Problem Set 4 - Motion Planning

Please complete the following methods in `rrt_planning.py`:

- `rrt_planning(max_iterations, goal_sample)` : implement the uni-directional RRT algorithm with goal bias.
- `bidirectional_rrt_planning(max_iterations)` : implement the bi-directional RRT algorithm.
- `smooth_path(path, attempts)` : implement a "random short-cut" smoother.

You will be uploading your completed version of `rrt_planning.py` to Gradescope.

The file `examples.py` has four set ups: two with a polygonal robot with (x, y, θ) degrees of freedom, and two with a 3-link revolute planar manipulator. Run this file with `DISPLAY = True` to test and visualize your code. Note that `robot_problem` and `robot_arm_problem` are the same as the graded test cases.

The file `robot.py` has the robot definitions. Note that these are mainly forward kinematics to create an instance of the robot at a specified configuration. They also define a distance function appropriate for the configurations of the robot. Note that the test cases define a `ConfigurationSpace` that defines the ranges of legal values of the joints and which supports (a) testing for valid configurations, (b) sampling random configurations and (c) interpolating between configurations.

Note that robot configurations are simply tuples of joint values. Paths are lists of configurations.

The file `geometry.py` defines the basic geometric classes, `Point`, `Line`, `Polygon`, `Object` and, in particular, implement collision checking.

The only methods that you need to use to implement rrt and birrt are:

- `ConfigurationSpace.sample()` samples a Cspace configuration uniformly at random.
- `RRT.nearest(sample)` find the `TreeNode` in the `rrt` that is closest to sample. Note the components on `TreeNode` are "value" (a configuration), "parent" (another node) and "children" (a list of nodes).
- `Problem.safe_path(conf1, conf2)` interpolates the path between the confs and returns the subset of the path that is collision free.
- `RRT.add_configuration(node, conf)` adds the conf as a child of the node.

To implement smoothing, you will need `Problem.test_path(conf1, conf2)` which returns a boolean indicating whether a path interpolating between the two confs is collision-free.

To run the Gradescope tests, run `test_pset_4.py` in a docker bash container. For example, if you are on Mac, run

```
docker/docker_run_bash_mac.sh drake-20181030 pset_4/
python test_pset_4.py results.json
```

Back to Pset 4 Main (../pset_4.html)

Web design with Bootstrap | Contact: manipulation-tas [AT] mit [dot] edu