Last Time:

- Hybrid methods for contact

Today:

- Reasoning about friction
- Iterative Learning Control

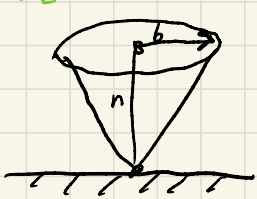
---

* Friction in Trajectory Optimization

- In our hybrid TrajOpt setup we assumed sticking / no slip
- Equivalent to infinite friction
- We can enforce additional friction cone constraints for each contact point to make sure the robot won't slip!

$$\|b\|_2 \leq \mu n$$

(μ: friction coefficient; n: normal force; b: friction force $\in \mathbb{R}^2$)



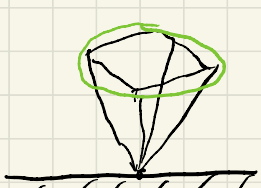
- This constraint is often linearized:

$$e^T d \leq \mu n, \quad d \in \mathbb{R}^4, \quad e = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$d \geq 0$$

$$b = \begin{bmatrix} \underset{2\times 2}{I} & \underset{2\times 2}{-I} \end{bmatrix} d$$

($e^T d = \|b\|_1$)



- The "friction pyramid" constraint is easy to enforce in QP-based MPC to avoid foot slip

- Hybrid methods generally try to avoid slip. If you want to model stick-slip behavior, need to add more modes.

- Friction is hard to model. Coulomb is only an approximation. Typically use a conservatively small value for μ.

---

* What Happens when our model has errors?

  - Models are always approximations

  - Simpler models are often preferred even if they're less accurate.

  - Feedback (e.g. LQR or MPC) can often compensate for model error

  - Sometimes that isn't good enough (e.g. very tight constraints, performance requirements).

  - Several Options:

    1) Parameter Estimation: Classical "grey-box" modeling / system ID. Fit e.g. mass in your model using data from the real system:

       + Very sample efficient
       + Generalizes
       - Assumes model structure

2) Learn Model: Fit a generic function approximator to full dynamics or residual. Classical "black-box" system ID.

+ Doesn't assume model structure
+ Generalizes
- Not sample efficient. Requires lots of data.

→ Improve the model

3) Learn a Policy: Standard reinforcement learning approach: Optimize a function approximator for feedback policy.

+ Makes few assumptions
- Doesn't generalize
- Not sample efficient. Requires lots of "rollouts"

4) Improve a trajectory: Assume we have a reference trajectory computed with a nominal model. Improve it with data from the real system.

+ Makes few assumptions
- Assumes a decent model
- Doesn't generalize
+ Very sample efficient

Improve the policy

* Iterative Learning Control

- Can think of this as a very specialized policy gradient method for the policy class:

$$u_n(\bar{u}_n) = \bar{u}_n - K_n(x_n - \bar{x}_n)$$

$\bar{u}_n$: reference input trajectory

$K_n(x_n - \bar{x}_n)$: can be any tracking controller

where we are improving the reference $\bar{u}$

- Can think of this as SQP method where we get the "right-hand side" vector from a rollout on the real system.

- Assume we have a reference trajectory $\bar{x}, \bar{u}$ that we want to track:

$$\min_{x_{1:N},\, u_{1:N-1}} \; J = \sum_{n=1}^{N-1} \tfrac{1}{2}(x_n - \bar{x}_n)^T Q (x_n - \bar{x}_n) + \tfrac{1}{2}(u_n - \bar{u}_n)^T R (u_n - \bar{u}_n)$$

(can be any cost function) $+ \tfrac{1}{2}(x_N - \bar{x}_N)^T Q_N (x_N - \bar{x}_N)$

$$\text{s.t.} \quad x_{n+1} = f(x_n, u_n)$$

- As we've seen before, the KKT system for this problem looks like:

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \lambda \end{bmatrix} = \begin{bmatrix} -\nabla J \\ -c(z) \end{bmatrix}$$

where $z = \begin{bmatrix} x_1 \\ u_1 \\ \vdots \\ x_N \end{bmatrix}$, $c(z) = \begin{bmatrix} \vdots \\ f(x_n, u_n) - x_{n+1} \\ \vdots \end{bmatrix}$, $C = \dfrac{\partial c}{\partial z}$

$$H = \begin{bmatrix} Q_R & & & 0 \\ & Q & & \\ & & \ddots & \\ 0 & & & Q_N \end{bmatrix}$$ ← Gauss-Newton Hessian

- Note that we have ∇J in the RHS instead of ∇L and λ instead of $\Delta\lambda$ in the LHS. We can do this because the Lagrangian is linear in λ. We normally prefer solving for $\Delta\lambda$ because that lets us regularize.

- Two important observations:

  1) If we do a rollout on the real system, $c(z) = 0$ always (for the true dynamics)

  2) Since we know J, given x_n, u_n from a rollout, we can compute ∇J

- Now we have the RHS vector for the KKT system. We also knew it from the cost function.

- We can compute $C = \frac{\partial c}{\partial z}$ using x_n, u_n and the nominal model.

- However, since our nominal model is approximate and assuming x_n, u_n is already close to $\bar{x}, \bar{u}$, we can just use $C = \frac{\partial c}{\partial z}\Big|_{\bar{x},\bar{u}}$ which can be computed once offline.

- Now just solve the KKT system for Δz, update $\bar{u} \leftarrow \bar{u} + \Delta u$, and repeat.

- Can easily add inequality constraints (e.g. torque limits) and solve a QP.