The gen_swimmers Utility for Generating Swim Files for the Rescue Lab

May 2024

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

1 Overview

The gen_swimmmers utility is a command-line tool for generating random swimmer locations for the MIT 2.680 Autonomous Rescue Lab missions. A swimmer location consists of a point in local X-Y coordinates, and further information about the swimmer such as a name and type. Points are generated from one or more provided polygons, and a minimal separation distance can be specified.

1.1 Example Usage

Here is an example usage, creating eight random swimmer locations in a polygon region used by the Autonomous Rescue Lab at the MIT Sailing Pavilion:

```
$ gen_swimmers --poly=-31,-37:-7,-92:95,-46:72,10 --swimmers=8
// gen_swimmers --poly=-31,-37:-7,-92:95,-46:72,10 --swimmers=8 --sep=20
// Lowest dist between swimmers: 20.40
poly = pts={-31,-37:-7,-92:95,-46:72,10}
swimmer = name=p01, x=21, y=-36
swimmer = name=p02, x=50, y=-38
swimmer = name=p03, x=86, y=-46
swimmer = name=p04, x=-7, y=-74
swimmer = name=p05, x=59, y=-11
swimmer = name=p06, x=25, y=-15
swimmer = name=p07, x=67, y=-54
swimmer = name=p08, x=-11, y=-54
```

Normally swim files are generated by re-directing the output from the command line to a file. Using the example above:

```
$ gen_swimmers --poly=-31,-37:-7,-92:95,-46:72,10 --swimmers=8 > swim_file.txt
```

Then the swim file can be specified when launching a mission in the Autonomous Rescue lab with:

```
$ ./launch.sh --swim_file=swim_file.txt 10  (time warp 10)
```

The eight generated swimmers from this example would look something like Figure 1 below.

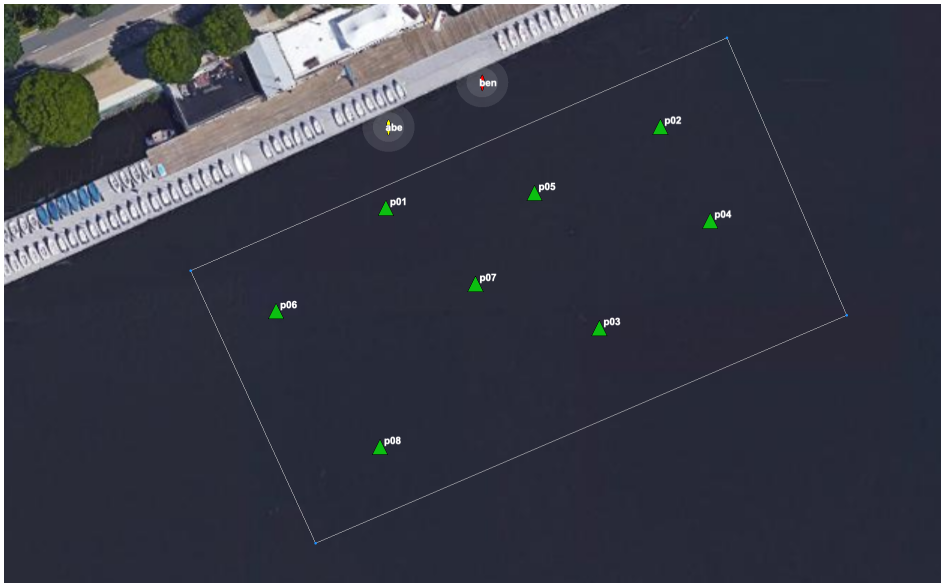

Figure 1: **An Example Set of Swimmers**: The eight triangles, labeled `p01` through `p08`, represent the eight swimmers generated by the example call to `gen_swimmers`.

This `gen_swimmers` web page can be opened on most systems from the command line with `gen_swimmers --web`. Running `gen_swimmers --help` will also show the supported command line options.

1.2 Motivation and Intended Usage

The motivation for the `gen_swimmers` utility is to support the MIT 2.680 Rescue Mission Lab, by creating random problem instances. This utility is used typically in one of two manners: (a) to create and store swim files files for an established set of problem instances prior to a competition, virtual or in-water, or (b) to create a problem instance on the fly during a simulation launch.

By way of example, the following two are equivalent. The first example generates a swim file, then launches with the newly created file.

```
$ cd missions/rescue_baseline
$ gen_swimmers --swimmers=12 --sep=20 --pav90 > new_swimfile.txt
$ ./launch.sh --swim_file=new_swimfile.txt 10
```

In the second case, the randomly generated file is created on the fly as part of the launch process.

```
$ cd missions/rescue_baseline
$ ./launch.sh --swimmers=12 --sep=20 --pav90
```

Both will result in a simulation using a swim file with 12 swimmers in the `pav90` oparea, with guaranteed separation of 20 meters between swimmers. In the second case, the launch script internally runs `gen_swimmers`, creating the `mit_rand.txt` swim file. This file will be overwritten on subsequent similar launches and is not under version control.

1.3 Picking Swimmer Locations with Separation

In the example in Section 1.1, `gen_swimmers` generated four points within the specified polygon. The eight chosen points are also at least 10 meters apart. The 10 meter buffer range is the default, and can be overridden with the `--sep=N` argument. `gen_swimmers` will successively choose a random point in the polygon, each time checking to see if it satisfies the buffer distance. If not, a new point will be chosen. For each point, `gen_swimmers` will retry to satisfy the buffer criteria up to 1000 times.

At some point, based on the polygon size and number of requested swimmers and buffer distance, it will become impossible to squeeze in any more swimmers while satisfying the buffer criteria. In this case `gen_swimmers` will automatically scale back the buffer distance each time the maxtries (default is 1000) has failed to produce a point. On each successive round, the buffer distance will be scaled back from the starting buffer distance by 1 per cent.

1.4 Registered vs Un-registerd Swimmers

The Autonomous Rescue missions can be started with all swimmer locations known at the outset. These are also known as *registered* swimmers. A set of *un-registered* swimmers can also be generated along with the registered swimmers. They are generated as one group to ensure the minimal separation requested. When un-registered swimmers are requested, the *type* of each swimmers is included on each line. For example:

```
$ gen_swimmers --poly=-31,-37:-7,-92:95,-46:72,10 --swimmers=4 --unreg=4
// gen_swimmers --poly=-31,-37:-7,-92:95,-46:72,10 --swimmers=4 --unreg=4
// Lowest dist between swimmers: 16.49
poly = pts={-31,-37:-7,-92:95,-46:72,10}
swimmer = type=reg, name=p01, x=52, y=-15
swimmer = type=reg, name=p02, x=8, y=-58
swimmer = type=reg, name=p03, x=5, y=-21
swimmer = type=reg, name=p04, x=79, y=-16
swimmer = type=unreg, name=x01, x=21, y=-17
swimmer = type=unreg, name=x02, x=45, y=-56
swimmer = type=unreg, name=x03, x=10, y=-40
swimmer = type=unreg, name=x04, x=92, y=-42
```

This swim file will look similar to Figure 1, with four registered swimmers and four un-registered swimmers.

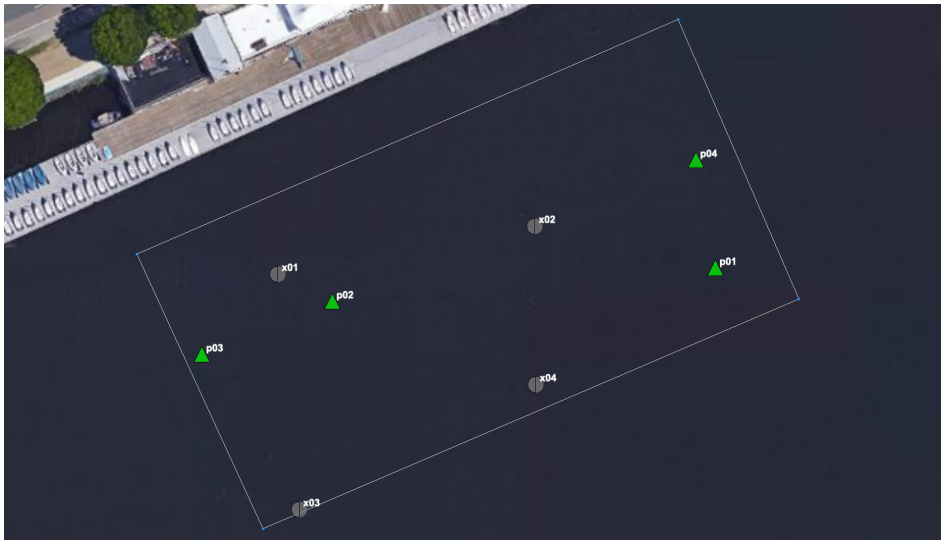

Figure 2: **Registered and Un-registered Swimmers**: The four triangles represent registered swimmers and the four circular markers represent unregistered swimmers.

Note that the swimmer names give away whether the swimmer is registered or un-registered. The name of registered swimmers begins with `'p'`, and the name of un-registered swimmers begins with `'x'`. In the Autonomous Rescue lab, the swimmer alerts use an ID "alias" that makes it impossible to discern the swimmer type from the alias.

1.5 Generating Swimmer Locations from Multiple Polygons

If multiple polygons are provided on the command-line, `gen_swimmers` will generate random points across polygons. For example:

```
$ gen_swimmers --swimmer=23 --poly="60,-40:60,-160:150,-160" \
                  --poly="260,-40:260,-160:350,-160" \
```

It does not matter if the given polygons overlap. It *does* matter if the polygon is non-convex. Only individual convex polygons may be given. However, clearly a non-convex region can be represented by several convex polygons.

Note: The `gen_swimmers` algorithm, when choosing a random point using multiple polygons, first chooses a random polygon. The weight given to each polygon is the same. So if you have one polygon that is 100x bigger than the second one, the distribution of points between the two will be roughly the same, not 100 to 1.

1.6 Pavilion Coordinates

There are two supported polygon region short-cuts supported `gen_swimmers`, representing a small and large operation area near the MIT Sailing Pavilion, shown in Figure 3. These two regions are referred to as `pav60` and `pav90` respectively, due to their length on the shorter dimension. For the

autonomous rescue labs, the smaller region is used for testing one vehicle or two vehicle missions. The larger region is for two-on-two competitions.

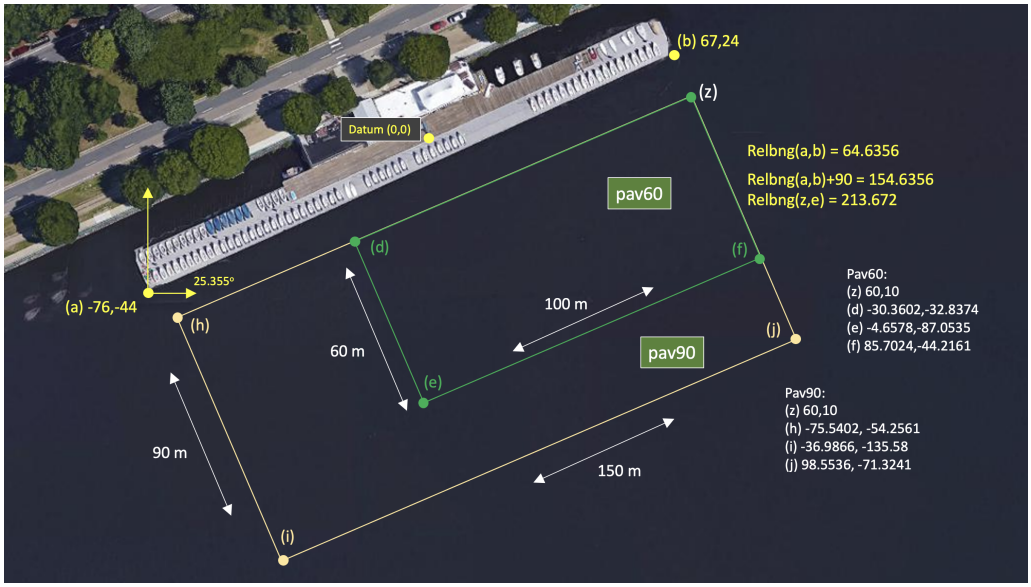

Figure 3: Useful coordinates related to this lab. Note there are two playing field rectangles, a large one (`"pav90"`) roughly the extent of the docks, and a smaller region (`"pav60"`) closer to the Pavlab doors on the East end of the dock.

For the small region:

```
$ gen_swimmers = --poly=60,10:-30,-32:-5,-87:85,-44 --swimmers=15
$ gen_swimmers = --pav60 --swimmers=15                          (same result)
```

For the big region:

```
$ gen_swimmers = --poly=60,10:-76,-54:-37,-136:99,-71 --swimmers=15
$ gen_swimmers = --pav90 --swimmers=15                          (same result)
```