C106B - Robotic Manipulation and Interaction **(Week 5)**

Discussion #5

Author: Amay Saxena

Problem 1 - Sinusoidal Inputs

1. Consider the sinusoidal function

$$f(t) = A\sin(\frac{2\pi}{T}t + \phi) \tag{0.1}$$

where A, T, ϕ are constants. For some arbitrary constant $c \in \mathbb{R}$, evaluate

$$\int_c^{c+T} f(t)dt \tag{0.2}$$

Directly evaluating the integral, we get

$$\begin{aligned}
\int_c^{c+T} A\sin(\frac{2\pi}{T}t + \phi)dt &= [-\frac{AT}{2\pi}\cos(\frac{2\pi}{T}(c+T)+\phi)] - [-\frac{AT}{2\pi}\cos(\frac{2\pi}{T}c+\phi)] &(0.3)\\
&= [-\frac{AT}{2\pi}\cos(\frac{2\pi}{T}c+\phi+2\pi)] - [-\frac{AT}{2\pi}\cos(\frac{2\pi}{T}c+\phi)] &(0.4)\\
&= [-\frac{AT}{2\pi}\cos(\frac{2\pi}{T}c+\phi)] - [-\frac{AT}{2\pi}\cos(\frac{2\pi}{T}c+\phi)] &(0.5)\\
&= 0 &(0.6)
\end{aligned}$$

What's the relevance of this? We have showed that the integral of an arbitrary sinusoidal signal over one period is going to be 0. In planning problems, we often have control over the derivatives of one of our state variables. Say for state variable x, we have direct control over $\dot{x}$ If we set the derivative of a state variable to be some sinusoidal function with period T, then we will have

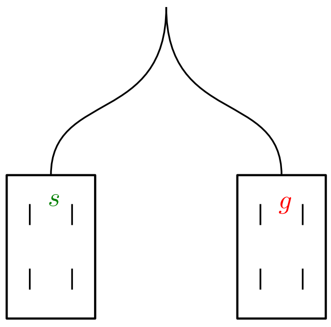
$$x(T) = x(0) \tag{0.7}$$

It's often the case in a dynamical system that the derivative of another state variable, say y, will be coupled in with value of x in some way. Loosely speaking, the same control input will likely result in

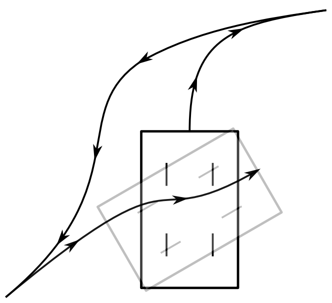
$$y(T) \neq y(0) \tag{0.8}$$

So consider the case when we have x as its desired value, but we still need to change y. Using sinusoidal inputs let us modify one state variable without affecting the other!

2. Describe planning problems for a bicycle modeled car in which you expect some state variables to remain constant and other to change. Consider how sinusoidal inputs are used to accomplish this. For our bicycle modeled car, our inputs are the steering velocity and linear velocity of the car. Using out-of-phase sinusoidal inputs lets us accomplish:



(a) Parallel Park


(b) Point Turn

Think about how executing these motions involves a back-and-forth sinusoidal motion for both steering and linear velocity inputs.

Problem 2 - Randomly Exploring Random Trees (RRT)

The RRT algorithm is as follows:

Algorithm 1 The RRT algorithm

```
Graph.add_node($q_{init}$)
while $goal \notin Graph$ do
    $q_{rand} \longleftarrow$ Sample_Configuration()
    $q_{near} \longleftarrow$ Nearest_Vertex($q_{rand}$, Graph)
    $q_{new} \longleftarrow$ Local_Planner($q_{near}$, $q_{rand}$)
    if NOT Check_Collision($q_{new}$) then
        Graph.add_node($q_{new}$)
        Graph.add_edge($q_{near}$, $q_{new}$)
    end if
end while
return Graph
```

1. What parts of this algorithm would need to be designed/changed in order to use it with the bicycle model car?
   There are many concerns to consider when using the RRT algorithm for a robot with state variables in different units and velocity constraints.

   - Sampling: A simple approach is just uniformly over all state variables (sampling from a 4D cube). However, performance might be improved by sometimes sampling within some radius of the goal (sampling from 4D sphere). However, what does "distance" mean in this context? This brings us to...
   - Distance: We are dealing with both Euclidean distance for the xy position of the car, and angular distance for the heading. There's no right way to combine these two, but what we do should be meaningful in terms of how "hard" it is to get from one configuration to another. A similar-problem has been solved for mobile which have a fixed turning radius - take a look at Dubins paths or Reeds-Shepp curves.
   - Local Planner: Velocity constraints make this planning non-trivial. We need to figure out the control inputs to move our robot at least in the direction of the q_{near} node. Resorting to heuristics might work, or even trying random inputs and choosing the set which results in the best progress towards our goal point. Some planning approaches for the car system which do not take obstacles into account (ex: steering with sinusoids) may also work well.

2. You are using the RRT planner to plan a path for the Baxter robot. How would you go about implementing collision-checking? Would you expect an RRT to work better or worse for an open-chain manipulator like Baxter as opposed to a bicycle model car? (*Hint: what velocity constraints exist for the Baxter in jointspace? workspace?*

   For this question, we assume our state space for the Baxter is a vector of joint angles

$$q = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \tag{0.9}$$

   - Collision-Checking: Rudimentary collision checking for the end-effector can be performed via the forward-kineatics map: we can compute the end-effector position in $\mathbb{R}^3$ and verify that this is not within an obstacle.
   - Comparison: For the bicycle model car, we have Pfaffian constraints that restrict the instantaneous motion of the car. However, the Baxter arm theoretically can set its vector of joint velocities to any vector in $\mathbb{R}^n$. If our manipulator jacobian is full rank, this means we can also set the workspace velocity to any vector in $\mathbb{R}^6$ as well. The local planning step becomes much simplified here as a

result. All of our state variables are in the same units as well, making it more straightforward to come up with a distance metric. These properties make the RRT algorithm generally work better for open-chain manipulators versus a bicycle modeled car.

3. Assume we are using the RRT algorithm for a turtlebot to plan from some starting configuration to an ending configuration with state limits

$$-10 \leq x \leq 10 \tag{0.10}$$
$$-10 \leq y \leq 10 \tag{0.11}$$
$$-\pi \leq \theta \leq \pi \tag{0.12}$$

Assuming that we sample states uniformly in this state space, how many times do we need to sample until we expect to sample a state within 1 unit of the goal configuration?
We assume we are just measuring 1 unit in Euclidean distance for the car's xy position. In this case, we can ignore the bounds on θ as the value we sample for θ does not determine how far our car is from the goal. We can view this problem as the probability of sampling a circle of radius 1 within a square of side length 20. In this case, the circle has area π and the square has area 400. This makes the probability of sampling a point within 1 unit of the goal

$$p_{sample} = \frac{\pi}{400} \approx .0078 \tag{0.13}$$

This translates to the expected amount of times we need to sample before sampling a point close to our goal as

$$\mathbb{E}[num\ sample] = \frac{1}{p_s ample} = \frac{400}{\pi} \approx 127 \tag{0.14}$$

Meaning we expect our RRT algorithm to undergo 127 iterations before we sample a point close to our goal. This may cause us to plan paths through regions of our state space we don't care about, motivating a sampling approach which biases state configurations closer to our goal. enumerate

Problem 3 - Optimization based path-planning

We will be converting the path planning problem into the following nonlinear optimization problem:

$$\begin{aligned}
q^*, u^* = \operatorname{argmin} \sum_{i=1}^{N} \left((q_i - q_{goal})^\top Q(q_i - q_{goal}) + u_i^\top R u_i\right) + (q_{N+1} - q_{goal})^\top P(q_{N+1} - q_{goal}) \\
s.t.\ & q_{min} \leq q_i \leq q_{max}, && \forall i \\
& u_{min} \leq u_i \leq u_{max}, && \forall i \\
& q_{i+1} - F(q_i, u_i) = 0, && \forall i \leq N \\
& (x_i - \text{obs}_{j_x})^2 + (y_i - \text{obs}_{j_y})^2 \geq \text{obs}_{j_r}^2, && \forall i, j \\
& q_1 = q_{start} \\
& q_{N+1} = q_{goal}
\end{aligned}$$

For Q, R, P all positive semi-definite matrices. Here, we *discretize* the problem over $N+1$ timesteps, with the first indicating the current time, and the last indicating the time at which we intend to reach the goal. q is an array of $N+1$ states (x, y, θ, ϕ) with $q_i = (x_i, y_i, \theta_i, \phi_i)$ for $i = 1, ..., N+1$, and u is an array of N inputs (v, ω). We don't have an input at the last state. obs is an array of obstacles. In order to simplify computation, we assume that all obstacles are circles with center (x, y) and radius r. We also assume that our robot is circular, and that its radius has been incorporated into the radii of all obstacles. $F(q, u)$ is the discrete dynamics of our system, $q_{k+1} = F(q_k, u_k)$.

(a) Explain, in words, what each constraint in the above formulation does.
The details are left to the homework, but on the surface level:

- We have some *inequality* constraints, which constrain some variables to be within a range. Geometrically, this defines some $n-$dimensional shape which contains all the possible values for our states.

- There are also some *equality* constraints, which constrain our variables to lie on certain sections on the boarder of the shape defined by the *inequality* constraints.

The collection of constraints define a *feasible region*, the set of all variables which satisfy the constraints of our optimization problem. The optimization solver then has to see where in this feasible region the *objective function* takes on a minimum or maximum value.

(b) Is the cost function we are using above convex?
Yes, we see that our cost function is defined as the sum of squared variables. This, in general, produces a convex function. Which constraints above are convex, and which are not? The details are left to the homework, but the only convex constraint are those which are made up of linear functions of state variables.

(c) Would it be possible to use a formulation like the above to path-plan for a 7DOF robot arm like the Baxter through an obstacle-rich workspace? Why or why not?
A similar optimization appraoch to the one we are using would likely not work for planning for a high-dimensional open-chain manipulator through an obstacle-rich workspace. The curse of dimensionality and potentially very complex obstacle constraints could cause our optimization problem to become very hard to solve. We would potentially need to throw the forward kinematics map into our constraints for obstacle checking, which our optimizer probably wouldn't like. enumerate