Lecture #6

Goals
1) Review MPC
2) Trajectory Optimization
3) Solving Nonlinear Programs
4) Collocation
5) Project
6) Free Time Optimization
7) Combining Everything Together

I. Review MPC

1. Talk through equations from last time
2. Talk through code

II. Trajectory Optimization

Def: A trajectory optimization problem is:

$$\min_{u:[0,1]\to\mathbb{R}^m,\ x_0\in\mathbb{R}^n} \int_0^1 L(t,x(t),u(t))\,dt + \phi(x(1))$$

Subject to
$$\dot{x}(t) = f(t,x(t),u(t)) \quad \forall t\in[0,1]$$
$$x(0) = x_0$$
$$g_i(x(t)) \le 0 \quad \forall t\in[0,1],\ \forall i\in\{1,\dots,p_{ie}\}$$

where $L:[0,1]\times\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}$ is the running cost and is assumed to be twice differentiable, $\phi:\mathbb{R}^m\to\mathbb{R}$ is the final cost and is assumed to be twice differentiable; $g:\mathbb{R}^n\to\mathbb{R}^p$

are inequality constraints and are also assumed to be twice differentiable, and the dynamics $f: [0,1] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is twice differentiable in all its arguments.

Note: Several strategies to solve this problem:

1. Indirect Methods:
   Develop a necessary condition for optimality, then discretize this condition to generate a nonlinear optimization program

---

Ex: $\min_{x \in \mathbb{R}} J(x)$

Necessary condition for optimality
$\nabla J(x) = 0$
So try to find zeros of $\nabla J(x)$

Unfortunately finding such necessary conditions for optimal control problems with constraints can be hard.

2. Direct Method:
   Discretize optimal control problem then solve using nonlinear programming.

Ex:

$u(0) \quad \dots \quad u(\Delta T)$

$u(2\Delta T$

$$\min_{\{u(\Delta Tk)\}_{k=0}^{\frac{1}{\Delta T}-1} \in \mathbb{R}^m} \Delta T \sum L(\Delta Tk, x(\Delta Tk), u(\Delta Tk)) + \varphi(x(1))$$

$x_0 \in \mathbb{R}^n$

subject to

have to plug in for $x(\Delta Tk)$:

$$x(0) = x_0$$

$$g_i(x(\Delta Tk)) \leq 0 \qquad \forall i \in \{1, \dots, P_{ie}\} \quad \forall k \in \{0, \dots, \tfrac{1}{\Delta T}\}$$

How do we solve this problem?

- Use nonlinear optimization.
- Need gradients of the cost function and the constraints to solve this problem.

III. Solving Nonlinear Programs

Recall

$$\min_{z \in Z} \quad J(z)$$

subject to $g_i(z) \leq 0 \quad \forall i \in \{1, \dots, P_e\}$

$h_i(z) = 0 \quad \forall i \in \{1, \dots, P_{ie}\}$

Note: 1) The top problem can be rewritten in this format:

$$z = \begin{bmatrix} u(0) \\ u(\Delta T) \\ \vdots \\ u(1-\Delta T) \\ x_0 \end{bmatrix} \quad \text{so} \quad Z = \mathbb{R}^{\left(\frac{m}{\Delta T}\right)} + \mathbb{R}^n$$

Notice that

$$x(\Delta T) = x(0) + \Delta T f(\Delta T, x(0), u(0)) = F_1(z)$$

Can similarly define $x(k(\Delta T)) = F_k(z)$

So the cost and constraints can be written as a function of z.

SNOPT, IPOPT, KNITRO, APMONITOR...

2) All nonlinear optimization solvers have similar format. We focus on fmincon:

$z = \text{fmincon}(J, z_0, A, b, A_{eq}, b_{eq}, lb, ub, \text{nonlcon}, \text{options})$

- J → cost
- z_0 → i.c. for optimization
- $Az \le b$
- $A_{eq} z = b_{eq}$
- $lb \le z \le ub$
- nonlcon → constraints → check (4)

3) $[J(z), \nabla J(z)] = J(z)$

WARNING: Matlab prefers its gradients as column vector rather than row vector

$$[\nabla J(z)]_{i1} = \frac{dJ}{dz_i}$$

4) $[g, h, \nabla g, \nabla h] = \text{nonlcon}(z)$

$g(i) = g_i(z) \leq 0 \quad \forall i \in \{1, \ldots, p_{ie}\}$

$h(i) = h_i(z) = 0 \quad \forall i \in \{1, \ldots, p_e\}$

$[\nabla g]_{ij} = \frac{\partial g(j)}{\partial z_i}$

$[\nabla h]_{ij} = \frac{\partial h(j)}{\partial z_i}$

WARNING: MATLAB takes the transpose of the Jacobian.

5) The last argument to fmincon is a set of options make sure to set them as follows:

```
options = optimoptions('fmincon',...
    'SpecifyConstraintGradient',true,...
    'SpecifyObjectiveGradient',true);
```

6) For our optimal control problem, we have to compute gradients of a constraint g:

$g(1) = g(F_1(z)) = g(x(0) + \Delta T f(\Delta T, x(0), u(0)))$

$g(2) = g(F_2(z)) = g(x(\Delta T) + \Delta T f(\Delta T, x(\Delta T), u(\Delta T)))$

but $x(\Delta T) = x(0) + \Delta T f(\Delta T, x(0), u(0))$

so have to substitute that in to expression so $g(1)$'s gradient is nonzero only for $x(0)$ and $u(0)$ term, whereas $g(2)$'s gradient is nonzero for $x(0)$, $u(0)$, and $u(1)$...

problem only gets worse as we go all the way to $g\left(\frac{1}{\Delta T}\right)$

IV. Collocation.

To avoid this problem of having to write a complex gradient, we will append the states of the system to the decision variables:

$$\min_{\{u(k\Delta T)\}_{k=0}^{\frac{1}{\Delta T}-1},\ \{x(k\Delta T)\}_{k=0}^{\frac{1}{\Delta T}}} \Delta T \sum_{k=0}^{\frac{1}{\Delta T}-1} L(\Delta T, x(\Delta T k), u(\Delta T k)) + \varphi(x(1))$$

$$\text{subject to } -x(\Delta T(k+1)) + x(\Delta T k) + \Delta T f(\Delta T, x(\Delta T k), u(\Delta T k)) = 0$$

$$g_i(x(\Delta T k)) \leq 0.$$

In this case, we treat the states and the inputs as independent variables that are only linked due to the equality constraints:

$$g(2) = g(x(2\Delta T k))$$

the gradient of $g(2)$ is treated as zero for the terms corresponding to $x(2\Delta T k)$

Note: 1) Larger # of decision variables
2) sparser set of constraints
3) Usually much better performance.

V. Walk through Project
show track, dynamics, and objective.

VI. Free Time Optimization.

What if we want to try to get the car to get to a place as quickly as possible?
OR optimize over different time interval $[a,b]$?

Theorem: Suppose $\dot{x}(t) = f(t, x(t), u(t))$ with $t \in [a,b]$ and $x(0) = x_0$.
Suppose we define:

$$\begin{bmatrix} \dot{\tilde{t}}(s) \\ \dot{\tilde{x}}(s) \end{bmatrix} = \begin{bmatrix} (b-a) \\ (b-a) f(\tilde{t}(s), \tilde{x}(s), \tilde{u}(s)) \end{bmatrix} \quad (\star)$$

for $s \in [0,1]$ where $\tilde{t}(0) = a$ and $\tilde{x}(0) = x_0$, with $\tilde{u}(s) = u(s(b-a)+a)$.

Called the time free transformation!

a) If $x : [a,b] \to \mathbb{R}^n$ is a solution to f, then

$$\begin{bmatrix} \tilde{t}(s) \\ \tilde{x}(s) \end{bmatrix} = \begin{bmatrix} a + s(b-a) \\ x(a + s(b-a)) \end{bmatrix}$$

is a solution to $(\star)$.

b) Similarly if $\hat{x} : [0,1] \to \mathbb{R}^n$ is a solution to $(\star)$ then $x(t) = \hat{x}((t-a)/(b-a))$ is a solution to $f(t, x(t), u(t))$.

Note:

1) By optimizing over this larger state space, we can introduce a component to the initial state corresponding to τ that we can leave as free and optimize:

$$\begin{bmatrix} \dot{\tau}(s) \\ \dot{x}(s) \\ \dot{b}(s) \end{bmatrix} = \begin{bmatrix} (b-a) \\ (b-a) f(\tau(s), x(s), u(s)) \\ 0 \end{bmatrix}$$

then:

$$\min_{\{u(\Delta T k)\}_{k=0}^{\frac{1}{\Delta T}-1}} \text{cost}$$

$$\left\{ \begin{bmatrix} \tau(\Delta T k) \\ x(\Delta T k) \\ b(\Delta T k) \end{bmatrix} \right\}_{k=0}^{\frac{1}{\Delta T}-1}$$

Subject to FWD Eules
State Constraints

$$\rightarrow b(0) \geq a$$

2) Be careful if you optimize these problems w/ no final cost, then obvious solution is to set $b(0) = a$.

VII. Putting Everything Together

1) Generate offline solution to design trajectory for dynamic scenarios using Trajectory Optimization

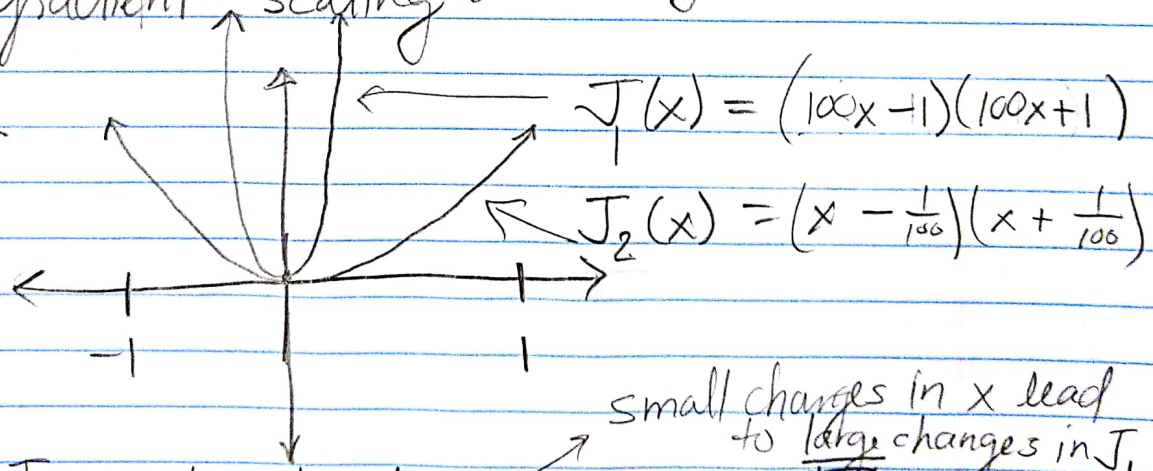
2) In real time use QP-MPC to follow trajectory as best as possible while avoiding obstacles that are only known at runtime.

Issues:

1) Nonlinear programming is prone to local minima.
   a) be cleves about initialization
   b) try different cost functions and constraints to avoid bad local minima.

2) Success of nonlinear programming relies on gradient scaling

Ex:



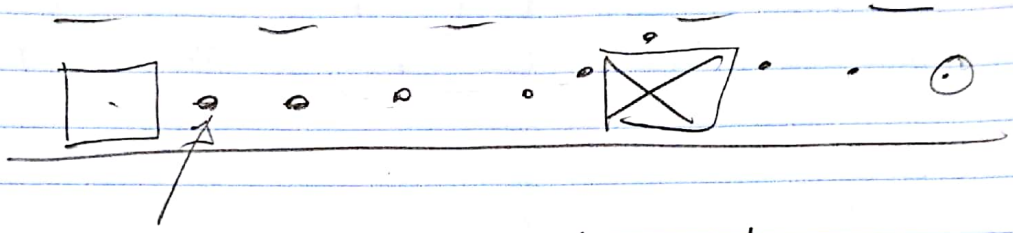
small changes in x lead to large changes in J_1

J_1 points sharply down compared to J_2 which points down less sharply

→ small changes in x lead to gradual changes in J_2

problems w/ cost functions like J_1 will be hard to solve.

3) Be very careful choosing discretization:



solver may generate solution like this, but that clips the obstacle! If you don't discretize finely enough or if the time horizon is too long or time free scaling is too large, then bad things may happen!