The uFldFlagManager Application

Fall 2017

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

1 Overview

The `uFldFlagManager` is a shoreside manager used for marine autonomy competitions where flags are involved. Flags are declared at the outset, each with a position and a unique label. Vehicles have the ability to grab a flag by posting a request. The flag may or not be granted, but if granted, then the grabbing vehicle then owns the flag and it cannot be grabbed by other vehicles.

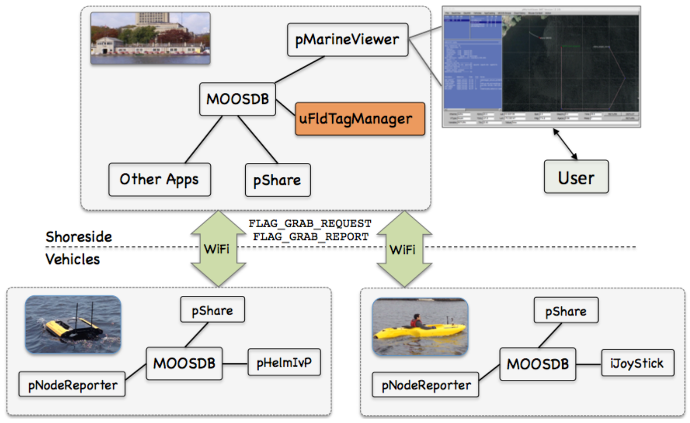

Figure 1: **Typical uFldFlagManager Topology:** The flag manager runs on the shoreside and handles requests from vehicles to grab flags at pre-defined locations provided to the flag manager through configuration upon startup.

2 Configuration Parameters for uFldFlagManager

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

Listing 2.1: Configuration Parameters for `uFldFlagManager`.

`default_flag_range`:	The default range of the flag if no range is specified. Default is 10 meters. Section 4.1.
`near_flag_range_buffer`:	The distance, beyond the flag range, within which a `near_post` event will occur. Default is 2 meters. This compensates for latency between the event and user receipt of the event notice.
`default_flag_type`:	The default type of the rendered marker if no type is speciFied. Default is circle. Section 4.1.
`default_flag_width`:	The default width of the rendered marker if no width is specified. Default is 5 meters. Section 4.1.
`flag`:	A declaration of a flag, with ID and location information. Section 4.1.
`ungrabbed_color`:	The color of the posted marker when not grabbed by any vehicle. Default is `"red"`. Section 4.2.
`grabbed_color`:	The color of the posted marker when grabbed by some vehicle. Default is `"white"`. Section 4.2.
`flag_follows_vehicle`:	If true, when a vehicle has a flag, the flag is rendered to be just behind the vehicle. Default is `"true"`. Section 4.2.
`poly_edge_size`:	Sets the edge width of the polygon rendered around each flag. Default is 1.
`poly_edge_size`:	Sets the edge width of the polygon rendered around each flag. Default is 1.
`poly_vertex_size`:	Sets the vertex size of the polygon rendered around each flag. Default is 1.
`poly_edge_color`:	Sets the edge color of the polygon rendered around each flag. Default is grey50.

poly_vertex_color:	Sets the vertex color of the polygon rendered around each flag. Default is blue.
poly_fill_color:	Sets the fill color of the polygon rendered around each flag. Default is grey90.
grab_post:	A MOOS variable and value posting to be made when a vehicle has successfully grabbed a flag. Section 5.7.
goal_post:	A MOOS variable and value posting to be made when a vehicle has successfully scored, returned home with a flag. Section 5.7.
home_post:	A MOOS variable and value posting to be made when a vehicle has returned home without a flag. Section 5.7.
lose_post:	A MOOS variable and value posting to be made when a flag is reset prior to successfully returning home. Section 5.7.
near_post:	A MOOS variable and value posting to be made when a vehicle comes within grabbing range of an enemy flag. Section 5.7.
away_post:	A MOOS variable and value posting to be made when a vehicle leaves grabbing range of an enemy flag. Section 5.7.
deny_post:	A MOOS variable and value posting to be made when a vehicle is unsuccessful in a flag grab attempt. Section 5.7.

An Example MOOS Configuration Block

An example MOOS configuration block can be obtained by entering the following from the command-line:

```
$ uFldFlagManager --example or -e
```

Listing 2.2: Example configuration of the uFldFlagManager application.

```
===============================================================
uFldFlagManager Example MOOS Configuration
===============================================================

ProcessConfig = uFldFlagManager
{
  AppTick   = 4
  CommsTick = 4

  default_flag_width = 3        // Default (in meters)
  default_flag_type  = circle   // Default is circle
  default_flag_range = 10       // Default (in meters)

  flag = x=60, y=-30,  label=one, range=15
  flag = x=60, y=-170, label=two

  ungrabbed_color = red         // Default is red
  grabbed_color   = white       // Default is white

  grab_post = var=SAY_MOOS, sval={say={$VNAME has $FLAG flag}}
```

```
   lose_post = var=SAY_MOOS, sval={say={$FLAG flag is reset}}
   near_post = var=SAY_MOOS, sval={file=sounds/shipbell.wav}
   away_post = var=SAY_MOOS, sval={file=sounds/buzzer.wav}
   deny_post = var=SAY_MOOS, sval={file=sounds/sf-no-soup.wav}
 }
```

3 Publications and Subscriptions of uFldFlagManager

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

```
$ uFldFlagManager --interface or -i
```

3.1 Variables Published by uFldFlagManager

The only output of uFldFlagManager 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. Section 6.
- FLAG_GRAB_REPORT: Shows the result of an incoming flag grab request. Section 5.4.
- FLAG_SUMMARY: A summary of all known flags, positions and whether the flag has been grabbed and by whom. Updated any time a single flag has had its state altered. Section 5.5.
- VIEW_MARKER: A geometric object for rendering a flag at its configured location and current grab state. Section 5.6.

3.2 Variables Subscribed for by uFldFlagManager

The uFldFlagManager 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. Section 6.
- FLAG_GRAB_REQUEST: A request from a vehicle to make a grab for any and all flags within grab range. Section 5.2.
- FLAG_RESET: A request to reset some or all flags back to the ungrabbed state. Section 5.3.
- NODE_REPORT: A report, usually in a steady stream, from a vehicle indicating its current position and other vehicle states. Section 5.1.

4 Configuration of uFldFlagManager

4.1 Basic Flag Configuration

Flags are configured minimally with a location, and a unique label. For example:

```
  flag = x=60, y=-30,  label=one
  flag = x=60, y=-170, label=two
```

If a flag configuration is provided with a non-unique label, the flag will not be loaded, and a configuration warning will be posted in the terminal or appcasting output:

```
Flag with duplicate label: one
```

If a flag configuration is provided with no label, the flag will not be loaded, and a configuration warning will be posted in the terminal or appcasting output:

```
Flag with missing label: x=1, y=2
```

4.2 Configuring the Grabbed and UnGrabbed Colors

Each flag is rendered by a marker configured with the `flag` parameter. For example:

```
flag = x=60, y=-30,  label=one, color=blue
flag = x=60, y=-170, label=two
```

If the color is not specified, it will default to the value given by the parametr `ungrabbed_color`.

When the flag has been grabbed, one of two things will happen. By default, the flag will follow the vehicle, keeping its original color, unless the parameter `flag_follows_vehicle` is set to false. The default is true. If the flag does not follow the vehicle, a grabbed flag will change color to the value specified in the `grabbed_color` parameter. This value by default is white.

4.3 Optional Flag Configuration Parameters

Flags may be additionally configured with the following optional parameters:

- `range`: The `range` parameter indicates the maximum distance between the flag and a vehicle beyond which a flag grab request will be rejected. The default is 10 meters.
- `type`: The `type` parameter refers to the shape of the rendered flag. Legal values are *circle, triangle, square, efield, gateway, diamond*. See Figure 2. The default is "circle".
- `width`: The `width` parameter determines the width of the *rendered* flag, in meters. It has no bearing on flag capturing otherwise. In terms of the range between a vehicle and flag the flag is treated as point-object. The default is 5 meters.
- `color`: The `color` parameter affects the color of the rendered flag. When a flag type has two colors (marker types *efield* and *gateway*, it only affects one of the colors. The default is red. This refers to the "ungrabbed" color. The "grabbed" color cannot be modified. See Section 5.2 for more on grabbing and colors.

Here are some examples:

```
flag = x=60, y=-30,  label=one, range=20, type=triangle, width=11
flag = x=60, y=-170, label=two, color=gree, width=11
```

Here are the supported marker shapes:

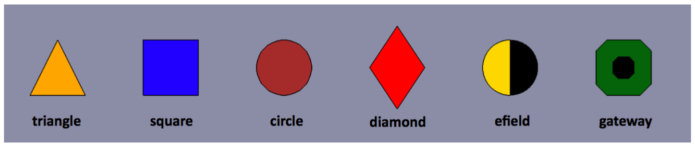

Figure 2: **Supported Marker Types:** Any of these types may be used to render the flags in the flag manager.

4.4 Changing the Default Values of Optional Parameters

It may be convenient, in cases of many flags to be configured, to simply change the default values globally for the optional parameters. The following `uFldFlagManager` parameters allow this:

```
default_flag_width = 4          // Default is 5
default_flag_range = 12         // Default is 10
default_flag_color = beige      // Default is white
default_flag_type  = diamond    // Default is circle
```

The flag color refers to the "ungrabbed" color of the flag. The "grabbed" cannot be overridden and is determined by the `grabbed_color` parameter.

5 Operation of uFldFlagManager

5.1 Handling Node Reports

The flag manager needs to know both the flag positions and the vehicle positions. The former is provided via the configuration file, the latter is derived from node reports, in the `NODE_REPORT` MOOS variable. An example node report:

```
NODE_REPORT = NAME=alpha,X=16.79,Y=-16.55,SPD=1.98,HDG=120.75,DEP=0,LAT=43.82515342,
              LON=-70.33018799,TYPE=kayak,MODE=MODE@ACTIVE:SURVEYING,ALLSTOP=clear,
              INDEX=102,YAW=-0.5367054,TIME=7250613964.12,LENGTH=3
```

Typically this variable is generated locally on the vehicle as `NODE_REPORT_LOCAL` by the `pNodeReporter` application and shared to the shoreside via the `pShare` application. In the shoreside community, the variable is renamed to `NODE_REPORT`. In some simple example missions, having only a single MOOS community serving both the vehicle and the viewer, you may only see `NODE_REPORT_LOCAL`.

5.2 Handling Flag Grab Requests

A flag grab request is handled through the receipt of the MOOS variable `FLAG_GRAB_REQUEST`. This variable is typically generated by one of the field nodes, e.g., robots, and sent by `pShare` to the shoreside MOOS community in which `uFldFlagManager` is running. The format of this variable simply contains the name of the grabbing vehicle:

```
FLAG_GRAB_REQUEST = vname=henry
```

Upon this request, a couple checks are performed. First, the vehicle name is checked against the *community* associated with the incoming message. This is to ensure that one vehicle cannot make a flag grab request on behalf of another vehicle. Typically the vehicle (MOOS) community and the vehicle name are configured to be the same.

The second check performed is the range (linear distance) between the requesting vehicle and the flag location. Each flag has a range associated with it, provided in the configuration block (Section 4.3). If the vehicle is not within this range, the flag grab is rejected. If *multiple* flags are within range of the vehicle when the request is made, all flags are considered to be grabbed by the vehicle. Note: this may change, perhaps with a configuration option to only grab the closest.

When a flag is grabbed, the rendered color will change to the color specified in the `grabbed_color` parameter, which has the default value of white.

5.3 Handling Flag Resets

Once a flag has been grabbed by a vehicle, it is possible to *reset* the flag, i.e., return it to the state where is is not associated/grabbed by any vehicle. This is done through the `FLAG_RESET` variable. When the flag manager receives this variable, flags may be set in one of three ways. First, a flag may be reset by naming a particular flag label, resulting in only that flag being reset. For example:

```
FLAG_RESET = label=one
```

Second, flag(s) may be reset by naming a particular vehicle and releasing all flags held by that vehicle:

```
FLAG_RESET = vname=henry
```

Lastly, the complete list of flags can be reset by posting `FLAG_RESET=all`.

5.4 Posting Flag Grab Reports

Each time a flag grab request has been received (`FLAG_GRAB_REQUEST`), a report is compiled and posted to the MOOS variable `FLAG_GRAB_REPORT`. Typically this variable is shared back out to at least the requesting vehicle, to inform the vehicle of the result of its request. The variable has the form:

```
 FLAG_GRAB_REPORT = grabbed=one,grabbed=seven
 FLAG_GRAB_REPORT = nothing_grabbed
```

5.5 Posting Flag Grab Summary Reports

A flag summary is posted upon application start-up and each time the status of a flag is changed. The summary is posted to the `FLAG_SUMMARY` variable. For example:

```
FLAG_SUMMARY = x=2,y=-4,width=5,range=10,type=circle,label=three #
               x=4,y=27,width=5,range=10,type=circle,owner=alpha,label=two #
               x=7,y=23,width=8,range=10,type=square,owner=alpha,label=one
```

If a flag does not indicate an owner, then the flag is currently ”ungrabbed”.

5.6 Posting Flag Markers

The flag manager publishes to the MOOS variable VIEW_MARKER to indicate the location and status of flags it is managing. Markers are object types known to the pMarineViewer app and their rendering can be turned on and off and resized within the viewer. Each marker is published once upon startup, and re-published whenever it changes state between *ungrabbed* and *grabbed*. An example posting will look something like:

```
  VIEW_MARKER = x=147,y=-43,width=5,range=10.00,primary_color=red,
                secondary_color=black,type=circle,label=five"
```

5.7 User-Configurable Event Postings

The flag manager may be configured to make one or more MOOS posting tied to one of several events:

- When a flag has been grabbed (with the grab_post parameter)
- When a flag has been reset, i.e., a vehicle loses control of the enemy flag, (with the lose_post parameter)
- When a vehicle comes comes sufficiently *near*, i.e., within grabbing range of, an enemy flag to enable grabbing (with the near_post parameter)
- When a vehicle leaves, i.e., *goes away* from, the grabbing range of an enemy flag (with the away_post parameter)
- When a flag grab request has been denied (with the deny_post parameter)
- When a goal has been scored, i.e., a flag has successfully be returned to home base (with the goal_post parameter)

Each of these postings has a set of macros available in either the MOOS variable or the value being posted to the MOOS variable.

5.7.1 Macros Available in MOOS Variable Names

The following macros are available, as part of the MOOS variable name used in an event posting:

- $VNAME: the name of the vehicle involved in the event.
- $UP_VNAME: the upper case name of the vehicle involved in the event.
- $FLAG: the name of the flag involved in the event.
- $VTEAM: the name of the team involved in the event.
- $UP_VTEAM: the upper case name of the team involved in the event.

The following are valid examples:

```
  grab_post = var=FLAG_GRAB_STATE_$UP_VNAME, sval={grabbed}
  deny_post = var=FLAG_DENY_INDEX_$UP_VNAME, dval=22
  away_post = var=LEAVING_$FLAG, sval={true}
```

5.7.2 Macros Available in MOOS Variable Values

The following additional macros are available, as part of the MOOS variable message in an event posting:

- `$REASON`: the reason, if there exists one, involved in the event.
- `$TIME`: the current time (as string value) when the event occurred.

The following are valid examples:

```
grab_post = var=SAY_MOOS, sval={say={$VNAME has $FLAG flag}}
lose_post = var=SAY_MOOS, sval={say={$FLAG flag is reset}}
near_post = var=SAY_MOOS, sval={file=sounds/shipbell.wav}
away_post = var=SAY_MOOS, sval={file=sounds/buzzer.wav}
deny_post = var=SAY_MOOS, sval={file=sounds/sf-no-soup.wav}
```

6 Terminal and AppCast Output

The uFldFlagManager 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 uFldFlagManager, 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 example described in Section 7.

Listing 6.3: Example terminal or appcast output for uFldFlagManager.

```
 1  ===================================================================
 2  uFldFlagManager alpha                                      0/0(447)
 3  ===================================================================
 4  Configuration Summary:
 5  ======================================
 6    default_flag_range: 10
 7    default_flag_width: 5
 8    default_flag_type:  circle
 9
10  Node Report Summary
11  ======================================
12          Total Received: 451
13                   ALPHA: 451       (0.0)
14
15  Vehicle Summary
16  ======================================
17  VName  Grabs  Flags  InFlagZone
18  -----  -----  -----  ----------
19  ALPHA  2      2      false
20
21  Flag Summary
22  ======================================
23  Flag   Range  Owner  Spec
```

```
24  -----  -----  -----  ------------------------------------------------------------
25  five   10            x=147,y=-43,width=5,range=10,type=circle,label=five
26  four   40            x=183,y=-93,width=5,range=40,type=circle,label=four
27  three  10            x=152,y=-164,width=5,range=10,type=circle,label=three
28  two    10     alpha  x=64,y=-157,width=5,range=10,type=circle,owner=alpha,label=two
29  one    10     alpha  x=57,y=-43,width=10,range=10,type=square,owner=alpha,label=one
```

The first few lines (6-8) show the configuration settings for `uFldFlagManager`. The status of `uFldFlagManager` is shown in Lines 10-29. The first status block, lines 10-13, simply confirm the number of node reports received in total and from each vehicle. The second status block, lines 15-19, indicate the number of flag grab requests received from each vehicle, and the total number flags successfully granted to to each vehicle. The latter number represents the total number of granted flags *ever*. So if flags are reset, it's possible that this number could be greater than the total number of flags in the flag manager. The last column in this block indicates whether the vehicle presently is within striking distance of an enemy flag. The last status block, lines 21-29, summarizes both the flag configuration and the status of each flag, under the *owner* column.

7 A Simple Example

The `s1_alpha_ufmgr` example mission distributed with moos-ivp-pavlab provides a simple working example. More explanation to come...