CS 6301.008.18S Lecture—March 6, 2017

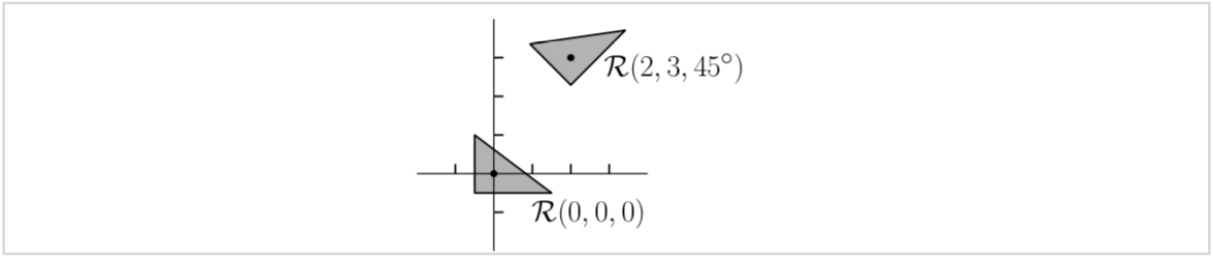
Main topics are #robot_motion_planning, #roadmaps, and #Minkowski_sums.

Prelude

- Project proposals are due. Please submit them to eLearning ASAP if you have not already done so.
- I will post Homework 3 before the end of the day. It is due in two and a half weeks, Thursday March 22nd. It's shorter than the others, because you have projects now.

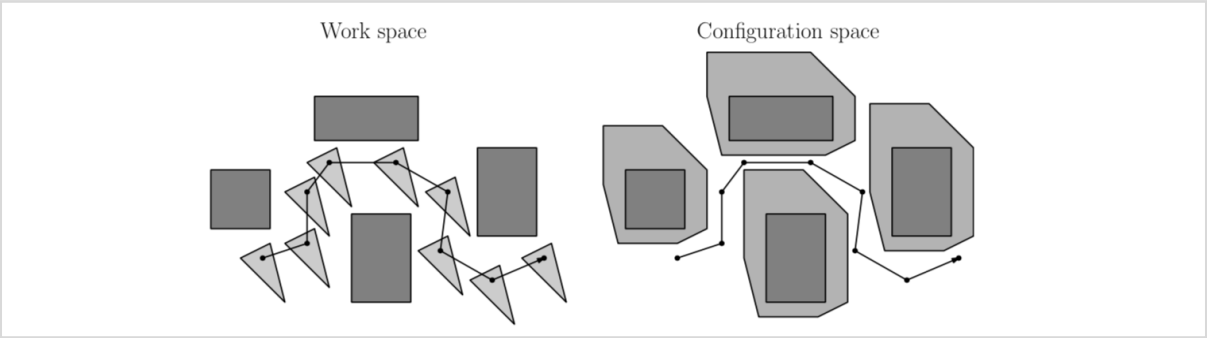
Robot Motion Planning

- Today, we're going to discuss the robot motion planning problem. Given a polygonal robot in the plane and a set of obstacles, how do we get the robot from point s to point t without colliding with the obstacles?
- We'll start with some definitions so it's clear what we're talking about later in the lecture.
- The *work space* is the environment the robot moves around in. Like the factory floor.
- For our purposes today, a robot is fully defined by two things.
  - The *configuration* is a finite sequence of values describing the position and possibly orientation of a robot.
  - The robot's geometric description. Basically, what shape is the robot?
- For example, maybe the robot is a triangle that we can translate and rotate in the plane.
  - Its configuration could be the (x, y)-coordinates of some reference point on the triangle and an angle theta describing its orientation.
  - The geometric description would be its shape at some canonical position.
  - Together we learn R(x, y, theta), the exact *placement* of the robot given configuration (x, y, theta).



- More complex situations require more complex configurations. If we want to work with an articulated arms, we need to describe how far out each joint lies in the geometric description. The configuration could be the location in space of its base and some extra information on the direction and orientation of its end manipulator.
- Today, we focus on computing compute collision free paths for translating robots when there are obstacles.

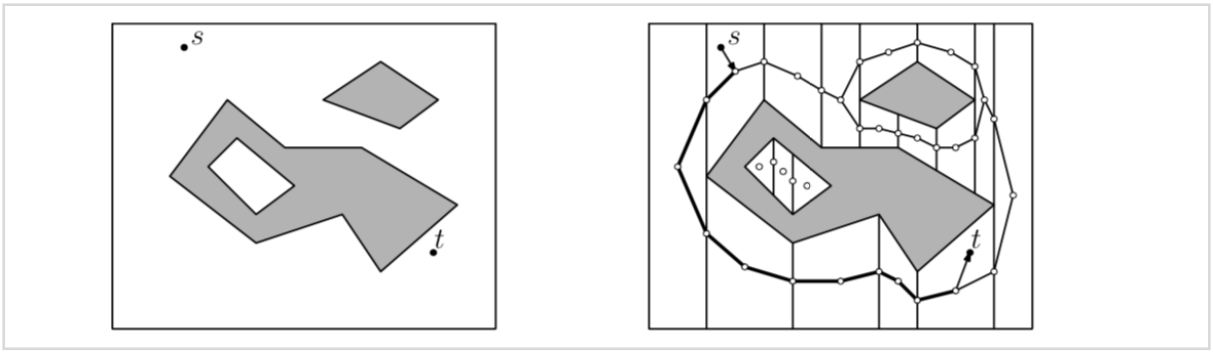
- Remember, the *work space* is the place where the robot and obstacles reside. The *configuration space* is the space of possible configuration vectors.
- Figuring out the motion of the robot reduces to computing a path in the configuration space.



- Let S be the work space, maybe the plane with polygonal obstacles.
- *Forbidden configurations* are ones where the robot intersects obstacles.
- We denote the set of forbidden configurations as C_forb(R, S).
- All other configurations are called *free configurations*. These are denoted C_free(R, S) which is sometimes called the *free space*.
- Almost there: In the *motion planning problem*, you are given a robot R, a work space S, and two configurations s and t. Can we go from configuration s to t with intersecting the obstacles? Or, can we move from point s to t while staying in C_free(R, S)?
- We'll stick to the case of R and every obstacle being a polygon. The only moves we can do are translations, so the configuration is just the (x,y)-coordinates of some reference point in R.
- You can imagine mapping each obstacle to a larger polygon in the configuration space which represents all positions of the reference point where the robot would intersect the polygon.
- These are called *configuration obstacles* or *C-obstacles* for short. We'll formally talk about computing them later.

Point Robots

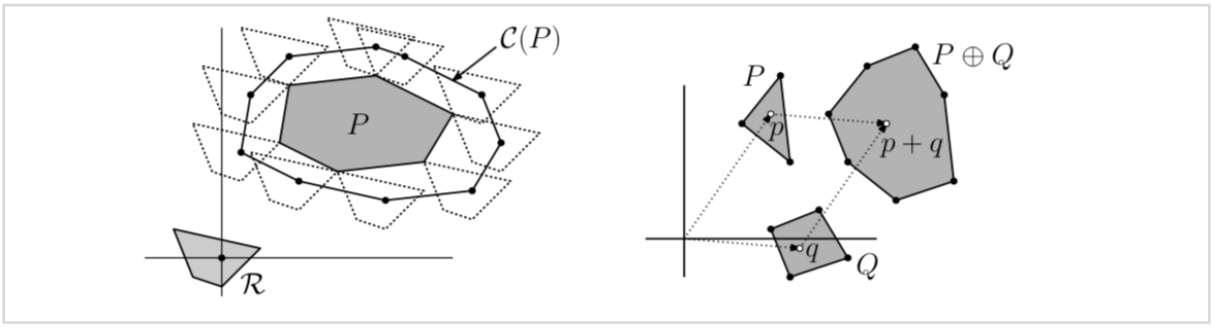
- For now, though, let's consider the easier problem of motion planning when R is a single point, but the obstacles are arbitrary polygons.
- Here, the free space is identical to the part of the work space outside of the obstacles.
- We need to move our point robot from s to t in the free space.



- To do so, we start by subdividing the free space into simple convex regions.
- So we'll compute a trapezoidal map of the line segments bounding the polygons, then throw away trapezoids inside the polygons.
- Now we create a planar graph called the *road map* using the subdivision. Put a vertex in the middle of each trapezoid and trapezoid wall. Add edges between trapezoid center vertices to the vertices on the trapezoid's walls.
- Add s and t as vertices and add edges from s and t to the center vertices of their trapezoids.
- Finally, we do a breadth-first search from s and see if we can reach t. If so, the embedding of the edges gives us the path of the point robot.
- This whole algorithms takes O(n log n) expected time if the obstacles have n vertices total.

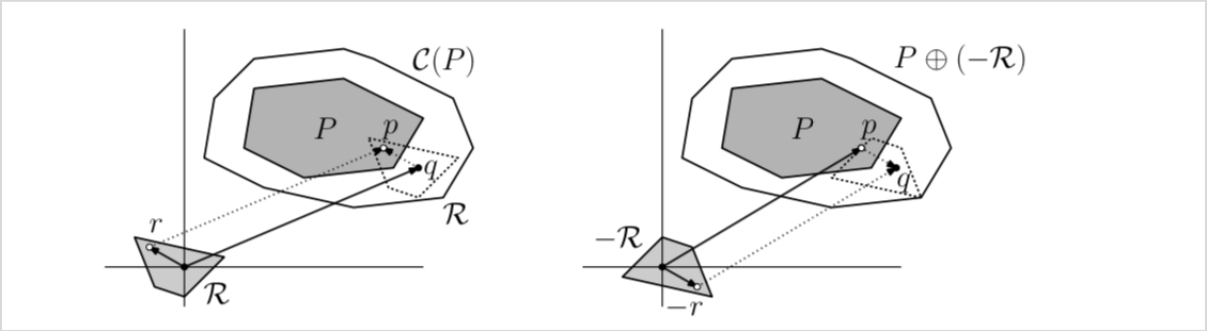
Polygon Robots

- But what if our robot is a polygon we can translate? How do we compute the free space?
- Suppose the reference point of robot R is the origin.
- R(p) denotes the *translate* of the robot so the reference lies at point p.
- Let P be an obstacle. Its corresponding C-obstacle is the set of placements of R that cause it to intersect P. Denote it as C(P) = {p : R(p) intersect P ≠ emptyset}.
- You can think of C(P)'s boundary as the path the reference point takes as we "scrape" R along the outside of P.



- OK, it sort of looks like R is pushing us away from P or maybe we're kind of "subtracting" R from P.
- We can formalize it as follows:
- Given two point sets P and Q in the plane, their *Minkowski sum* is the set of all pairwise sums of points taken from each set.

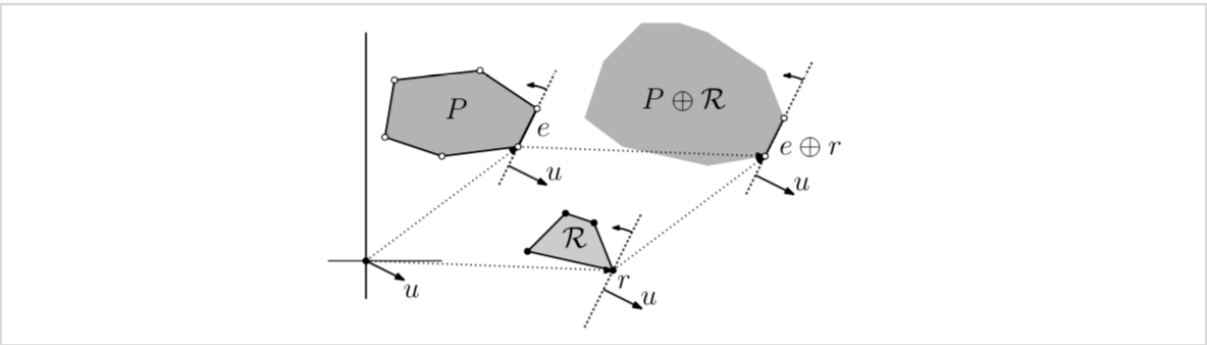
- It is denoted as P oplus Q = {p + q : p in P, q in Q}.
- And given point set S, let -S = {-p : p in S}.
- Note the translate of R by vector p is R(p) = R oplus {p} = R oplus p.
- Claim: C(P) = P oplus (-R)
- Proof: q in C(P) if and only if exists r in R and p in P such that p = r + q if and only if exists -r in -R and p in P such that q = p + (-r) or q in P oplus (-R).
    - So the set of such q is C(P) = P oplus (-R).



- So to compute the free space, we need to compute Minkowski sums.

Computing Minkowski Sums

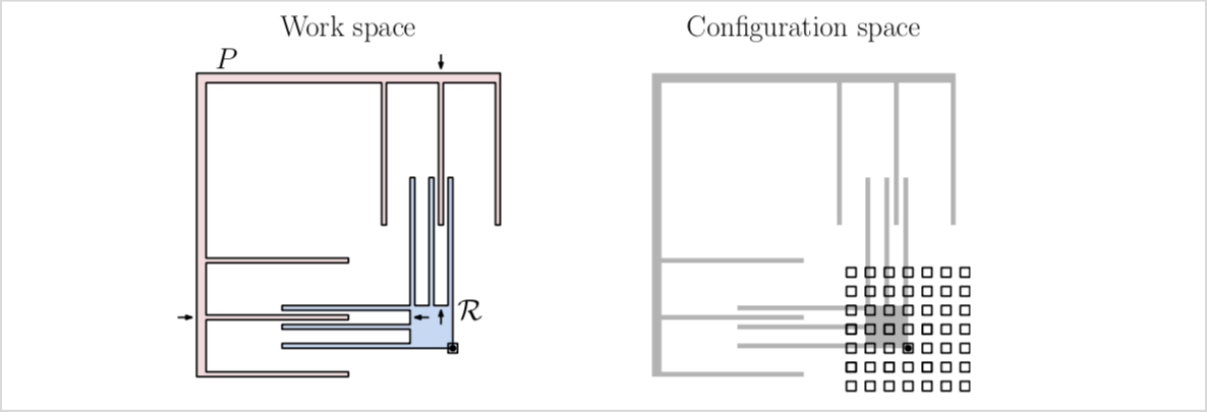
- We'll start with a simple case. Computing P oplus R where P and R are convex with n and m vertices, respectively.
- First off, P oplus R is convex, pretty much by definition of convexity.
- Given vector u, a point p in P is *extreme* in direction u if it maximizes dot product p dot u.
- Observation: The set of extreme points P oplus R in direction u is the set of sums of points p and r that are extreme in direction u for P and R, respectively.
- I'm leaving details for the book, but we can compute P oplus R by doing an *angular sweep*. Where we continuously change the direction of u going counterclockwise around a circle. As we do so, we maintain the most extreme point of P and R.



- When u goes perpendicular to an edge of P or R, we add the edge to the current vertex of the other polygon.
- Each edge of P or R contributes one edge to P oplus R, so the algorithm runs in O(m + n) time.

Complexity of Minkowski Sums

- So great, we know the free space is the stuff outside these Minkowski sums. And we can even compute them in the simplest case. But it turns out Minkowski sums can have a lot of edges in more general cases. We should learn how bad things can get so we know which cases are practical in robot motion planning.
- It turns out thing can get pretty bad! Suppose P and R are just arbitrary (non-convex) polygons.
- We can get a rough *upper bound* on how things get as follows. First off, P and R are the unions of n - 2 and m - 2 triangles, respectively.
- P = union_{i=1}^{n-2} T_i and R = union_{j=1}^{m-2} S_j => P oplus R = union_{i=1}^{n-2} union_{j=1}^{m-2} (T_i oplus S_j).
- So the Minkowski sum is the union of O(mn) polygons, each of constant complexity. So that's O(mn) sides total. At worst, they have O(m^2 n^2) intersection points which is an upper bound on the number of vertices in P oplus R.
- So that's an upper bound, but can things really be that bad? Yes.
- Consider this example. P and R have n and m teeth respectively.



- We can fit any pair of teeth from P between any pair of teeth from R, but when we do this, we can't move to different pairs of teeth without causing a collision.
- It looks like the free space has Omega(m^2 n^2) connected components, so the Minkowski sum has at least this many vertices.
- Alright, but maybe it's not so bad if we assume at least R is convex.
- The convexity of R actually helps a lot.
- I may or may not prove this theorem on Thursday depending on how long other things will take.
- Theorem: Let R be a convex m-gon and P a simple n-gon. Then P oplus R has total complexity O(mn).
- In fact, the proof implies that even if P is several disjoint simple polygons with n vertices total, the complexity is still O(mn).

Motion Planning with a Simple Convex Robot

- So let me get to the main application of all of this from the book. We're given a convex robot R with O(1) sides, a set of polygonal obstacles P with n sides total, and two configurations s and t.
- To figure out if we can get from s to t, we first triangulate each polygon.
- Then we compute Minkowski sums between R and each triangle to get the C-obstacles from each triangle.
- We then compute the whole *forbidden space* by taking the union of these constant complexity C-obstacles. The book offers an O(n log^2 n) time divide-and-conquer algorithm.
- Now we understand the whole forbidden space, and it has complexity O(n).
- Motion planning with R is the same as moving a point from s to t through the free space. So we use the earlier point robot algorithm on the compliment of the forbidden space.
- The whole thing takes O(n log^2 n) time total.