EECS C106B / 206B Robotic Manipulation and Interaction **Spring 2020**

Lecture 9: (Optimal Control, Model Predictive Control)

Scribes: Mark Scheble, Michael Wu

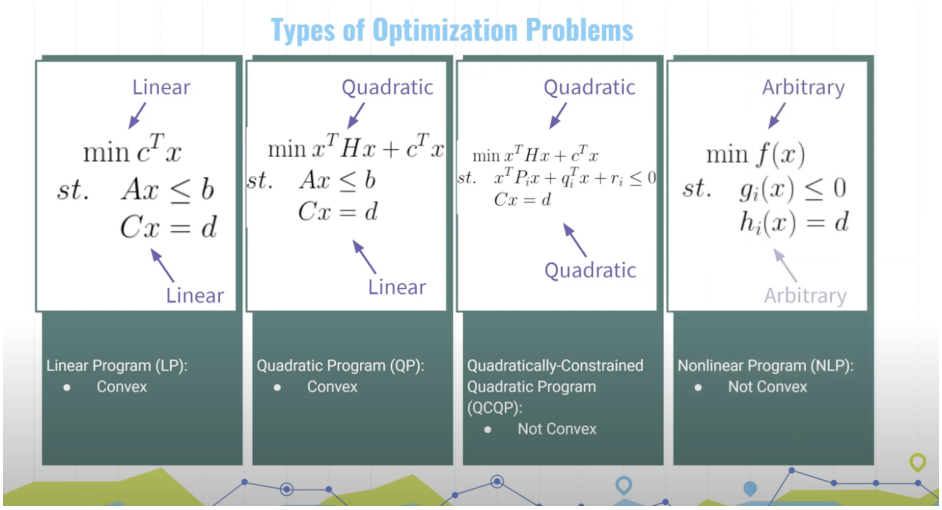
9.1 Review of Nonholonomic to Optimization Problem

9.1.1 Path Planning as Optimization Cost

$$
\begin{aligned}
& argmin_{x_t,u_t} \sum_{0}^{T} cost(x_t, u_t) \\
s.t. \quad & constraints(x_t, u_t) <= 0 \\
& x_{t+1} = f(x_t, u_t \\
& x_0 = x_{start} \\
& x_T = x_{goal}
\end{aligned}
$$

9.1.2 Types of Optimization Costs

- Linear Program (LP) → Convex
- Quadratic Program (QP) → Convex
- Quadratically - Constrained Quadratic Program (QCQP) → None Convex
- Non linear program (NLP) → None Convex



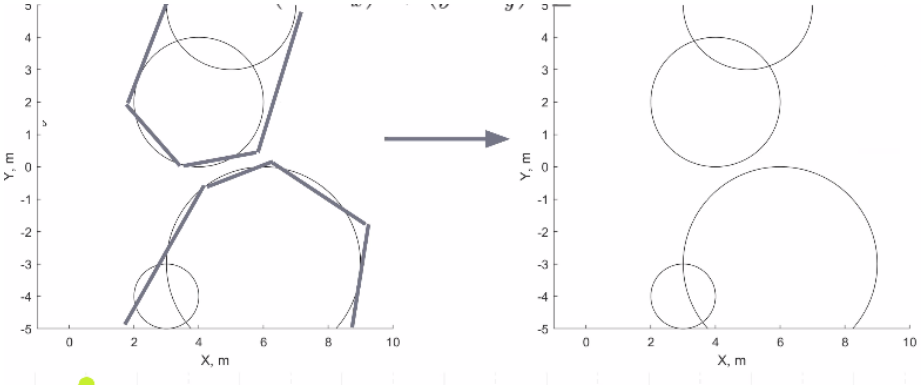
Optimization Based Planners commonly used to smooth RRTs. Everything in this lecture is in NLP area.

9.1.3 Obstacle as Constraints: Spheres

Take obstacles and represent them as "tiled" spheres which are just quadratic constraints. Need to be careful of way points on either side of quadratic constraints such that path goes through obstacle as system may phase since both points are not colliding and valid points. Quadratic constraints with radius r are the easiest to program and use for optimization.

$$(x - c_x)^2 + (y - c_y)^2 >= r^2 \tag{9.1}$$

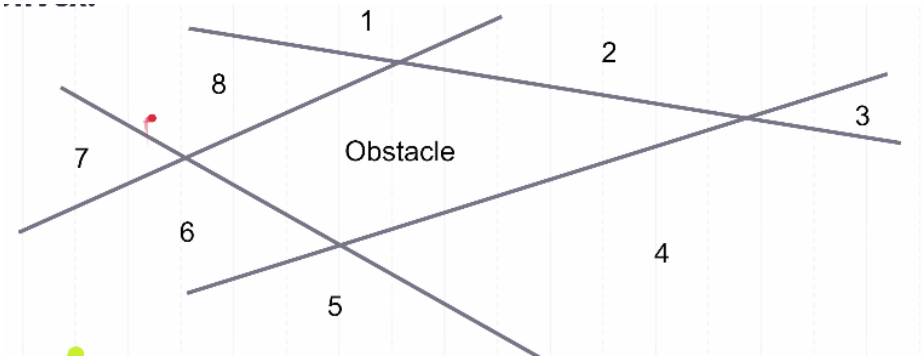
When phasing through sphere, constraints are not being ignored. However, the way we represented our system, the constraints are satisfied (by not colliding with the obstacle) but the system is not doing what we would like. We are assuming the obstacles are small enough that we won't run into this problem (taking real world problems to simpler model).



9.1.4 Obstacle as Constraints: Polytopes

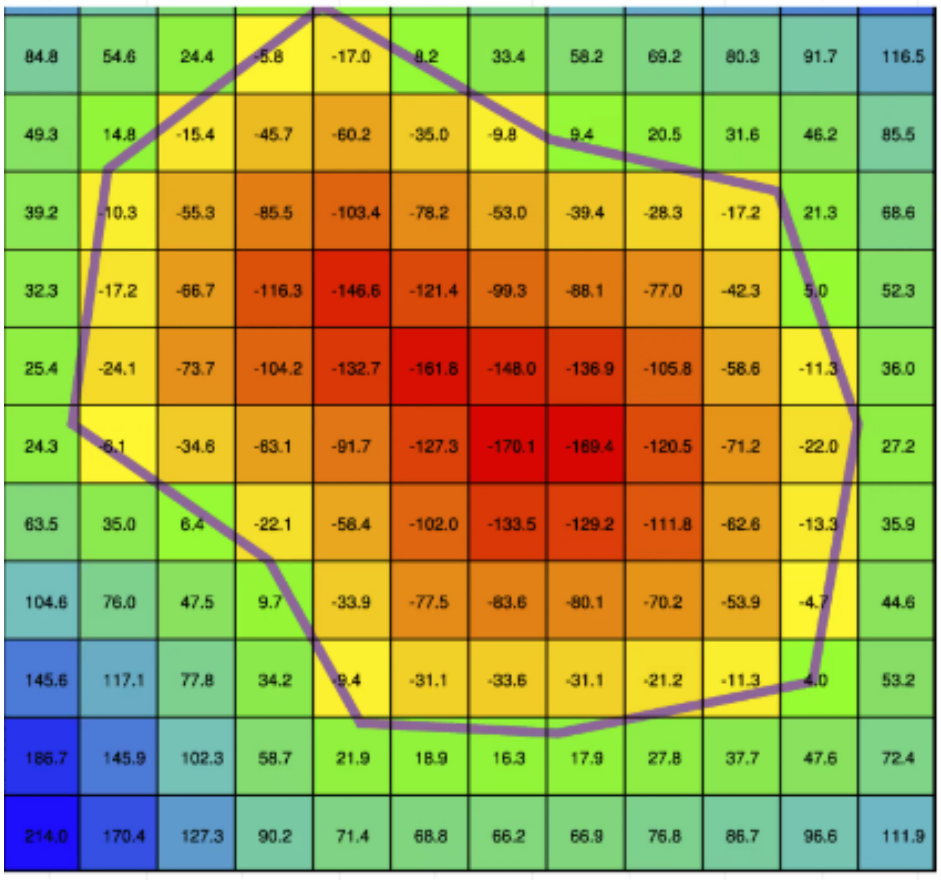
Polytopes - generalized polyhedron

Constraints that make up object are linear, but we have to specify convex regions. This make the problem a Mixed Integer Program, and therefore Not Convex. These problems get very big very fast as you have to choose a convex region for each way point.



9.1.5 Obstacle as Constraints: Signed Distance Fields

- Signed distance fields require interpolating over a lookup table, so certain solvers don't work well with them.
- They also suffer heavily from the curse of dimensionality.
- This so we know how badly we are colliding with an item. Tend to only work with low DOF systems as we run into storing errors.
- We're listing closest distance to obstacle in grid (-) inside and (+) outside. This allows us to know which way to move to avoid obstacle.



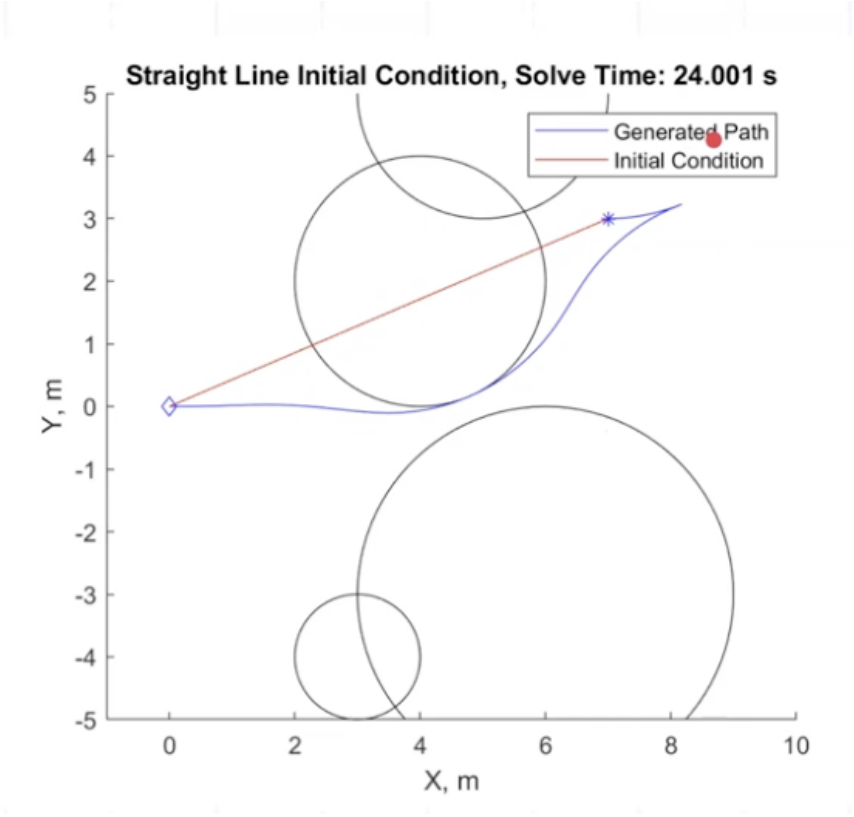
Hoping to represent system as something simpler and solvable. Constraints are never going to be perfect but will help simplify the system.

9.1.6 Implementation Tips for Optimization Based Planners: Warm-Starting

Warm start- initialize your path with a guess of what the solution will look like, start gradients here

- Solution to a simplified planning program (like standard RRT).
- A straight line (through obstacles), easiest to implement and works reasonably well.

- It doesn't need to be right (or possible). It just needs to be better than nothing.
- Do not instantiate randomly.



Valmik ran system with random initialization and computer crashed after 30 minutes showing straight line guessing is sufficient(took 24 seconds to solve)

9.2 Sampling

Optimization works great if we can represent constraints nicely (linearly). However, this is not always sufficient.

In driving, sometimes its not easy to represent collision for non-circular obstacle. Aka if your robot can change direction, then representing the obstacle will be more difficult.

9.2.1 Sampling-Based Planners

- Building a graph by sampling way points, then find the shortest path in the graph.
- No explicit representation of free space needed.
- Scales well for high dimensional systems.
- Examples: RRT, PRM, Rabit, etc

9.2.2 RRTs for Nonholonomic Systems

1. Initialize an empty graph G and add the start configuration
2. Until we either succeed or reach the maximum iterations, do:
   (a) sample a configuration C_{rand} from the configuration space
   (b) Find the nearest node C_{near} in G to C_{rand}
   (c) Construct a local plan p that steers the robot from C_{near} to C_{rand}. If this plan is in collision with the environment, discard it
   (d) Take some small prefix of this plan p* which ends at C_{new}. Add C_{new} to G with P* as an edge.
   (e) If C_{new} is within epsilon of goal g, then we have success

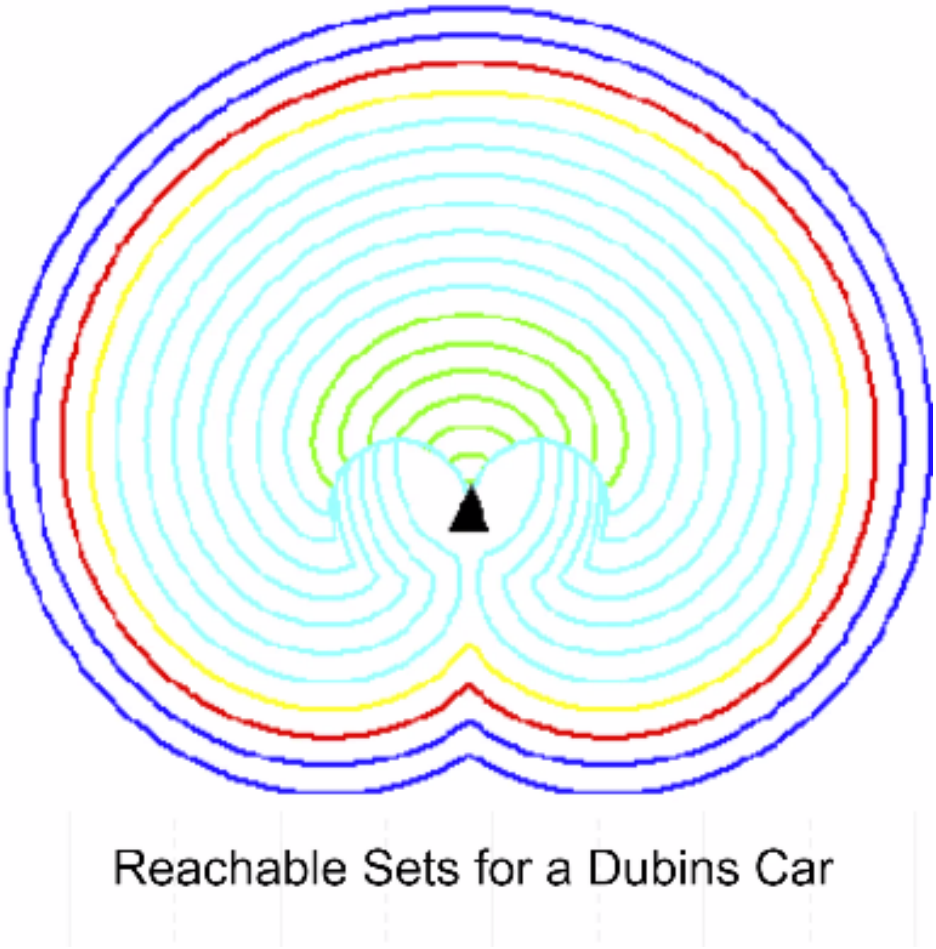
9.2.3 Problems with RRTs

- A little too slow, efficiency is quite slow (trade off).
- This algorithm assumes that generating paths are very fast.
- Build up Error similar to Euler's method, not algorithm dependent but rather model dependent
- More samples, more probability of finding a path that exists. However, RRTs not guaranteed to give solution because they are probabilistically complete.
- Non holonomic system lack reliable distance metrics. 2b) is where most error in this algorithm occurs
- How to define local plan.

9.2.4 Distance Metrics in RRTs

What is the Closest Node? 2b) problem can be defined in multiple ways below

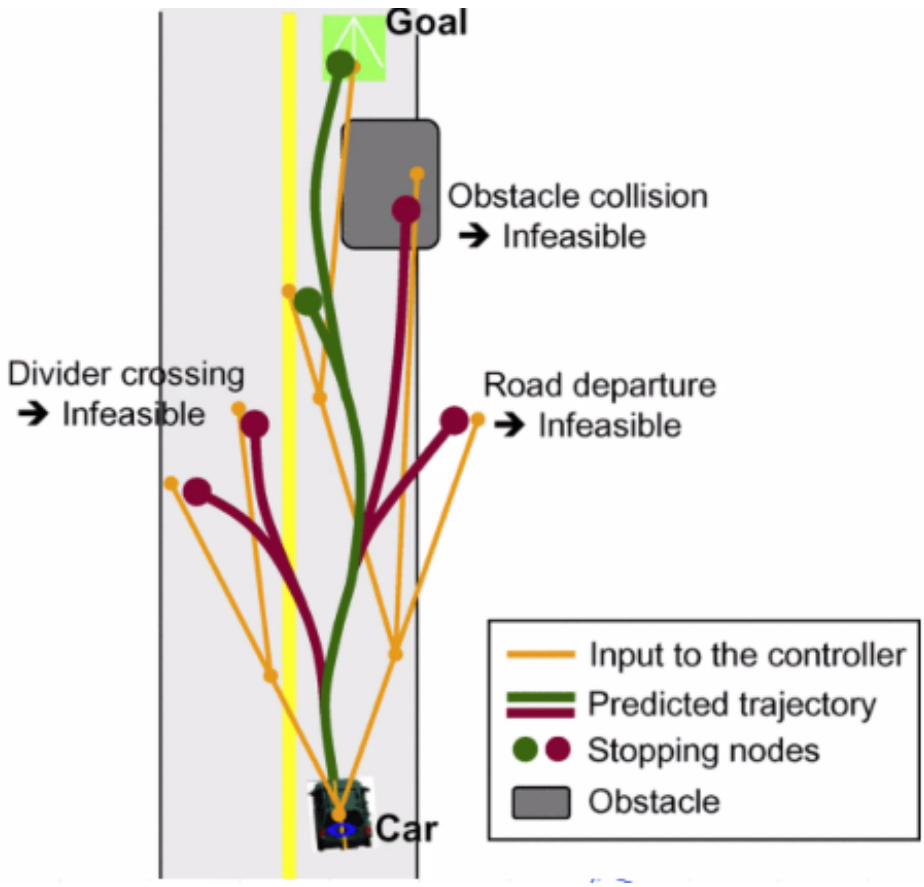
- Euclidean Distance?
- Local Plan length?
- Heuristics?



9.2.5 Local Planners

9.2.5.1 Motion Primitives

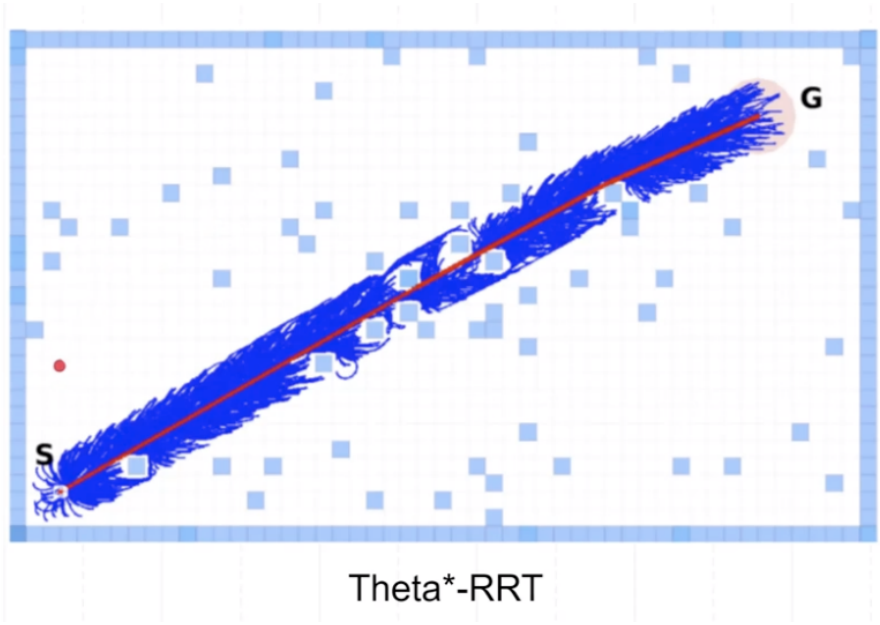
- Splines
- Sinusoids
- Dubins/Reed-Shepp Paths



9.2.6 Implementation Tip: Path Biasing

Trying to smooth RRT, concentrate sampling towards "useful" configurations

- Concentrate samples towards "useful" configurations
- Goal Biasing: Periodically sample the goal
- Path Biasing: Concentrate sampling around a nominal path
- Theta* - RRT, A*-RRT



A* does not work because of the curse of dimensionality and scales quickly. Think about if you have a really long trailer – Yes, you can probably navigate the front of the vehicle around an obstacle, but it would be really hard to predict for the trailers.