alerts. See Section 3.3.

5.2 Variables Subscribed for by pObstacleMgr

The pObstacleMgr application will subscribe for the following four MOOS variables:

- APPCAST_REQ: A request to generate and post a new apppcast report, with reporting criteria, and expiration.
- GIVEN_OBSTACLE: One of the obstacle manager input options is to receive the obstacle information in the form of a convex polygon with a unique label. See Section 2.1.2.
- NAV_X: Ownship's current position in x coordinates.
- NAV_Y: Ownship's current position in y coordinates.
- OBM_ALERT_REQUEST: A message, typically from the obstacle avoidance behavior of the helm, to configure the criteria and format for posting obstacle manager alerts. See Section 3.3.
- TRACKED_FEATURE: One of the obstacle manager's input options is to received simulated LIDAR points. Each point is received as a message of this variable. See Section 2.1.2.

6 Terminal and AppCast Output

The pObstacleMgr application produces some useful information to the terminal on every iteration of the application. An example is shown in Listing 3 below. This application is also appcast enabled, meaning its reports are published to the MOOSDB and viewable from any uMAC application or pMarineViewer. The counter on the end of line 2 is incremented on each iteration of pObstacleMgr, and serves a bit as a heartbeat indicator. The "0/0" also on line 2 indicates there are no configuration or run warnings detected.

The output in the below example comes from the s1_alpha_obstacles mission.

Listing 6.3: Example terminal or appcast output for pObstacleMgr.

```
1  ===================================================================
2  pObstacleMgr alpha                                          0/0(238)
3  ===================================================================
 4  Configuration (point handling):
 5    point_var:    TRACKED_FEATURE
 6    max_pts_per_cluster: 50
 7    max_age_per_point:   60
 8    ignore_range:        40
 9  Configuration (alerts):
10    alert_var:   OBSTACLE_ALERT
11    alert_name:  avoid_obstacle_
12    alert_range: 19
13  Configuration (lasso option):
14    lasso:        true
15    lasso_points: 8
16    lasso_radius: 6
17  ============================================
18  State:
19    Nav Position:      (61,-129.7)
20    Points Received:   58
21    Points Invalid:    0
22    Points Ignored:    147
23    Polygon obstacles: 4
24    Clusters:          4
25    Clusters released: 0
26
27  ObstacleKey  Points  HullSize  Updates
28  -----------  ------  --------  -------
29  b            14      8         n/a
30  c            16      8         55
31  d            13      8         23
32  e            11      8         46
```

The first group of lines (4-16) show the configuration settings for pObstacleMgr. The status of pObstacleMgr is shown in Lines 18-32.

7 Simple Example Missions

As of Release 19.8, there are two example missions using the obstacle manager.

- `s1_alpha_obstaclemgr`: A single vehicle mission with obstacles generate by a stream of points.
- `m2_berta_obstacles`: A two vehicle mission with obstacles given at fixed locations.

Differences between the obstacle manager and the contact manager:
- obstacles don't have type or group
- Only one global alert range for all obstacles. Set in the config block but may be overridden by a behavior when it registers wit the obstacle manager