Week 5

Inverse-Dynamics Tutorial

Kangmin Lee

This tutorial covers inverse-dynamics from Operational Space Control to QP, HQP, and TSID focusing on the math, intuition, and practical design patterns behind them. Through code and simulation, you'll learn to formulate tasks and constraints and compute constraint-satisfying torques suitable for robots.

파이썬 코드 실행 방법

1. 터미널 실행

Ubuntu 에서 터미널을 엽니다

2. 저장소 폴더로 이동

```
cd ~/path/to/demo_code.py
```

3. 실습 코드 실행

```
python3 demo_code.py
```

파이썬 코드 실행 방법

4. MP4 영상 저장을 위한 ffmpeg 설치 (선택 사항)

코드를 실행했을 때 예를 들어 다음과 같은 출력이 보일 수 있습니다.

```
Saved GIF: outputs/tsid_qp_control.gif
MP4 save failed (need ffmpeg?): [WinError 2] 지정된 파일을 찾을 수 없습니다
```

이는 **GIF 파일은 저장되었지만, MP4 파일을 저장하려면 ffmpeg가 필요하다**는 의미입니다.

Ubuntu에서 ffmpeg를 설치하려면 다음 명령을 실행합니다.

```
sudo apt update
sudo apt install ffmpeg
```

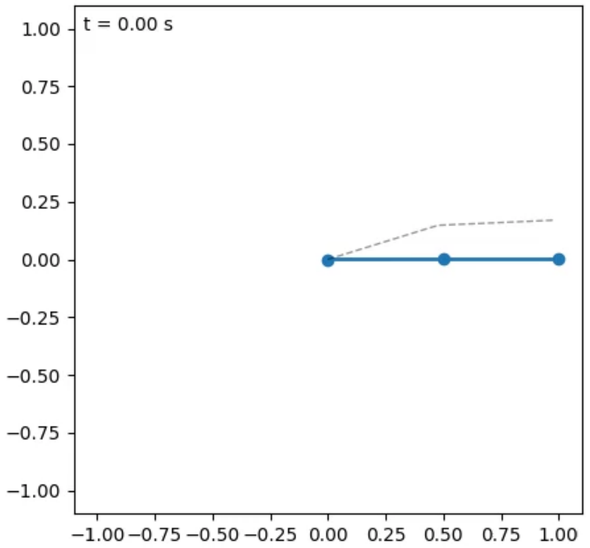
설치 후 다시 코드를 실행합니다.

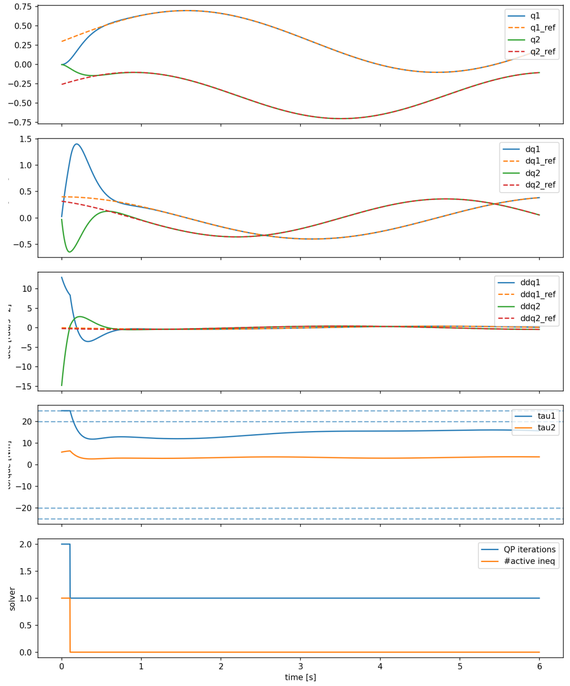
```
python3 demo_code.py
```

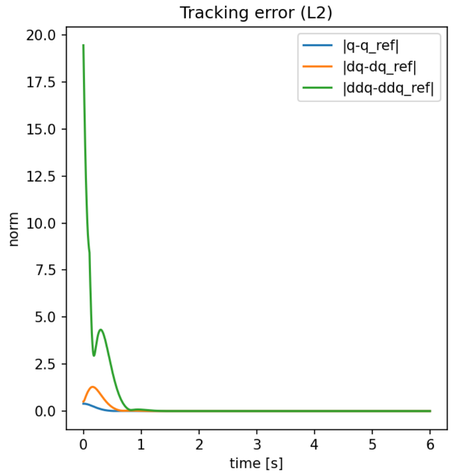
파이썬 코드 실행 결과

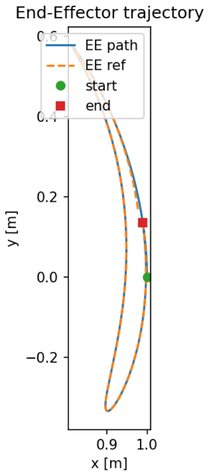
코드 실행이 끝나면 다음과 같은 파일들이 outputs/ 폴더에 생성됩니다.

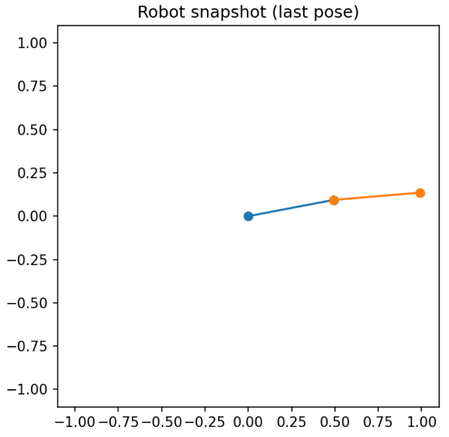
- **tsid_qp_control.gif :** 2-자유도 로봇팔이 궤적을 추종하는 애니메이션
- **tsid_qp_demo_plots.png :** 조인트 q, dq, ddq, 토크, QP 솔버 상태 등의 시간 그래프
- **tsid_qp_demo_plots_explain.png :** 트래킹 에러, EE 궤적, 최종 포즈 등을 설명하는 플롯











코드 내용

이 예제는 **2-자유도 평면 로봇팔**에 대해

- **TSID(Task-Space / Torque-Space Inverse Dynamics) 스타일의 QP 제어**를 적용하고
- **토크 한계**를 만족시키면서
- **원하는 관절 가속도**를 최대한 따라가도록 하는 풀-파이프라인을
- 하나의 파이썬 파일로 구현한 데모입니다

`ActiveSetQP`라는 **아주 작은 교육용 Active-Set QP 솔버**를 파일 안에 직접 구현하여, 외부 QP 라이브러리 없이도 **TSID + QP의 전체 흐름**을 실험해볼 수 있습니다.

시스템 개요

- 로봇: 2-자유도 평면 매니퓰레이터 (기저 고정, 중력 및 관성 포함)

- 상태:
  - 관절 위치: q
  - 관절 속도: $\dot{q}$
  - 관절 가속도: $\ddot{q}$

- 입력(제어 변수):
  - 관절 토크: τ

- 동역학 식:

$$M(q)\,\ddot{q} + h(q,\dot{q}) = \tau$$

시스템 개요

1. Reference 궤적 생성 $(q_{ref}(t), \dot{q}_{ref}(t), \ddot{q}_{ref}(t))$

2. TSID 형식으로 원하는 가속도 계산 $\ddot{q}_{des}(t) = K_p(q_{ref} - q) + K_d(\dot{q}_{ref} - \dot{q}) + \ddot{q}_{ref}$

3. QP로 $\ddot{q}$, τ 결정

- 목적 : $\ddot{q} \approx \ddot{q}_{\mathrm{des}}$
- 제약 : $M(q)\,\ddot{q} + h(q,\dot{q}) - \tau = 0 \qquad \tau_{\min} \le \tau \le \tau_{\max}$

4. 컨트롤러 출력 : Forward dynamics

- QP 해에서 τ 부분만 뽑아서 **제어 입력**으로 사용

$$M(q)\,\ddot{q}_{plant} + h(q,\dot{q}) = \tau$$

$$\ddot{q}_{\mathrm{plant}} = M(q)^{-1}(\tau - h(q,\dot{q})$$

이 전체 흐름을 코드에서는 run_sim() 안의 루프에서 한 타임스텝마다 수행한다.

코드 뜯어보기

main 함수 → run_sim() 실행

```
513  if __name__ == "__main__":
514      run_sim(T=6.0, dt=0.002, seed=0, plot=True, animate=True)
515      print("Saved:", os.path.join(OUTDIR, 'tsid_qp_demo_plots.png'))
516      print("Saved:", os.path.join(OUTDIR, 'tsid_qp_demo_plots_explain.png'))
517      print("Saved: outputs/tsid_qp_control.gif (and .mp4 if ffmpeg is available)")
```

run_sim() 함수

- 로봇 정의 및 초기화 :

```
325      robot = Planar2Link()
326
327      # 초기 상태
328      q  = np.array([0.0, 0.0])
329      dq = np.array([0.0, 0.0])
330
331      # 토크 한계
332      tau_min = np.array([-25.0, -20.0])
333      tau_max = np.array([25.0,  20.0])
```

코드 뜯어보기

Class Planar2Link : 2DOF 링크 로봇 구현

```
132  # ---------- 2-DoF planar arm ----------
133  class Planar2Link:
134      def __init__(self):
135          # link length
136          self.l1 = 0.5
137          self.l2 = 0.5
138          # masses
139          self.m1 = 2.0
140          self.m2 = 1.5
141          # COM positions (from joint)
142          self.c1 = self.l1 * 0.5
143          self.c2 = self.l2 * 0.5
144          # inertias
145          self.I1 = 0.06
146          self.I2 = 0.04
147          # gravity
148          self.g = 9.81
```

run_sim() 안의 for 루프(핵심 제어 흐름)

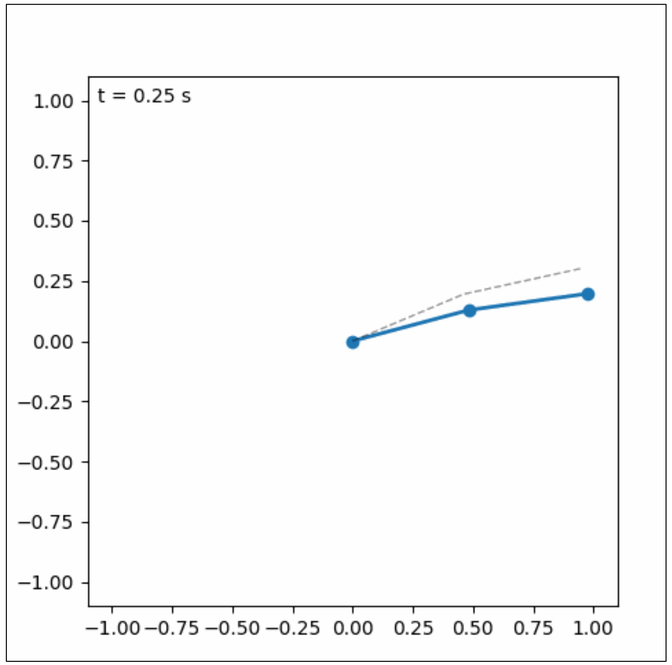
```
353      for k in range(steps):
```

코드 뜯어보기 - run_sim()

Reference 궤적 생성 $(q_{\mathrm{ref}}(t),\ \dot{q}_{\mathrm{ref}}(t),\ \ddot{q}_{\mathrm{ref}}(t)) = \mathtt{posture\_reference}(t)$

```
356         # 1) reference trajectory
357         qref, dqref, ddqref = posture_reference(t, q0=np.array([0.3, -0.4]))
```

```
238 # ---------- reference & TSID desired ddq ----------
239 def posture_reference(t, q0):
240     """시간에 따라 변하는 간단한 관절 참조 궤적."""
241     amp   = np.array([0.4, 0.3])
242     omega = np.array([1.0, 1.2])
243     phase = np.array([0.0, 0.5])
244 
245     qref   = q0 + amp * np.sin(omega*t + phase)
246     dqref  = (omega * amp) * np.cos(omega*t + phase)
247     ddqref = - (omega**2) * amp * np.sin(omega*t + phase)
248 
249     return qref, dqref, ddqref
```



- Sin 함수 형태로 시간에 따라 각도가 변하도록 설계

코드 뜯어보기 - run_sim()

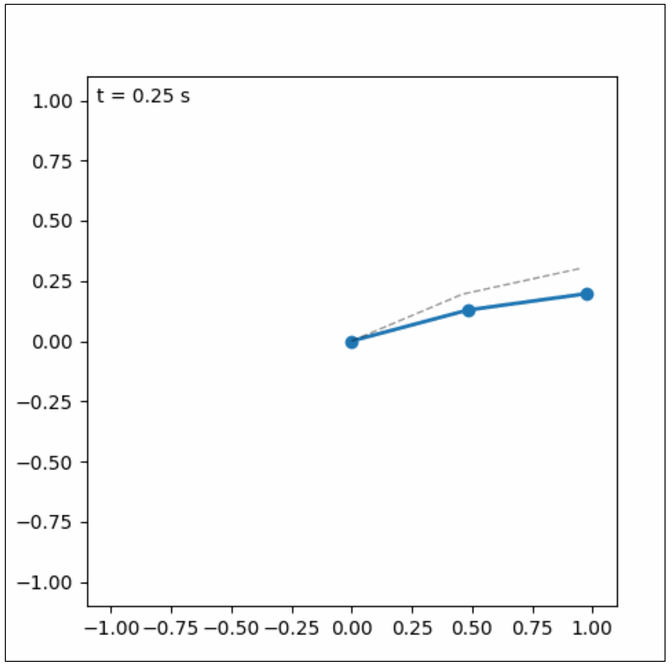
TSID 형식의 오차 및 가속도 정의

```
359         # 2) TSID 스타일 원하는 가속도
360         ddq_des = tsid_ddq_des(q, dq, qref, dqref, ddqref)
```

```
252     def tsid_ddq_des(q, dq, qref, dqref, ddqref, kp=60.0, kd=12.0):
253         """TSID 스타일의 원하는 관절 가속도 ddq_des 계산."""
254         e  = qref - q
255         ed = dqref - dq
256         return kp*e + kd*ed + ddqref
```

$$e(t) = q_{\mathrm{ref}}(t) - q(t), \qquad \dot{e}(t) = \dot{q}_{\mathrm{ref}}(t) - \dot{q}(t)$$

$$\ddot{q}_{\mathrm{des}}(t) = K_p\, e(t) + K_d\, \dot{e}(t) + \ddot{q}_{\mathrm{ref}}(t)$$



코드 뜯어보기 - run_sim()

QP형태에 맞는 꼴 맞추기

```
362         # 3) QP 구성 및 해 (컨트롤러: ddq, tau 동시 최적화)
363         H, g, Aeq, beq, Ain, binv = build_qp(
364             robot, q, dq, ddq_des, tau_min, tau_max, w_ddq=1.0
365         )
```

```
195 def build_qp(robot, q, dq, ddq_des, tau_min, tau_max, w_ddq=1.0):
196     """
197     Build TSID-style QP:
198 
199         min_{ddq, tau}  0.5 * w_ddq * ||ddq - ddq_des||^2
200 
201         s.t. M(q) ddq + h(q,dq) - tau = 0      (equality)
202              tau_min <= tau <= tau_max         (inequalities)
203     """
```

우리가 풀고자 하는 문제 형태

$$\begin{aligned} \min_{\ddot{q},\,\tau} \quad & \frac{1}{2}\, w_{\ddot{q}} \left\| \ddot{q} - \ddot{q}_{\mathrm{des}} \right\|^2 \\ \text{s.t.} \quad & M(q)\,\ddot{q} + h(q,\dot{q}) - \tau = 0, \\ & \tau_{\min} \le \tau \le \tau_{\max}. \end{aligned}$$

QP가 요구하는 형식

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}\, x^{\top} H x + g^{\top} x \\ \text{s.t.} \quad & A_{\mathrm{eq}} x = b_{\mathrm{eq}}, \\ & A_{\mathrm{in}} x \le b_{\mathrm{in}}. \end{aligned}$$

코드 뜯어보기 - run_sim()

형식을 맞추기 위해 단순히 전개한 후 개선을 위해 행렬 형태로 정리

$$\min_x \quad \frac{1}{2} x^\top H x + g^\top x$$

$$x = \begin{bmatrix} \ddot{q} \\ \tau \end{bmatrix} = \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \tau_1 \\ \tau_2 \end{bmatrix}$$

$$\frac{1}{2} w_{\ddot{q}} (\ddot{q} - \ddot{q}_{\text{des}})^\top (\ddot{q} - \ddot{q}_{\text{des}}) = \frac{1}{2} w_{\ddot{q}} \left(\ddot{q}^\top \ddot{q} - 2\, \ddot{q}_{\text{des}}^\top \ddot{q} + \ddot{q}_{\text{des}}^\top \ddot{q}_{\text{des}} \right)$$

$$H = \begin{bmatrix} w_{\ddot{q}} & 0 & 0 & 0 \\ 0 & w_{\ddot{q}} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} w_{\ddot{q}} I_2 & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} \end{bmatrix}, \qquad g = \begin{bmatrix} -w_{\ddot{q}}\, \ddot{q}_{\text{des}} \\ 0 \end{bmatrix} = \begin{bmatrix} -w_{\ddot{q}}\, \ddot{q}_{\text{des},1} \\ -w_{\ddot{q}}\, \ddot{q}_{\text{des},2} \\ 0 \\ 0 \end{bmatrix}$$

```
211     # Cost: 0.5 * ||ddq - ddq_des||^2
212     H = np.zeros((n, n))
213     g = np.zeros(n)
214     H[0:n_ddq, 0:n_ddq] = w_ddq * np.eye(n_ddq)
215     g[0:n_ddq] = -w_ddq * ddq_des
```

코드 뜯어보기 - run_sim()

형식을 맞추기 위해 단순히 전개한 후 개선을 위해 행렬 형태로 정리

$$\text{s.t.} \quad A_{\text{eq}}x = b_{\text{eq}}, \qquad A_{\text{in}}x \le b_{\text{in}}.$$

$$M(q)\,\ddot{q} + h(q,\dot{q}) = \tau \qquad M(q) = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \quad h(q,\dot{q}) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}, \quad \tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

전개 : $M_{11}\,\ddot{q}_1 + M_{12}\,\ddot{q}_2 + h_1 = \tau_1, \qquad M_{21}\,\ddot{q}_1 + M_{22}\,\ddot{q}_2 + h_2 = \tau_2$

변형 : $M_{11}\,\ddot{q}_1 + M_{12}\,\ddot{q}_2 - \tau_1 = -h_1, \qquad M_{21}\,\ddot{q}_1 + M_{22}\,\ddot{q}_2 - \tau_2 = -h_2$

정리 :

$$A_{\text{eq}}\,x = b_{\text{eq}} \iff \begin{cases} M_{11}\,\ddot{q}_1 + M_{12}\,\ddot{q}_2 - \tau_1 = -h_1, \\ M_{21}\,\ddot{q}_1 + M_{22}\,\ddot{q}_2 - \tau_2 = -h_2. \end{cases}$$

$$x = \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \tau_1 \\ \tau_2 \end{bmatrix}, \quad A_{\text{eq}} = \begin{bmatrix} M_{11} & M_{12} & -1 & 0 \\ M_{21} & M_{22} & 0 & -1 \end{bmatrix}, \quad b_{\text{eq}} = \begin{bmatrix} -h_1 \\ -h_2 \end{bmatrix}$$

코드 뜯어보기 - run_sim()

형식을 맞추기 위해 단순히 전개한 후 개선을 위해 행렬 형태로 정리

$$\text{s.t.} \quad A_{\text{eq}}x = b_{\text{eq}}, \quad A_{\text{in}}x \le b_{\text{in}}.$$

토크 한계 : $A_{\text{in}}\,x \le b_{\text{in}} \iff \tau_{\min} \le \tau \le \tau_{\max}, \quad \tau = \begin{bmatrix}\tau_1 \\ \tau_2\end{bmatrix},\ \tau_{\min} = \begin{bmatrix}\tau_{1,\min} \\ \tau_{2,\min}\end{bmatrix},\ \tau_{\max} = \begin{bmatrix}\tau_{1,\max} \\ \tau_{2,\max}\end{bmatrix}$

변형 : $x = \begin{bmatrix}\ddot{q}_1 \\ \ddot{q}_2 \\ \tau_1 \\ \tau_2\end{bmatrix}$

$$\tau_1 \le \tau_{1,\max} \iff 0\cdot\ddot{q}_1 + 0\cdot\ddot{q}_2 + 1\cdot\tau_1 + 0\cdot\tau_2 \le \tau_{1,\max} \implies \begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}$$

$$\tau_2 \le \tau_{2,\max} \iff 0\cdot\ddot{q}_1 + 0\cdot\ddot{q}_2 + 0\cdot\tau_1 + 1\cdot\tau_2 \le \tau_{2,\max} \implies \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}$$

$$-\tau_1 \le -\tau_{1,\min} \iff 0\cdot\ddot{q}_1 + 0\cdot\ddot{q}_2 - 1\cdot\tau_1 + 0\cdot\tau_2 \le -\tau_{1,\min} \implies \begin{bmatrix}0 & 0 & -1 & 0\end{bmatrix}$$

$$-\tau_2 \le -\tau_{2,\min} \iff 0\cdot\ddot{q}_1 + 0\cdot\ddot{q}_2 + 0\cdot\tau_1 - 1\cdot\tau_2 \le -\tau_{2,\min} \implies \begin{bmatrix}0 & 0 & 0 & -1\end{bmatrix}$$

정리 :

$$A_{\text{in}} = \begin{bmatrix}0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1\end{bmatrix}, \qquad b_{\text{in}} = \begin{bmatrix}\tau_{1,\max} \\ \tau_{2,\max} \\ -\tau_{1,\min} \\ -\tau_{2,\min}\end{bmatrix}$$

코드 뜯어보기 - run_sim()

계산을 통해 x 구하기

```
366         solver = ActiveSetQP(H, g, Aeq, beq, Ain, binv, tol=1e-8, max_iter=100)
367         x, info = solver.solve()
```

- KKT 조건이 QP 문제의 최적해에 대한 필요충분 조건 (모든 QP문제에 해당되지는 않음, 볼록 QP일 경우만)
- solve() 함수는 결국 KKT 4조건(정지 + primal + dual + complementarity) 을 만족하는 (x, λ)를 찾으려고 도는 루프

- **“KKT 조건을 만족하는 (x, λ)”를 찾는 알고리즘**

코드 뜯어보기 - run_sim()

계산을 통해 x 구하기

- KKT 4가지 조건 $g(x) := A_{\text{in}}x - b_{\text{in}} \leq 0$

Stationarity: $\nabla_x L(x^\star, \lambda_{\text{eq}}^\star, \lambda_{\text{in}}^\star) = 0$

$$L(x, \lambda_{\text{eq}}, \lambda_{\text{in}}) = f(x) + \lambda_{\text{eq}}^\top \big(A_{\text{eq}}x - b_{\text{eq}}\big) + \lambda_{\text{in}}^\top \big(A_{\text{in}}x - b_{\text{in}}\big)$$

$$\nabla_x L(x, \lambda_{\text{eq}}, \lambda_{\text{in}}) = Hx + g + A_{\text{eq}}^\top \lambda_{\text{eq}} + A_{\text{in}}^\top \lambda_{\text{in}}$$

$$Hx^\star + g + A_{\text{eq}}^\top \lambda_{\text{eq}}^\star + A_{\text{in}}^\top \lambda_{\text{in}}^\star = 0$$

코드 뜯어보기 - run_sim()

계산을 통해 x 구하기

- KKT 4가지 조건 $g(x) := A_{\text{in}}x - b_{\text{in}} \leq 0$

Stationarity: $\nabla_x L(x^\star, \lambda_{\text{eq}}^\star, \lambda_{\text{in}}^\star) = 0$

Primal feasibility: $A_{\text{eq}}x^\star = b_{\text{eq}}, \qquad A_{\text{in}}x^\star \leq b_{\text{in}}$

Dual feasibility: $\lambda_{\text{in}}^\star \geq 0$

Complementary slackness: $\lambda_{\text{in},i}^\star \left(A_{\text{in},i}x^\star - b_{\text{in},i}\right) = 0, \quad \forall i$

코드 뜯어보기

이렇게 최종 구하고자 하는 토크를 구했다

1. 실제 로봇

- 제어기가 모터 드라이버에 토크 명령 τ을 보낸다.
- 로봇 본체(질량, 관성, 링크, 중력, 마찰, 접촉 등)가 물리 법칙에 따라 자동으로 $\ddot{q}$, $\dot{q}$, q 를 만들어 낸다

$$M(q)\,\ddot{q} + h(q,\dot{q}) = \tau$$

- **이 식을 직접 계산하는 건 컴퓨터가 아니라 “세상(물리)”이다**

코드 뜯어보기

이렇게 최종 구하고자 하는 토크를 구했다

2. MuJoCo 시뮬레이터

- 컨트롤러가 τ를 계산해서 data->ctrl 등에 넣어준다
- MuJoCo는 내부적으로 관성행렬 M(q), 비선형항 h(q,q˙) 계산
- 알고리즘(ABA 등)을 통해 $\ddot{q} = M(q)^{-1}\left(\tau - h(q,\dot{q})\right)$ 와 동등한 Forward Dynamics 계산

- **사용자는 보통 mj_step(model, data) 같은 함수만 호출하고,forward dynamics 수식을 직접 쓰지 않는다**

코드 뜯어보기

이렇게 최종 구하고자 하는 토크를 구했다

3. 예제 파이썬 코드

- $\ddot{q} = M(q)^{-1}\left(\tau - h(q, \dot{q})\right)$ 이를 코드로 구현하고 **적분하여 로봇의 상태를 업데이트**
- 이 결과를 **2D 그림 + 애니메이션**으로 시각화한다

```
372         # 4) Plant (forward dynamics): tau -> ddq
373         #    M(q) ddq + h(q,dq) = tau  를 풀어서 ddq 계산
374         M = robot.M(q)
375         h_vec = robot.h(q, dq)
376         ddq = np.linalg.solve(M, tau - h_vec)
377
378         # 여기서는 "플랜트는 tau를 입력받아 forward dynamics를 돈다"
379         # ddq (forward dynamics 결과)를 적분에 사용
380
381         # 5) 상태 적분 (semi-implicit Euler)
382         dq = dq + dt * ddq
383         q  = q  + dt * dq
```

Thank you