Week 4

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.

QP 예제 1 (1차원 문제)

- 문제 : 원하는 토크 τ_{PD} 에 가깝게 가되, 모터 한계 $\tau_{\min} \le \tau \le \tau_{\max}$ 를 지켜야 함
- 정식화 : $\min_\tau \ \frac{1}{2}(\tau - \tau_{PD})^2 \quad \text{s.t.} \quad \tau_{\min} \le \tau \le \tau_{\max}$
- Case : $\tau_{PD} = 1.8, \qquad \tau_{\min} = -2, \qquad \tau_{\max} = 1$
- 해법 : $\frac{d}{d\tau}\left(\frac{1}{2}(\tau - \tau_{PD})^2\right) = 0 \ \Rightarrow \ \tau^\star = \tau_{PD} = 1.8$
- 제약 검토 : $\tau^\star = 1.8 > \tau_{\max} = 1, \ \text{activate the upper bound} \ \Rightarrow \ \tau = \tau_{\max} = 1$
- 결과(최적해) : $\boxed{\tau^\star = 1}$

QP 예제 2 (2차원 문제)

- 2차원 QP를 풀기 위해서는 2가지 개념을 더 알아야 한다
- 해당 강의에서는 두 개념에 대한 설명을 진행하며 자세한 증명은 생략

라그랑주 승수법과 KKT 조건

- **라그랑주 승수법 :** 제약이 있는 최소화 문제를 제약이 없는 최소화 문제로 바꿀 수 있다
- **KKT(Karush–Kuhn–Tucker)** 조건은 "제약이 있는 최적화"에서 최적해가 만족해야 하는 필요조건

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g_i(x) = 0, \quad i = 1, 2, \ldots, m \end{aligned}$$

$$\mathcal{L}(x, \lambda) \;=\; f(x) \;+\; \sum_{i=1}^{m} \lambda_i \, g_i(x) \;=\; f(x) + \lambda^{\top} g(x)$$

제약이 없는 하나의 식으로 변형 가능

QP 예제 2 (2차원 문제)

- 함수 $f(x)$가 x^* 에서 로컬 최소값이 되기 위한 필요조건은 $x = x^*$ 와 $\lambda = \lambda^*$ 에서 L 의 gradient가 0이 되는 것

KKT 정지조건 $\nabla_x L(x^*, \lambda^*) = 0$

- 문제 정식화 : $\min_y \ \frac{1}{2}\|y - c\|^2 \quad \text{s.t.} \quad a^\top y \le b$ $\qquad c = \begin{bmatrix} 2 \\ 1 \end{bmatrix},\ a = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ b = 2$

- 단계 A : 무제약 해 $f(y) = \frac{1}{2}\|y - c\|^2 = \frac{1}{2}(y - c)^\top (y - c)$

$$\nabla f(y) = y - c \qquad \nabla f(y) = 0 \implies y_0 = c = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

- 단계 B : 제약 검토 $a^\top c = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 3 \ > \ b(= 2)$

무제약 해는 제약을 위반함

QP 예제 2 (2차원 문제)

- 함수 $f(x)$가 x^* 에서 로컬 최소값이 되기 위한 필요조건은 $x = x^*$ 와 $\lambda = \lambda^*$ 에서 L 의 gradient가 0이 되는 것

KKT 정지조건 $$\nabla_x L(x^*, \lambda^*) = 0$$

- 문제 정식화 : $\min_y \ \frac{1}{2}\|y - c\|^2 \quad \text{s.t.} \quad a^\top y \le b$ $c = \begin{bmatrix} 2 \\ 1 \end{bmatrix},\ a = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ b = 2$

- 단계 C : 따라서 제약이 활성(active)되어, 최적해는 경계 $a^\top y = b$ 위에 존재

$$\mathcal{L}(y, \lambda) = \tfrac{1}{2}\|y - c\|^2 + \lambda\,(a^\top y - b), \qquad \lambda \ge 0$$

KKT 정지조건 $$\nabla_y \mathcal{L}(y, \lambda) = (y - c) + \lambda a = 0$$

$$y = c - \lambda a$$

QP 예제 2 (2차원 문제)

- 함수 $f(x)$가 x^* 에서 로컬 최소값이 되기 위한 필요조건은 $x = x^*$ 와 $\lambda = \lambda^*$ 에서 L 의 gradient가 0이 되는 것

KKT 정지조건 $\nabla_x L(x^*, \lambda^*) = 0$

- 문제 정식화 : $\min_y \ \frac{1}{2}\|y - c\|^2 \quad \text{s.t.} \quad a^\top y \le b$ $\qquad c = \begin{bmatrix} 2 \\ 1 \end{bmatrix},\ a = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ b = 2$
- 활성 제약에 대입 : $y = c - \lambda a$ $\qquad a^\top y = b$

$$a^\top y = b \implies a^\top (c - \lambda a) = b \implies \lambda^\star = \frac{a^\top c - b}{\|a\|^2} = \frac{3-2}{1^2+1^2} = \frac{1}{2} = 0.5$$

- 결과(최적해) :

$$y^\star = c - \lambda^\star a = \begin{bmatrix} 2 \\ 1 \end{bmatrix} - 0.5 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \boxed{\begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix}}$$

QP Solver

- 동일하게 **KKT 조건을 만족하는 최적해**를 찾는다
- 손으로 직접 계산할 수 있지만, 실제 문제는 변수·제약이 많고 반복적이므로 **컴퓨터가 자동으로** 그 과정을 수행
- 솔버마다 **계산 방법(알고리즘)이 다를 뿐 결과적으로 노리는 해는 동일**
- 종료 판단은 KKT 잔차(정지성·타당성·상보성)가 충분히 작아졌는지로 한다
- 수치 안정성을 위해 스케일링, 정규화, 사전조건화, 피벗팅 등을 쓴다
- **성능을 좌우하는 요소 :** 문제 크기와 희소성, 제약 구조, 실시간 제약, 하드웨어(CPU/GPU)

결론 : QP 솔버들은 접근법만 다를 뿐, 모두 KKT 조건을 만족하는 지점을 찾도록 설계되어 있다

> 문제 성격(크기·희소성·실시간성)에 맞는 알고리즘을 골라 쓰면 된다

Multi-Objective Optimization

- 복잡한 로봇은 주과제(main task) 에 비해 자유도가 더 많아 여자유도를 갖는다

  예) 7-DoF 매니퓰레이터가 EE의 6DoF 포즈를 제어 → 여분 1 DoF

  예) 18-DoF 이족보행 로봇이 양발의 12DoF 포즈를 제어 → 여분 6 DoF

- 남는 자유도는 **부과제(secondary tasks)**를 수행하는 데 활용할 수 있다(자세·관절 여유도 관리, 에너지 최소화, 접촉력 정규화 등)

Multi-Objective Optimization

가중합(Weighted Sum)

- 여러 과제를 동시에 다루는 가장 단순한 방법은 **가중합(Weighted Sum)**
- 각 과제를 작업 함수 $g_i(y)$로 두고, 가중치 w_i 를 곱해 합을 최소화
- 장점: 표준 QP 꼴을 유지해 계산이 빠르고 안정적
- 단점: 가중치 선정이 어렵고, 너무 크거나 작은 가중치는 수치 악화·스케일링 문제를 유발

$$g_i(y) \;=\; \left\| A_i\, y - a_i \right\|^2, \quad i = 1, \ldots, N$$

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & \sum_{i=1}^{N} w_i\, g_i(y) \\ \text{subject to} \quad & \begin{bmatrix} J(q) & 0 & 0 \\ M(q) & -J(q)^\top & -S^\top \end{bmatrix} \begin{bmatrix} \ddot{q} \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -\dot{J}(q,\dot{q})\,\dot{q} \\ -\,h(q,\dot{q}) \end{bmatrix}. \end{aligned}$$

HQP (Hierarchical QP)

- 또 다른 방법은 여러 작업 $g_i(y)$ 가 있을 때 **우선순위(priority)**를 정해 해결하는 것 = HPQ

  "작업 1 ≫ 작업 2 ≫ … ≫ 작업 N"처럼 **상위 과제가 하위 과제보다 무한히 중요**하다고 본다

원리

- 구현은 연속(cascade) QP: i=1 부터 N까지 순서대로 작은 QP를 풀며,
- 동역학/접촉 등 공통 평등제약은 매 단계에 모두 유지하고,
- 이전 단계 $j<i$에서 얻은 최적값 g_j 를 그대로 유지하도록 **제약에 추가**

장점 : 가중치 찾기보다 우선순위 정하기가 쉽다

단점 : 여러 번 QP를 풀어야 해서 계산량 증가, 상위 과제 결과가 너무 제한적이면 하위 과제 실행 불가

HQP (Hierarchical QP)

- 또 다른 방법은 여러 작업 $g_i(y)$ 가 있을 때 **우선순위(priority)**를 정해 해결하는 것 = HPQ

  "작업 1 ≫ 작업 2 ≫ … ≫ 작업 N"처럼 **상위 과제가 하위 과제보다 무한히 중요**하다고 본다

- 이전 단계 $j<i$에서 얻은 최적값 g_j 를 그대로 유지하도록 **제약에 추가**

$$\forall i = 1, \ldots, N: \qquad \underset{y}{\text{minimize}} \quad g_i(y)$$

$$\text{subject to} \quad \begin{bmatrix} J & 0 & 0 \\ M & -J^\top & -S^\top \end{bmatrix} \begin{bmatrix} \ddot{q} \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -\dot{J}\,\dot{q} \\ -h \end{bmatrix},$$

$$g_j(y) = g_j^\star, \qquad \forall j < i.$$

Thank you