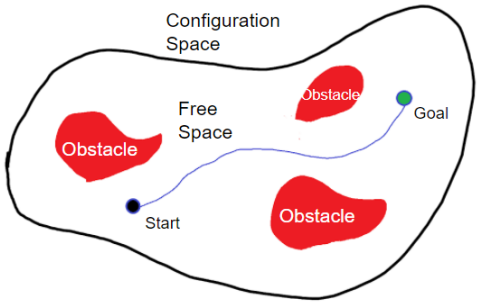
EECS C106B / 206B Robotic Manipulation and Interaction **Spring 2021**

Lecture 8: (A Review of Robot Motion Planning)

Scribes: Alex Zhou, Gary Yang

8.1 The goal of motion planning

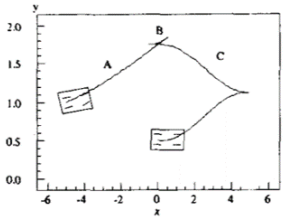
To find a path through the free space that links the start and goal. Formally defined, the problems include planning to satisfy dynamic/kinematic constraints and planning around obstacles. Solutions to this include gridding (which is limited since it discretizes the space, and apply path finding algorithms, for example, A*), some classical methods (expanded upon later), optimization based methods, and finally sampling based methods.

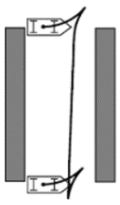


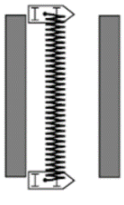
8.2 Classical techniques

8.2.1 Sinusoids

Sinusoids is the first classical technique. The robot use sinusoids at integrally related frequencies to achieve motion at a given bracketing level.The pros of this is that the trajectories are somewhat easy to compute as opposed to other methods, and the math is constructive and complete. However, it suffers from the fact that it cannot handle obstacles and is sometimes aggressively suboptimal. For example, a car that only needs to move in the y direction for parking may exhibit the behavior shown below where it moves in a oscillating fashion, since the x and orientation are already at the target position.

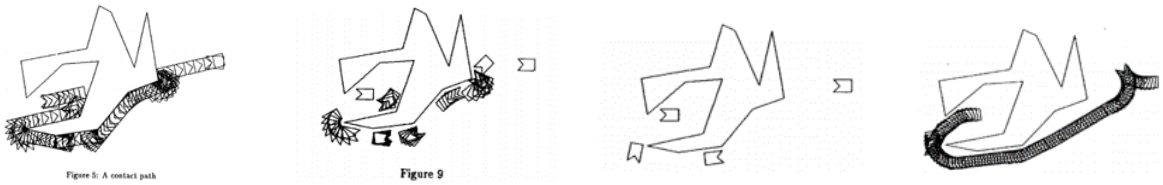






8.2.2 Jacobs-Laumond

The Jacobs-Laumond method ignores constraints, produce a path which stays at least δ away from obstacles, approximately following this path with nonholonomic motions, then iteratively smoothen and refine the trajectory. It works reasonably well, but not complete, and can result in jagged paths if smoothing does not work.



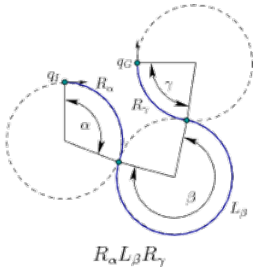
8.2.3 Dubins/Reeds-Shepp paths

Dubins/Reeds-Shepp paths are a collection of lines and arcs of constant radius. It is optimal but only work for Dubins/Reeds-Shepp cars (for example, Turtlebot), and does not consider obstacles.

8.2.3.1 Dubins car

The Dubins car moes forward at constant speed, and has the following dynamics:

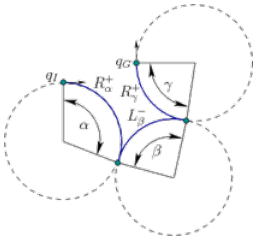
$$\begin{aligned} \dot{x} &= u_i \cos\theta \\ \dot{y} &= u_2 \sin\theta \\ \dot{\theta} &= u_2 \\ u_1 &\in \{0, 1\} \\ u_2 &\in [-1, 1] \end{aligned} \tag{8.1}$$



8.2.3.2 Reeds-Shepp car

The Reeds-Shepp car can move both forwards and backwards, and has the following dynamics:

$$\begin{aligned} \dot{x} &= u_i \cos\theta \\ \dot{y} &= u_2 \sin\theta \\ \dot{\theta} &= u_2 \\ u_1 &\in \{-1, 0, 1\} \\ u_2 &\in [-1, 1] \end{aligned} \tag{8.2}$$



8.3 Optimization

8.3.1 Path Planning as an Optimization Problem

The goal here is to turn path planning into an optimization problem. The most general formulation is as follows: We want to find continuous functions $x(t), u(t)$ that satisfy

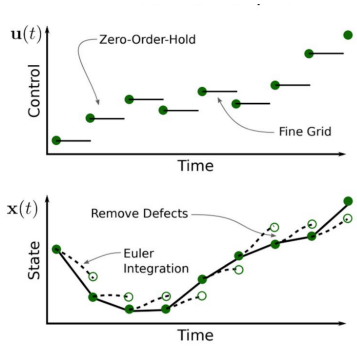
$$\begin{aligned} \arg \min_{x(t),u(t)} & \int_0^T cost(x(t), u(t))dt \\ & s.t. constraints(x(t), u(t)) \leq 0 \\ & \dot{x}(t) = f(x(t), u(t)) \\ & x(0) = x_{\text{start}} \\ & x(T) = x_{\text{goal}} \end{aligned} \tag{8.3}$$

8.3.2 Representing Paths

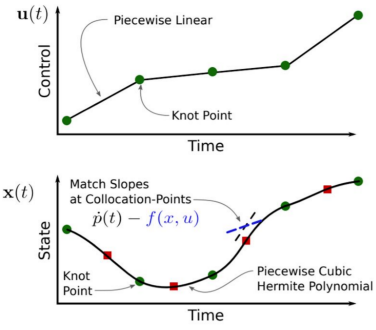
It is not possible to solve such optimization problems in general using continuous paths. To solve this, we can parameterize paths.

8.3.2.1 Waypoints

The most common way of parameterizing a path is to use waypoints and discretize the system. For linear systems, we can just integrate the dynamics, but this is too expensive for nonlinear dynamics. For nonlinear dynamics, the most common methods for discretizing the dynamics are **Euler discretization** (zero-order input), where you use a first order approximation of the dynamics (so $x_{t+1} = x_t + f(x_t, u_t)\delta t_t$), and **direct collocation**. Two methods of collocation are trapezoidal collocation, which takes first-order input, and Hermite-Simpson collocation, which takes second-order input.



Euler Discretization. Inputs are step functions. Uses first order approximation of dynamics.



Trapezoidal Collocation. Assume control inputs are piecewise linear or piecewise quadratic.

With waypoints and discretized dynamics, our optimization problem becomes as follows:

$$\begin{aligned} \arg\min_{x_t,u_t} & \sum_{0}^{T} cost(x_t, u_t) \\ & s.t. constraints(x_t, u_t) \leq 0 \\ & x_{t+1} = f(x_t, u_t) \\ & x_0 = x_{\text{start}} \\ & x_T = x_{\text{goal}} \end{aligned} \tag{8.4}$$

8.3.3 Optimality

8.3.3.1 Self-Driving Cars

There are many things to take into consideration when thinking about what makes a path optimal. This can include path length, path time, smoothness, and fuel efficiency, to name a few. What you ultimately choose as your cost function depends on the task you are trying to perform, and there may be many cost functions that work.

8.3.4 The Linear Quadratic Regulator

Suppose $f(x, u) = Ax + Bu, cost(x, u) = x^T Qx + u^T Ru$ and $T = \infty$. Then, the optimization problem LQR tries to solve is

$$\begin{aligned} \arg\min_{x_t,u_t} & \sum_{0}^{\infty} x_t^T Q x_t + u_t^T R u_t \\ & s.t. constraints(x_t, u_t) \leq 0 \\ & x_{t+1} = Ax_t + Bu_t \\ & x_0 = x_{\text{start}} \end{aligned} \tag{8.5}$$

The path you get as the solution to LQR is the motion of the system if we were to use an LQR controller.

8.3.5 Model Predictive Control (aka Receding Horizon Control)

Model predictive control (MPC) is essentially the following procedure:

1. Given a system model, find the best sequence of system inputs for the next T timesteps.
2. Execute the first input in the sequence
3. Repeat

We call parameter T the horizon, hence receding horizon control.

8.3.6 Path Planning versus Control

The distinction between motion planning and control can be unclear, especially when considering how MPC seems similar to re-performing path planning at every timestep. From lecture, these are the general distinctions between the two categories.

Motion Planning

- Longer time scales
- High fidelity/very abstract models
- Detailed obstacle models
- Potentially larger search space

Control

- Short time scales
- Simplified models or environments for fast computation
- Potentially smaller search space
- Often "warm-started" with nominal control

Note that in optimization, often path planning and control are the same thing.

8.3.7 Path Planning Hierarchy: A Self Driving Car

This is a sample hierarchy for a self-driving car. In this class, we are most focused on the middle two levels, which are the path planner and steering controller.

8.3.8 Types of Optimization Problems

In general there are 4 types of optimization problems: linear programs, quadratic programs, quadratically-constrained quadratic programs, and nonlinear programs. These are good to know about, because there are already very good out-of-the-box optimization solvers that you can use to solve these.

8.3.8.1 Linear Programs

A linear program (LP) is formulated as follows:

$$\begin{aligned} &\min c^T x \\ &s.t. Ax \leq b \\ &\quad Cx = d \end{aligned} \tag{8.6}$$

They have linear cost and linear constraints. They are convex, meaning you'll have basically a constant time guaranteed solution to the problem.

8.3.8.2 Quadratic Programs

Another type of convex problem is quadratic programs (QP). One example would be linear least squares. In the context of this course, this type of problem can be found in grasping, drones, and many other areas in robotics. The problem is formulated as follows:

$$\begin{aligned} &\min x^T H x + c^T x \\ &s.t. Ax \leq b \\ &\quad Cx = d \end{aligned} \tag{8.7}$$

They have quadratic cost and linear constraints.

8.3.8.3 Quadratically-Constrained Quadratic Program

Quadratically-Constrained Quadratic Programs (QCQP) are not convex. They have both quadratic cost and quadratic constraints. However, as with convex problems, there are solvers specifically made for QCQP's that are pretty efficient. These problems show up often in discrete control, and is somewhat common in robotics. They are formulated as follows:

$$\begin{aligned} &\min x^T H x + c^T x \\ &s.t. x^T P_i x + q_i^T x + r_i \leq 0 \\ &\quad Cx = d \end{aligned} \tag{8.8}$$

8.3.8.4 General Nonlinear Program

A general nonlinear program is not convex and has arbitrary cost and arbitrary constraints, with no guarantees about finding an optimum (can get stuck at local minima). This is an NP-hard problem. Nonholonomic path planning problems are all nonlinear programs.

$$\begin{aligned} &\min f(x) \\ &s.t. g_i(x) \leq 0 \\ &\quad h_i(x) = d \end{aligned} \tag{8.9}$$

8.3.9 Obstacles as Constraints

8.3.9.1 Spheres

We can approximate our obstacles as spheres. For example, in $\mathbb{R}^2$, this would be

$$(x - c_x)^2 + (y - c_y)^2 \geq r^2$$

Such modeling gives us quadratic constraints, which means we would at best have a QCQP (not convex).

8.3.9.2 Polytopes

We can also model our obstacle constraints as a choice of different convex regions/linear constraints. This makes the problem a mixed integer program which is not convex, and can be difficult to solve because we have no gradient information to work with.