Homework #3
due Wednesday 1/24/24 11:59pm

This week we explore the Probabilistic Roadmap planner. We start with the slightly simpler case of a point moving in physical 2D, then advance to the mattress and multi-link robot problems, requiring a C-space planner. We use simple geometric shapes to keep the code reasonably fast.

Indeed this certainly involves coding. We are providing significant portions for the first and second problem, then ask you to extrapolate in the third problem. So please take a look at the demo code, collected in `hw3democode.zip`. Review especially places mark with "FIXME" that need editing. We are also providing supporting code (`astar.py` and `vandercorput.py`). To accelerate the geometry checks, we use the python `shapely` package. If necessary, please install with

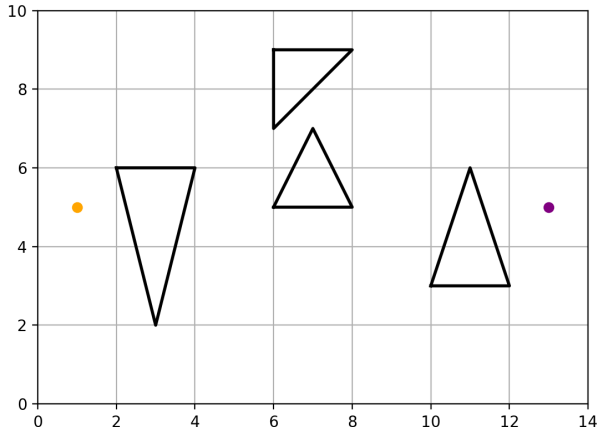
```
pip3 install shapely
```

You are certainly free to write code however you like, but hopefully these will help.

For your submission, please answer any relevant questions, include a screenshot for each part, and append the critical code pieces, especially pieces that you added/changed relative to the demo code.

Problem 1 (2D Point Planner) - 27 points:

Consider the 2D space, with the dimensions/locations as indicated in the figure.



The objective is to create a PRM planner allowing a point to traverse from start (orange) location (1,5) to goal (purple) location (13,5). For variables, we'll use N as the number of nodes we want in our graph and K as the number of nearest neighbors (some of whom should become connected).

Please create the planner using N random samples with a uniform distribution across the 2D space, trying to connect nodes to up to K nearest neighbors, and finally post-processing to remove unnecessary steps. The demo code create a `Node` class providing the geometry checks

- `node.inFreespace(self)` to test whether a node is free (not inside any triangle),
- `node.connectsTo(self, other)` to check whether the straight-line connection between a node and another is collision-free (does not intersect any triangle).

Using these, we ask that you add/edit the functions (marked with "FIXME")

(i) to determine the distance between two nodes,

(ii) to create a list of N valid nodes, uniformly sampled in the space,

(iii) and finally, to post-process the path, removing nodes from the path if they can be skipped (the previous/subsequent nodes can connect).

With working code, please answer the following questions:

(a) For $K = 5$, how many nodes N do you need to find a path consistently (approximately at least 4 out of 5 times)?

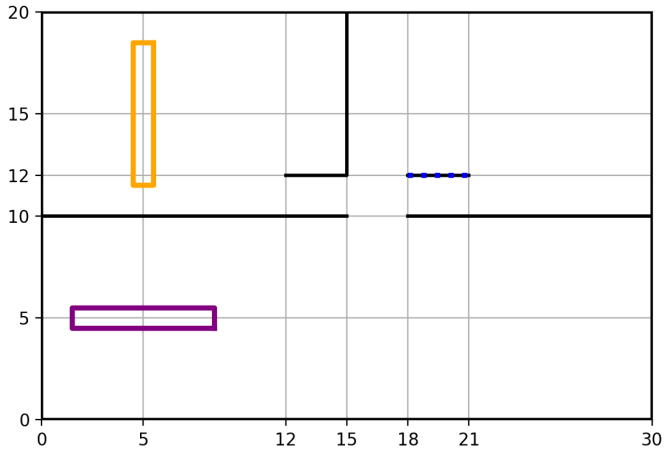
(b) If you increase to $K = 10$, how many nodes N do you need to consistently find a path? Which is better?

(c) What values of K and N will consistently find a "good" path through the narrow space (passing (7.25, 7.25))?

You are welcome to try other sampling strategies, which may reduce the number of samples needed, but we'll definitely do that in Problem 2. Besides the answers, please submit the relevant code and an example single screen shot showing the full graph and final path for each part.

Problem 2 (3D Mattress-Movers Planner) - 32 points:

While we're still in a 2D world (making graphing a little easier), let's consider the 3D problem of moving a long, thin box. I.e. the mattress moving problem with the mattress flipped on it's side. Consider the following space.



The start pose (orange) is centered at (5,15) aligned with the y axis, the goal pose (purple) is centered at (5,5) aligned with the x axis. Assume the mattress is 7ft long and 1ft thick. For parts (a) and (b), please *leave out* the small wall (18,12)-(21,12). This bonus wall makes the problem harder (the path is more constrained) and will appear in part (c).

Also note, unlike other angles, the orientation of the mattress repeats every 180°. That is, we do not distinguish between the head or foot end of the mattress. You will notice that the code defines the `distance()` accordingly and wraps the angle to ±90°.

We are going to investigate the sampling strategy. So, working from the demo code, please add/edit the functions

(i) to create nodes sampling the x, y, θ space (appropriate to the cases below)

(ii) and again, to post-process the path, removing unnecessary nodes in the path.

Please answer the following questions. Do not go crazy trying to find exact values for K or N - within a factor of 2 or 3 is fine. We care about the order of magnitude and trends between cases.

(a) For a random sampling (uniform distribution) of the state space, *approximately* what values of K and N consistently (4 out of 5 times) find a path?

(b) For a random sampling focused on the edges of objects in C-space, *approximately* what values of K and N again consistently provide solutions.

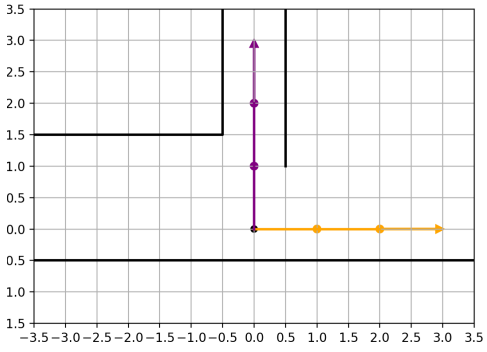
(c) If you repeat the non-uniform sampling approach from (b) but add the small wall (18,12)-(21,12), how much do you need to increase K, N to keep consistently creating solutions.

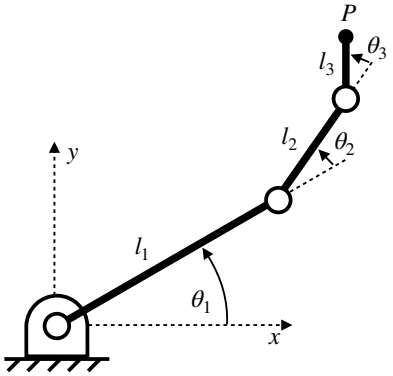
(d) Finally, you may have noticed that the demo code changed the connection strategy to connect each node to K neighbors rather than the standard approach of only attempting K connections. This makes the graph more effective (for our situation). If you return to case (b) (removing the small wall and using a edge-adjacent sampling) but also update the code to use the standard connection approach, how much do you need to increase K, N to get solutions.

As before, besides the answers, please submit the relevant code and an example single screen shot showing final (post-processed) path for each case.

Problem 3 (3DOF Robot Planner) - 37 points:

And finally, to flex our planning muscles, consider a planar 3R robot.





The start pose (orange) is at joints values (0°,0°,0°) pointing to the right. The goal pose (purple) is pointing up with joint values (90°, 0°, 0°). The objects are as marked (and given in the code). Clearly the arm is going to have to fold up before it can reach upward.

Having seen the PRM in the previous problems, we are asking you to program the majority of this problem. We have provided the graphics and overall architecture, while the to-be-completed spots are again marked with "FIXME". These include

(i) most of the Node class, including the initialization, utility and PRM support functions,
(ii) the main PRM functions to sample/connect the nodes and post-process the path.

Several "hints"/suggestions:

- Uniform sampling should be sufficient here.
- Continue to use the alternate connection strategy used in the last problem (connecting to K neighbors, tested in order of distance, not just testing K neighbors).
- Testing whether a particular state/node is in collision with the walls is easy. But checking whether nodes connect, i.e. whether the area swept by a movement overlaps any walls, is harder. So I would generally test whether the links remain a certain Cartesian distance from the walls. And then test intermediate nodes (states) at a corresponding joint step size.
- The Shapely library provides geometric support functions. The
  - robot links are encoded as a Shapely `LineString` object, being a connected sequence of line segments and stored in `node.links`,
  - walls are encoded in a Shapely `MultiLineString` object, being multiple `LineString`s, providing connected and disconnected line segments and stored in `walls`.

  The library then provides the functions
  - `walls.disjoint(node.link)` returning `True` or `False` whether the walls and robot links are not overlapping.
  - `walls.distance(node.link)` returning the minimum distance between any portion of the links and walls.

How far from the walls do you want to remain? Keep in mind the passage is 1m wide and the links are 1m long. And with which joint step size do you test intermediate nodes?

We will proceed in 2 steps/cases:

(a) First, consider the joints ranges limited to $\pm 180°$ on all joints. Do not allow wrapping. So a joint at $+170°$ has to rotate back $-340°$ to get to $-170°$. Wires are unwrapped. The distance between any two angles is thus always just the difference $|\theta_A - \theta_B|$ and the coordinates are just the joint angles.

(b) Then consider unlimited joints that can move past $\pm 180°$, not worried about wires wrapping. Thus an angle of $-170°$ should be treated the same as $190°$ and the above movement would take just $+20°$. The algorithm does not need to sample the nodes any differently. But `intermediate()`, `distance()`, and `coordinates()` functions will need modification to reflect the shorter/"better" option for the above movement and thus allow the wrapping behavior.

Does the planner find a solution that wraps up the arm, placing the tip correctly but ending with a joint at $\pm 360°$?

Please submit the step size/distance/N/K parameters you used, a screenshot and the way points the planner reported in both cases, as well as the code you created.

Problem 4 (Time Spent) - 4 points:

Approximately how much time did you spend on this homework? Any particular bottlenecks?

And, if you are handing in late, would you prefer to use penalty-free late hours (default) or take a 10 points/day penalty (we are happy to change later).