Task-Space Inverse Dynamics

Optimization-based Robot Control

Andrea Del Prete

University of Trento

Table of contents

1. From Joint Space to Task Space Control
2. Task Models
3. Optimization-Based Control
4. Multi-Task Control

From Joint Space to Task Space Control

Limits of Joint-Space Control

Joint-space control needs reference joint trajectory $q^r(t)$.

Limits of Joint-Space Control

Joint-space control needs reference joint trajectory $q^r(t)$.

What if we have reference trajectory $x^r(t)$ for end-effector?

Option 1: Mapping End-Effector Space to Joint Space

Compute joint trajectory $q^r(t)$ corresponding to $x^r(t)$, then apply joint-space control:

Option 1: Mapping End-Effector Space to Joint Space

Compute joint trajectory $q^r(t)$ corresponding to $x^r(t)$, then apply joint-space control:

$$\text{Find } q^r(t) \quad \text{such that} \quad FG(q^r(t)) = x^r(t) \qquad \forall t \in [0, T],$$

Option 1: Mapping End-Effector Space to Joint Space

Compute joint trajectory $q^r(t)$ corresponding to $x^r(t)$, then apply joint-space control:

$$
\begin{aligned}
\text{Find } q^r(t) \quad \text{such that} \quad FG(q^r(t)) &= x^r(t) \qquad \forall t \in [0, T], \\
\rightarrow \quad q^r(t) &= FG^{\dagger}(x^r(t)) \qquad \forall t \in [0, T],
\end{aligned} \tag{1}
$$

where:

- $FG(.) \triangleq$ forward geometry function of end-effector
- $FG^{\dagger}(.)$ is such that $FG(FG^{\dagger}(x)) = x, \forall x$

Option 1: Mapping End-Effector Space to Joint Space

Compute joint trajectory $q^r(t)$ corresponding to $x^r(t)$, then apply joint-space control:

$$\begin{aligned} \text{Find}\, q^r(t) \quad \text{such that} \quad & FG(q^r(t)) = x^r(t) \qquad \forall t \in [0, T], \\ \rightarrow \quad & q^r(t) = FG^\dagger(x^r(t)) \qquad \forall t \in [0, T], \end{aligned} \tag{1}$$

where:

- $FG(.) \triangleq$ forward geometry function of end-effector
- $FG^\dagger(.)$ is such that $FG(FG^\dagger(x)) = x, \forall x$

ISSUES

Problem (1) is challenging (Inverse Geometry, nonconvex problem with infinitely many solutions).

Option 1: Mapping End-Effector Space to Joint Space

Compute joint trajectory $q^r(t)$ corresponding to $x^r(t)$, then apply joint-space control:

$$\begin{aligned} \text{Find } q^r(t) \quad \text{such that} \quad FG(q^r(t)) = x^r(t) \qquad \forall t \in [0, T], \\ \rightarrow \quad q^r(t) = FG^\dagger(x^r(t)) \qquad \forall t \in [0, T], \end{aligned} \tag{1}$$

where:

- $FG(.) \triangleq$ forward geometry function of end-effector
- $FG^\dagger(.)$ is such that $FG(FG^\dagger(x)) = x, \forall x$

ISSUES

Problem (1) is challenging (Inverse Geometry, nonconvex problem with infinitely many solutions).

Tracking $q^r(t)$ is sufficient but not necessary to track $x^r(t)$: controller rejects also perturbations affecting q without affecting $FG(q)$.

Option 2: End-Effector Control

Feedback directly end-effector configuration:

$$\ddot{x}^d = \ddot{x}^r - K_d(\dot{x} - \dot{x}^r) - K_p(x - x^r) \tag{2}$$

Option 2: End-Effector Control

Feedback directly end-effector configuration:

$$\ddot{x}^d = \ddot{x}^r - K_d(\dot{x} - \dot{x}^r) - K_p(x - x^r) \tag{2}$$

Differenciate relationship between q and x:

$$x = FG(q)$$

Option 2: End-Effector Control

Feedback directly end-effector configuration:

$$\ddot{x}^d = \ddot{x}^r - K_d(\dot{x} - \dot{x}^r) - K_p(x - x^r) \tag{2}$$

Differenciate relationship between q and x:

$$\begin{aligned} x &= FG(q) \\ \dot{x} &= \frac{d}{dt}FG(q) = \underbrace{\frac{d\,FG}{dq}}_{J}\frac{dq}{dt} = Jv \end{aligned}$$

Option 2: End-Effector Control

Feedback directly end-effector configuration:

$$\ddot{x}^d = \ddot{x}^r - K_d(\dot{x} - \dot{x}^r) - K_p(x - x^r) \tag{2}$$

Differenciate relationship between q and x:

$$\begin{aligned} x &= FG(q) \\ \dot{x} &= \frac{d}{dt}FG(q) = \underbrace{\frac{d\,FG}{dq}}_{J}\frac{dq}{dt} = Jv \\ \ddot{x} &= J\dot{v} + \dot{J}v \end{aligned} \tag{3}$$

Option 2: End-Effector Control

Feedback directly end-effector configuration:

$$\ddot{x}^d = \ddot{x}^r - K_d(\dot{x} - \dot{x}^r) - K_p(x - x^r) \tag{2}$$

Differenciate relationship between q and x:

$$\begin{aligned} x &= FG(q) \\ \dot{x} &= \frac{d}{dt}FG(q) = \underbrace{\frac{d\,FG}{dq}}_{J}\frac{dq}{dt} = Jv \\ \ddot{x} &= J\dot{v} + \dot{J}v \end{aligned} \tag{3}$$

Desired accelerations should be:

$$\dot{v}^d = J^\dagger(\ddot{x}^d - \dot{J}v) \tag{4}$$

Option 2: End-Effector Control

Feedback directly end-effector configuration:

$$\ddot{x}^d = \ddot{x}^r - K_d(\dot{x} - \dot{x}^r) - K_p(x - x^r) \tag{2}$$

Differenciate relationship between q and x:

$$\begin{aligned} x &= FG(q) \\ \dot{x} &= \frac{d}{dt}FG(q) = \underbrace{\frac{d\,FG}{dq}}_{J}\frac{dq}{dt} = Jv \\ \ddot{x} &= J\dot{v} + \dot{J}v \end{aligned} \tag{3}$$

Desired accelerations should be:

$$\dot{v}^d = J^\dagger(\ddot{x}^d - \dot{J}v) \tag{4}$$

Finally compute joint torques as:

$$\tau = M\dot{v}^d + h \tag{5}$$

Option 1 VS Option 2

To summarize, both options compute joint torques as:

$$\tau = M\dot{v}^d + h \tag{6}$$

Option 1 VS Option 2

To summarize, both options compute joint torques as:

$$\tau = M\dot{v}^d + h \tag{6}$$

Option 1 computes $\dot{v}^d$ as:

$$\dot{v}^d = \dot{v}^r - PD(q - FG^\dagger(x^r)) \tag{7}$$

FG is "inverted" at configuration level.

Option 1 VS Option 2

To summarize, both options compute joint torques as:

$$\tau = M\dot{v}^d + h \tag{6}$$

Option 1 computes $\dot{v}^d$ as:

$$\dot{v}^d = \dot{v}^r - PD(q - FG^\dagger(x^r)) \tag{7}$$

FG is "inverted" at configuration level.

Option 2 computes $\dot{v}^d$ as:

$$\dot{v}^d = J^\dagger(\ddot{x}^r - PD(x - x^r) - \dot{J}v) \tag{8}$$

FG is "inverted" at acceleration level.

Option 1 VS Option 2

Option 2 typically preferred:

Option 1 VS Option 2

Option 2 typically preferred:

+ Gains defined in Cartesian space

Option 1 VS Option 2

Option 2 typically preferred:

+ Gains defined in Cartesian space

+ No pre-computations

Option 1 VS Option 2

Option 2 typically preferred:

\+ Gains defined in Cartesian space

\+ No pre-computations

\+ Online specification of reference trajectory

Option 1 VS Option 2

Option 2 typically preferred:

+ Gains defined in Cartesian space

+ No pre-computations

+ Online specification of reference trajectory

- More complex controller

End-Effector Control as LSP

End-effector control law (Option 2):

$$\begin{aligned}\tau &= M\dot{v}^d + h \\ \dot{v}^d &= J^\dagger(\ddot{x}^d - \dot{J}v) \\ \ddot{x}^d &= \ddot{x}^r - PD(x - x^r)\end{aligned} \tag{9}$$

End-Effector Control as LSP

End-effector control law (Option 2):

$$\begin{aligned} \tau &= M\dot{v}^d + h \\ \dot{v}^d &= J^\dagger(\ddot{x}^d - \dot{J}v) \\ \ddot{x}^d &= \ddot{x}^r - PD(x - x^r) \end{aligned} \tag{9}$$

can be computed as:

$$\begin{aligned} \underset{\tau,\dot{v}}{\text{minimize}} \quad & ||J\dot{v} + \dot{J}v - \ddot{x}^d||^2 \\ \text{subject to} \quad & M\dot{v} + h = \tau \end{aligned} \tag{10}$$

Task Models

Task-Function Approach

Generalize concept of end-effector with Task.

Task-Function Approach

Generalize concept of end-effector with Task.

Task = control objective.

Task-Function Approach

Generalize concept of end-effector with Task.

Task = control objective.

Describe tasks as functions e to minimize (as in optimal control).

Task-Function Approach

Generalize concept of end-effector with Task.

Task = control objective.

Describe tasks as functions e to minimize (as in optimal control).

Assume e measures error between real and reference output $y \in \mathbb{R}^m$:

$$\underbrace{e(x, u, t)}_{\text{error}} = \underbrace{y(x, u)}_{\text{real}} - \underbrace{y^*(t)}_{\text{reference}}$$

Task-Function Approach

Generalize concept of end-effector with Task.

Task = control objective.

Describe tasks as functions e to minimize (as in optimal control).

Assume e measures error between real and reference output $y \in \mathbb{R}^m$:

$$\underbrace{e(x, u, t)}_{\text{error}} = \underbrace{y(x, u)}_{\text{real}} - \underbrace{y^*(t)}_{\text{reference}}$$

N.B.

Here: e depends on instantaneous state-control value.
In optimal control: e depends on state-control trajectory.

Task-Function Types

IDEA

Given $e(x, u, t)$, find affine function of $\dot{v}$ and u to minimize.

Task-Function Types

IDEA

Given $e(x, u, t)$, find affine function of $\dot{v}$ and u to minimize.

Three kinds of task functions:

- Affine functions of u: $e(u, t) = A_u u - a(t)$
- Nonlinear functions of v: $e(v, t) = y(v) - y^*(t)$
- Nonlinear functions of q: $e(q, t) = y(q) - y^*(t)$

Task-Function Types

IDEA

Given $e(x, u, t)$, find affine function of $\dot{v}$ and u to minimize.

Three kinds of task functions:

- Affine functions of u: $e(u, t) = A_u u - a(t)$
- Nonlinear functions of v: $e(v, t) = y(v) - y^*(t)$
- Nonlinear functions of q: $e(q, t) = y(q) - y^*(t)$

Issue

q and v are not variables in Inverse Dynamics LSP.

Task-Function Types

IDEA

Given $e(x, u, t)$, find affine function of $\dot{v}$ and u to minimize.

Three kinds of task functions:

- Affine functions of u: $e(u, t) = A_u u - a(t)$
- Nonlinear functions of v: $e(v, t) = y(v) - y^*(t)$
- Nonlinear functions of q: $e(q, t) = y(q) - y^*(t)$

Issue

q and v are not variables in Inverse Dynamics LSP.

Solution

Impose dynamics of $e(x, t)$ (e.g., $\dot{e} = ...$)
which should be affine function of $\dot{v}$
such that $\lim_{t \to \infty} e(x, t) = 0$

Velocity Task-Function

Consider task function: $e(v, t) = y(v) - y^*(t)$.

Velocity Task-Function

Consider task function: $e(v, t) = y(v) - y^*(t)$.

Impose first-order linear dynamic:

$$\dot{e} = -Ke$$

Velocity Task-Function

Consider task function: $e(v,t) = y(v) - y^*(t)$.

Impose first-order linear dynamic:

$$
\begin{aligned}
\dot{e} &= -Ke \\
\underbrace{\frac{\partial y}{\partial v}}_{Jacobian} \dot{v} - \dot{y}^* &= -Ke
\end{aligned}
$$

Velocity Task-Function

Consider task function: $e(v,t) = y(v) - y^*(t)$.

Impose first-order linear dynamic:

$$\begin{aligned} \dot{e} &= -Ke \\ \underbrace{\frac{\partial y}{\partial v}}_{Jacobian} \dot{v} - \dot{y}^* &= -Ke \\ \underbrace{J}_{A_v} \dot{v} &= \underbrace{\dot{y}^* - Ke}_{a} \end{aligned} \tag{11}$$

Velocity Task-Function

Consider task function: $e(v,t) = y(v) - y^*(t)$.

Impose first-order linear dynamic:

$$\begin{aligned}\dot{e} &= -Ke \\ \underbrace{\frac{\partial y}{\partial v}}_{Jacobian} \dot{v} - \dot{y}^* &= -Ke \\ \underbrace{J}_{A_v} \dot{v} &= \underbrace{\dot{y}^* - Ke}_{a}\end{aligned} \tag{11}$$

We got affine function of $\dot{v}$.

Velocity Task-Function

Consider task function: $e(v,t) = y(v) - y^*(t)$.

Impose first-order linear dynamic:

$$\begin{aligned} \dot{e} &= -Ke \\ \underbrace{\frac{\partial y}{\partial v}}_{Jacobian} \dot{v} - \dot{y}^* &= -Ke \\ \underbrace{J}_{A_v} \dot{v} &= \underbrace{\dot{y}^* - Ke}_{a} \end{aligned} \tag{11}$$

We got affine function of $\dot{v}$.

N.B.

Could also impose nonlinear dynamics, but linear is ok for most cases.

Configuration Task-Function

Consider task function: $e(q, t) = y(q) - y^*(t)$.

Configuration Task-Function

Consider task function: $e(q, t) = y(q) - y^*(t)$.

Impose second-order linear dynamics:

$$\ddot{e} = -Ke - D\dot{e}$$

Configuration Task-Function

Consider task function: $e(q, t) = y(q) - y^*(t)$.

Impose second-order linear dynamics:

$$\begin{aligned} \ddot{e} &= -Ke - D\dot{e} \\ J\dot{v} + \dot{J}v - \ddot{y}^* &= -Ke - D\dot{e} \end{aligned}$$

Configuration Task-Function

Consider task function: $e(q,t) = y(q) - y^*(t)$.

Impose second-order linear dynamics:

$$\begin{aligned} \ddot{e} &= -Ke - D\dot{e} \\ J\dot{v} + \dot{J}v - \ddot{y}^* &= -Ke - D\dot{e} \\ \underbrace{J}_{A_v}\dot{v} &= \underbrace{\ddot{y}^* - \dot{J}v - Ke - D\dot{e}}_{a} \end{aligned} \tag{12}$$

Configuration Task-Function

Consider task function: $e(q, t) = y(q) - y^*(t)$.

Impose second-order linear dynamics:

$$\begin{aligned}
\ddot{e} &= -Ke - D\dot{e} \\
J\dot{v} + \dot{J}v - \ddot{y}^* &= -Ke - D\dot{e} \\
\underbrace{J}_{A_v}\dot{v} &= \underbrace{\ddot{y}^* - \dot{J}v - Ke - D\dot{e}}_{a}
\end{aligned} \tag{12}$$

We got affine function of $\dot{v}$.

Configuration Task-Function

Consider task function: $e(q, t) = y(q) - y^*(t)$.

Impose second-order linear dynamics:

$$\begin{aligned} \ddot{e} &= -Ke - D\dot{e} \\ J\dot{v} + \dot{J}v - \ddot{y}^* &= -Ke - D\dot{e} \\ \underbrace{J}_{A_v} \dot{v} &= \underbrace{\ddot{y}^* - \dot{J}v - Ke - D\dot{e}}_{a} \end{aligned} \tag{12}$$

We got affine function of $\dot{v}$.

N.B.

Could also impose nonlinear dynamics, but linear is ok for most cases.

From Euclidian Spaces to Lie Groups

So far $y(x, u) \in \mathbb{R}^m$.

From Euclidian Spaces to Lie Groups

So far $y(x, u) \in \mathbb{R}^m$.

What if $y(x, u) \in SE(3)$? (very common in practice)

From Euclidian Spaces to Lie Groups

So far $y(x, u) \in \mathbb{R}^m$.

What if $y(x, u) \in SE(3)$? (very common in practice)

SOLUTION Represent SE(3) elements using homogeneous matrices $y \in \mathbb{R}^{4\times4}$ and redefine error function:

$$e(q, t) = \log(y^*(t)^{-1} y(q)),$$

where $\log \triangleq$ inverse operation of matrix exponential (i.e. exponential map): transforms displacement into twist.

Task-Function Types: Summary

Functions of $u \rightarrow$ affine.

Task-Function Types: Summary

Functions of $u \rightarrow$ affine.

Functions of $x \rightarrow$ nonlinear, but cannot be directly imposed.

Task-Function Types: Summary

Functions of $u \to$ affine.

Functions of $x \to$ nonlinear, but cannot be directly imposed.

- For functions of v impose first derivative.

Task-Function Types: Summary

Functions of $u \rightarrow$ affine.

Functions of $x \rightarrow$ nonlinear, but cannot be directly imposed.

- For functions of v impose first derivative.
- For functions of q impose second derivative.

Task-Function Types: Summary

Functions of $u \to$ affine.

Functions of $x \to$ nonlinear, but cannot be directly imposed.

- For functions of v impose first derivative.
- For functions of q impose second derivative.

End up with affine function of $\dot{v}$ and u:

$$g(z) \triangleq \underbrace{\begin{bmatrix} A_v & A_u \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \dot{v} \\ u \end{bmatrix}}_{z} - a$$

Optimization-Based Control

Task-Space Inverse Dynamics (TSID)

Find τ that minimizes task function:

$$\begin{aligned} \underset{z=(\dot{v},\tau)}{\text{minimize}} \quad & ||Az - a||^2 \\ \text{subject to} \quad & \begin{bmatrix} M & -S^\top \end{bmatrix} z = -h \end{aligned} \tag{13}$$

TSID for Robots in Soft Contact

If system in contact $\rightarrow$ account for contact forces f.

TSID for Robots in Soft Contact

If system in contact $\rightarrow$ account for contact forces f.

If contacts are soft, use estimated forces $\hat{f}$:

$$\begin{aligned} \underset{z=(\dot{v},\tau)}{\text{minimize}} \quad & ||Az - a||^2 \\ \text{subject to} \quad & \begin{bmatrix} M & -S^\top \end{bmatrix} z = -h + J^\top \hat{f} \end{aligned} \tag{14}$$

TSID for Robots in Rigid Contact

Rigid contacts constrain motion.

$$c(q) = \text{const} \qquad \Longleftrightarrow \qquad \text{Contact points do not move}$$

TSID for Robots in Rigid Contact

Rigid contacts constrain motion.

$$c(q) = \text{const} \iff \text{Contact points do not move}$$

$$Jv = 0 \iff \text{Contact point velocities are null}$$

$$J\dot{v} + \dot{J}v = 0 \iff \text{Contact point accelerations are null}$$

TSID for Robots in Rigid Contact

Rigid contacts constrain motion.

$$
\begin{aligned}
c(q) = \text{const} \quad &\Longleftrightarrow \quad \text{Contact points do not move} \\
Jv = 0 \quad &\Longleftrightarrow \quad \text{Contact point velocities are null} \\
J\dot{v} + \dot{J}v = 0 \quad &\Longleftrightarrow \quad \text{Contact point accelerations are null}
\end{aligned}
$$

Introduce forces and constraints:

$$
\begin{aligned}
\underset{z=(\dot{v}, f, \tau)}{\text{minimize}} \quad & ||Az - a||^2 \\
\text{subject to} \quad & \begin{bmatrix} J & 0 & 0 \\ M & -J^\top & -S^\top \end{bmatrix} z = \begin{bmatrix} -\dot{J}v \\ -h \end{bmatrix}
\end{aligned} \tag{15}
$$

Inequality Constraints

Benefit of optimization: inequality constraints.

Inequality Constraints

Benefit of optimization: inequality constraints.

Any inequality affine in $z = (\tau, f, \dot{v})$:

- joint torque bounds: $\tau^{min} \leq \tau \leq \tau^{max}$
- (linearized) force friction cones: $Bf \leq 0$
- joint bounds: $\dot{v}^{min} \leq \dot{v} \leq \dot{v}^{max}$
- collision avoidance (more complicated)

Multi-Task Control

Multi-Objective Optimization

Complex robots are redundant w.r.t. task they perform

Multi-Objective Optimization

Complex robots are redundant w.r.t. task they perform:

- 7-DoF manipulator that controls end-effector placement (6 DoFs) has 1 DoF of redundancy

Multi-Objective Optimization

Complex robots are redundant w.r.t. task they perform:

- 7-DoF manipulator that controls end-effector placement (6 DoFs) has 1 DoF of redundancy
- 18-DoF biped that controls placement of two feet (12 DoFs) has 6 DoFs of redundancy

Multi-Objective Optimization

Complex robots are redundant w.r.t. task they perform:

- 7-DoF manipulator that controls end-effector placement (6 DoFs) has 1 DoF of redundancy
- 18-DoF biped that controls placement of two feet (12 DoFs) has 6 DoFs of redundancy

Can use redundancy to execute secondary tasks, but how?

Weighted Multi-Objective Optimization

N tasks, each defined by task function

$$g_i(z) = ||A_i z - a_i||^2 \qquad i = 1 \dots N$$

Weighted Multi-Objective Optimization

N tasks, each defined by task function

$$g_i(z) = ||A_i z - a_i||^2 \qquad i = 1 \dots N$$

Simplest strategy: sum functions using user-defined weights w_i:

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

Weighted Multi-Objective Optimization

N tasks, each defined by task function

$$g_i(z) = ||A_i z - a_i||^2 \qquad i = 1 \dots N$$

Simplest strategy: sum functions using user-defined weights w_i:

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

PROS Problem remains computationally-efficient LSP.

Weighted Multi-Objective Optimization

N tasks, each defined by task function

$$g_i(z) = ||A_i z - a_i||^2 \qquad i = 1 \dots N$$

Simplest strategy: sum functions using user-defined weights w_i:

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

PROS Problem remains computationally-efficient LSP.

CONS Hard to find weights $\rightarrow$ too large/small weights lead to numerical issues.

Hierarchical Multi-Objective Optimization

Alternative: order tasks according to priority

Hierarchical Multi-Objective Optimization

Alternative: order tasks according to priority

- task 1 more important than task 2

Hierarchical Multi-Objective Optimization

Alternative: order tasks according to priority

- task 1 more important than task 2
- ...
- task N-1 more important than task N

Solve sequence (cascade) of N problems, from $i = 1$:

$$
\begin{aligned}
g_i^* = \underset{z=(\dot{v}, f, \tau)}{\text{minimize}} \quad & g_i(z) \\
\text{subject to} \quad & \begin{bmatrix} J & 0 & 0 \\ M & -J^\top & -S^\top \end{bmatrix} z = \begin{bmatrix} -\dot{J}v \\ -h \end{bmatrix} \\
& g_j(z) = g_j^* \qquad \forall j < i
\end{aligned}
$$

Hierarchical Multi-Objective Optimization

Alternative: order tasks according to priority

- task 1 more important than task 2
- ...
- task N-1 more important than task N

Solve sequence (cascade) of N problems, from $i = 1$:

$$
\begin{aligned}
g_i^* = \underset{z=(\dot{v},f,\tau)}{\text{minimize}} \quad & g_i(z) \\
\text{subject to} \quad & \begin{bmatrix} J & 0 & 0 \\ M & -J^\top & -S^\top \end{bmatrix} z = \begin{bmatrix} -\dot{J}v \\ -h \end{bmatrix} \\
& g_j(z) = g_j^* \qquad \forall j < i
\end{aligned}
$$

PROS Easier to find priorities than weights.

Hierarchical Multi-Objective Optimization

Alternative: order tasks according to priority

- task 1 more important than task 2
- ...
- task N-1 more important than task N

Solve sequence (cascade) of N problems, from $i = 1$:

$$\begin{aligned} g_i^* = \underset{z=(\dot{v},f,\tau)}{\text{minimize}} \quad & g_i(z) \\ \text{subject to} \quad & \begin{bmatrix} J & 0 & 0 \\ M & -J^\top & -S^\top \end{bmatrix} z = \begin{bmatrix} -\dot{J}v \\ -h \end{bmatrix} \\ & g_j(z) = g_j^* \qquad \forall j < i \end{aligned}$$

PROS Easier to find priorities than weights.

CONS More computationally expensive to solve several LSPs.