DDP

Thursday, August 7, 2025 5:56 PM

Differential Dynamic Programming

Core Idea

- Linearizes the dynamics of the system & quadratizes the cost function around some nominal trajectory, then solves the resulting Linear Quadratic Regulator (LQR) Problem.
- This is repeated to handle non-linearity

Simple Example (1D)

Step 0: Setup

System: A point w/ position x & velocity v

- state: $s = [x, v]^T$
- control: u (acceleration)
- dynamics: $\dot{x} = v$, $\dot{v} = u$

.1 sec increments

  - $x[k+1] = x[k] + (.1)(v[k])$
  - $v[k+1] = v[k] + (.1)(u[k])$

- Goal: Move from $(x_0, v_0) = (0,0)$ to $(x_f, v_f) = (1,0)$ in 3 timesteps while minimizing control effort
- Cost function:

$$J = \underbrace{(x_3 - 1)^2}_{\text{Final Position Cost}} + \underbrace{v_3^2}_{\text{Final Velocity Cost}} + \underbrace{.1(u_0^2 + u_1^2 + u_2^2)}_{\text{Acceleration Penalty}}$$

Final b/c 3 timesteps

Step 1: Initialize

Initial Guess

$u = [10, 0, -10]$

Step 2: Forward Simulation

k (time step of .1 sec)

s (state vector) = $[\text{position}, \text{velocity}]^T$

i	k_i	u_i	s_i
0	0	10	[0,0]
1	1	0	[0,1]
2	2	-10	[0.1, 1]
3	3	–	[0.2, 0]

Step 3: Backward Pass

$$J = (x_3 - 1)^2 + v_3^2 + .1(u_0^2 + u_1^2 + u_2^2)$$

$k=3$:

$S_3 = [.2, 0]$

$u_3 = 0$

$V_3 = (.2-1)^2 + 0^2 = .64$ ← Value Function

$V_{3x} = \frac{\partial V_3}{\partial x_3} = 2(x_3 - 1) = 2(.2-1) = -1.6$

$V_{3v} = \frac{\partial V_3}{\partial v_3} = 2v_3 = 2(0) = 0$

Gradients

$V_{3xx} = \frac{\partial^2 V_3}{\partial x_3^2} = 2$ $\quad V_{3xv} = \frac{\partial^2 V_3}{\partial x_3 \partial v_3} = 0$

$V_{3vv} = \frac{\partial^2 V_3}{\partial v_3^2} = 2$

Hessian

So,

$V_x = [-1.6, 0]^T \quad V_{xx} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

Q-function & Dynamics Linearization ??

- Need to review those concepts
- LQR too