Task-Space Inverse Dynamics

Quadratic-Programming based Control for Legged Robots

Andrea Del Prete

University of Trento

This document explains the control framework known as Task-Space Inverse Dynamics (TSID).

TSID is a popular control framework for legged robots.

Introduction

This document explains the control framework known as Task-Space Inverse Dynamics (TSID).

TSID is a popular control framework for legged robots.

It all started in 1987 with this paper by *Oussama Khatib*: "A unified approach for motion and force control of robot manipulators: The operational space formulation" [6]

Introduction

This document explains the control framework known as Task-Space Inverse Dynamics (TSID).

TSID is a popular control framework for legged robots.

It all started in 1987 with this paper by *Oussama Khatib*: "A unified approach for motion and force control of robot manipulators: The operational space formulation" [6]

Very active research topic between 2004 and 2015 [11, 7, 8, 10, 4, 3].

Introduction

This document explains the control framework known as Task-Space Inverse Dynamics (TSID).

TSID is a popular control framework for legged robots.

It all started in 1987 with this paper by *Oussama Khatib*: "A unified approach for motion and force control of robot manipulators: The operational space formulation" [6]

Very active research topic between 2004 and 2015 [11, 7, 8, 10, 4, 3].

Now not so active anymore (i.e. problem solved), but widely used.

Schedule

1. Theory ($\approx$ 1/1.5 hours)
2. Implementation ($\approx$ 0.5/1 hour)
3. Coding ($\approx$ 0.5/1 hour)

Schedule

1. Theory ($\approx$ 1/1.5 hours)
2. Implementation ($\approx$ 0.5/1 hour)
3. Coding ($\approx$ 0.5/1 hour)

Options for coding:

- install TSID in Nicolas's VM (recommended)
    - go to `https://github.com/stack-of-tasks/tsid/issues`
    - open issue 28 (should be the latest)
    - execute list of commands I posted

Schedule

1. Theory ($\approx$ 1/1.5 hours)
2. Implementation ($\approx$ 0.5/1 hour)
3. Coding ($\approx$ 0.5/1 hour)

Options for coding:

- install TSID in Nicolas's VM (recommended)
  - go to `https://github.com/stack-of-tasks/tsid/issues`
  - open issue 28 (should be the latest)
  - execute list of commands I posted
- use my 11 GB VM (prepared with *VMware Fusion*)

Schedule

1. Theory ($\approx$ 1/1.5 hours)
2. Implementation ($\approx$ 0.5/1 hour)
3. Coding ($\approx$ 0.5/1 hour)

Options for coding:

- install TSID in Nicolas's VM (recommended)
    - go to `https://github.com/stack-of-tasks/tsid/issues`
    - open issue 28 (should be the latest)
    - execute list of commands I posted
- use my 11 GB VM (prepared with *VMware Fusion*)
- install TSID and dependencies on your machine
    - TSID branch master $\rightarrow$ Pinocchio branch master (same as robotpkg binaries)
    - TSID branch pinocchio-v2 $\rightarrow$ Pinocchio branch devel

Notation & Definitions

The state of the system is denoted x.

The control inputs are denoted u.

Notation & Definitions

The state of the system is denoted x.

The control inputs are denoted u.

The identity matrix is denoted I. The zero matrix is denoted 0. When needed, the size of the matrix is written as index, e.g., I_3.

Notation & Definitions

The state of the system is denoted x.

The control inputs are denoted u.

The identity matrix is denoted I. The zero matrix is denoted 0. When needed, the size of the matrix is written as index, e.g., I_3.

A system is fully actuated if the number of actuators is equal to the number of degrees of freedom (e.g., manipulator).

A system is under actuated if the number of actuators is less than the number of degrees of freedom (e.g., legged robot, quadrotor).

Table of contents

Actuation Models

Actuation Models

Everything starts with a model (not really, but here it does).

Actuation Models

Everything starts with a model (not really, but here it does).

Appropriate model choice depends on both robot and task.

Actuation Models

Everything starts with a model (not really, but here it does).

Appropriate model choice depends on both robot and task.

Let us discuss three models for the robot actuators:

- velocity source
- acceleration source
- torque source

Velocity Control

Assume motors are velocity sources.

- Good approximation for hydraulic actuators.
- Good approximation for electric motors only in certain conditions (e.g., industrial manipulators, not for legged robots).

Velocity Control

Assume motors are velocity sources.

- Good approximation for hydraulic actuators.
- Good approximation for electric motors only in certain conditions (e.g., industrial manipulators, not for legged robots).

Robot state x is described by its configuration q.

Control inputs u are robot velocities v_q.

Dynamic for fully-actuated systems is a simple integrator:

$$v_q = u$$

Acceleration Control

Assume motors are acceleration sources.

- Good approximation for electric motors as long as large contact forces are not involved.

Acceleration Control

Assume motors are acceleration sources.

- Good approximation for electric motors as long as large contact forces are not involved.

Robot state x is described by its configuration q and its velocity v_q:

$$x \triangleq (q, v_q)$$

Control inputs u are robot accelerations $\dot{v}_q$.

Dynamic for fully-actuated systems is a double integrator:

$$\begin{bmatrix} v_q \\ \dot{v}_q \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ v_q \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u$$

Torque Control

Assume motors are torque sources. Good approximation for electric motors.

Robot state x is described by its configuration q and its velocity v_q:

$$x \triangleq (q, v_q)$$

Control inputs u are motor torques τ.

Torque Control: Fully-Actuated Dynamic

The dynamic equation of a fully-actuated mechanical system is:

$$M(q)\dot{v}_q + h(q, v_q) = \tau + J(q)^\top f,$$

where $M(q) \in \mathbb{R}^{n_v \times n_v}$ is the mass matrix, $h(q, v_q) \in \mathbb{R}^{n_v}$ are the bias forces, $\tau \in \mathbb{R}^{n_v}$ are the joint torques, $f \in \mathbb{R}^{n_f}$ are the contact forces, and $J(q) \in \mathbb{R}^{n_f \times n_v}$ is the contact Jacobian.

Torque Control: Fully-Actuated Dynamic

The dynamic equation of a fully-actuated mechanical system is:

$$M(q)\dot{v}_q + h(q, v_q) = \tau + J(q)^\top f,$$

where $M(q) \in \mathbb{R}^{n_v \times n_v}$ is the mass matrix, $h(q, v_q) \in \mathbb{R}^{n_v}$ are the bias forces, $\tau \in \mathbb{R}^{n_v}$ are the joint torques, $f \in \mathbb{R}^{n_f}$ are the contact forces, and $J(q) \in \mathbb{R}^{n_f \times n_v}$ is the contact Jacobian.

Bias forces are sometimes decomposed in two components:

$$h(q, v_q) = C(q, v_q)v_q + g(q)$$

- $C(q, v_q)v_q$ contains Coriolis and centrifugal effects
- $g(q)$ contains the gravity forces

Torque Control: Under-Actuated Systems

Underactuated systems (such as legged robots) have less actuators than degrees of freedom (DoFs). Calling n_{va} the number of actuators, and n_v the number of DoFs, we have $n_{va} < n_v$.

Torque Control: Under-Actuated Systems

Underactuated systems (such as legged robots) have less actuators than degrees of freedom (DoFs). Calling n_{va} the number of actuators, and n_v the number of DoFs, we have $n_{va} < n_v$.

Assume elements of q are ordered, $q \triangleq (q_u, q_a)$, where:

- $q_u \in \mathbb{R}^{n_{qu}}$ are the passive (unactuated) joints,
- $q_a \in \mathbb{R}^{n_{qa}}$ are the actuated joints.

Similarly, $v_q \triangleq (v_u, v_a)$, where $v_u \in \mathbb{R}^{n_{vu}}$ and $v_a \in \mathbb{R}^{n_{va}}$.

Torque Control: Under-Actuated Systems

Underactuated systems (such as legged robots) have less actuators than degrees of freedom (DoFs). Calling n_{va} the number of actuators, and n_v the number of DoFs, we have $n_{va} < n_v$.

Assume elements of q are ordered, $q \triangleq (q_u, q_a)$, where:

- $q_u \in \mathbb{R}^{n_{qu}}$ are the passive (unactuated) joints,
- $q_a \in \mathbb{R}^{n_{qa}}$ are the actuated joints.

Similarly, $v_q \triangleq (v_u, v_a)$, where $v_u \in \mathbb{R}^{n_{vu}}$ and $v_a \in \mathbb{R}^{n_{va}}$.

$S \triangleq \begin{bmatrix} 0_{n_{va} \times n_{vu}} & I_{n_{va}} \end{bmatrix}$ is a selection matrix associated to the actuated joints:

$$v_a = S v_q$$

Torque Control: Under-Actuated Dynamic

The dynamic of an under-actuated mechanical system is:

$$M(q)\dot{v}_q + h(q, v_q) = S^\top \tau + J(q)^\top f,$$

where, contrary to the fully-actuated case, $\tau \in \mathbb{R}^{n_{va}}$.

Torque Control: Under-Actuated Dynamic

The dynamic of an under-actuated mechanical system is:

$$M(q)\dot{v}_q + h(q, v_q) = S^\top \tau + J(q)^\top f,$$

where, contrary to the fully-actuated case, $\tau \in \mathbb{R}^{n_{va}}$.

This dynamic is often decomposed into unactuated and actuated parts:

$$\begin{aligned} M_u(q)\dot{v}_q + h_u(q, v_q) &= J_u(q)^\top f \\ M_a(q)\dot{v}_q + h_a(q, v_q) &= \tau + J_a(q)^\top f \end{aligned} \tag{1}$$

where

$$M = \begin{bmatrix} M_u \\ M_a \end{bmatrix} \quad h = \begin{bmatrix} h_u \\ h_a \end{bmatrix} \quad J = \begin{bmatrix} J_u & J_a \end{bmatrix} \tag{2}$$

Task Models

Task-Function Approach

IDEA: Describe task to be performed (i.e. control objective) as a function to minimize (similar to optimal control).

Task-Function Approach

IDEA: Describe task to be performed (i.e. control objective) as a function to minimize (similar to optimal control).

Without loss of generality, assume this function measures an error between real and reference value of 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

IDEA: Describe task to be performed (i.e. control objective) as a function to minimize (similar to optimal control).

Without loss of generality, assume this function measures an error between real and reference value of 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.

Contrary to an optimal control cost function, e does not depend on the state-control trajectory, but only on the instantaneous state-control value.

Task-Function Types

Consider three kinds of task functions:

- Affine functions of control inputs: $e(u, t) = A_u u - a(t)$
- Nonlinear functions of robot velocities: $e(v_q, t) = y(v_q) - y^*(t)$
- Nonlinear functions of robot configuration: $e(q, t) = y(q) - y^*(t)$

Task-Function Types

Consider three kinds of task functions:

- Affine functions of control inputs: $e(u, t) = A_u u - a(t)$
- Nonlinear functions of robot velocities: $e(v_q, t) = y(v_q) - y^*(t)$
- Nonlinear functions of robot configuration: $e(q, t) = y(q) - y^*(t)$

Issue

Control inputs u can be instantaneously changed, but that is not the case for the state x.

Task-Function Types

Consider three kinds of task functions:

- Affine functions of control inputs: $e(u, t) = A_u u - a(t)$
- Nonlinear functions of robot velocities: $e(v_q, t) = y(v_q) - y^*(t)$
- Nonlinear functions of robot configuration: $e(q, t) = y(q) - y^*(t)$

Issue

Control inputs u can be instantaneously changed, but that is not the case for the state x.

Solution

Impose dynamic of task function $e(x, t)$ such that $\lim_{t \to \infty} e(x, t) = 0$

Velocity Task-Function

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

Velocity Task-Function

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

Let us impose a first-order linear dynamic:

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

Velocity Task-Function

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

Let us impose a first-order linear dynamic:

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

$$\underbrace{\frac{\partial y}{\partial v_q}}_{Jacobian} \dot{v}_q - \dot{y}^* = -Ke$$

Velocity Task-Function

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

Let us impose a first-order linear dynamic:

$$
\begin{aligned}
\dot{e} &= -Ke \\
\underbrace{\frac{\partial y}{\partial v_q}}_{Jacobian} \dot{v}_q - \dot{y}^* &= -Ke \\
\underbrace{J}_{A_v} \dot{v}_q &= \underbrace{\dot{y}^* - Ke}_{a}
\end{aligned}
\tag{3}
$$

Velocity Task-Function

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

Let us impose a first-order linear dynamic:

$$\begin{aligned} \dot{e} &= -Ke \\ \underbrace{\frac{\partial y}{\partial v_q}}_{Jacobian} \dot{v}_q - \dot{y}^* &= -Ke \\ \underbrace{J}_{A_v} \dot{v}_q &= \underbrace{\dot{y}^* - Ke}_{a} \end{aligned} \tag{3}$$

We got an affine function of the accelerations $\dot{v}_q$.

Velocity Task-Function

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

Let us impose a first-order linear dynamic:

$$
\begin{aligned}
\dot{e} &= -Ke \\
\underbrace{\frac{\partial y}{\partial v_q}}_{Jacobian} \dot{v}_q - \dot{y}^* &= -Ke \\
\underbrace{J}_{A_v} \dot{v}_q &= \underbrace{\dot{y}^* - Ke}_{a}
\end{aligned}
\tag{3}
$$

We got an affine function of the accelerations $\dot{v}_q$.

N.B.

We could also impose a nonlinear dynamic, but in practice a linear dynamic is ok for most cases.

Configuration Task-Function

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

Configuration Task-Function

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

Let us impose a second-order linear dynamic:

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

Configuration Task-Function

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

Let us impose a second-order linear dynamic:

$$\ddot{e} = -Ke - D\dot{e}$$
$$J\dot{v}_q + \dot{J}v_q - \ddot{y}^* = -Ke - D\dot{e}$$

Configuration Task-Function

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

Let us impose a second-order linear dynamic:

$$\begin{aligned} \ddot{e} &= -Ke - D\dot{e} \\ J\dot{v}_q + \dot{J}v_q - \ddot{y}^* &= -Ke - D\dot{e} \\ \underbrace{J}_{A_v}\dot{v}_q &= \underbrace{\ddot{y}^* - \dot{J}v_q - Ke - D\dot{e}}_{a} \end{aligned} \tag{4}$$

Configuration Task-Function

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

Let us impose a second-order linear dynamic:

$$\begin{aligned} \ddot{e} &= -Ke - D\dot{e} \\ J\dot{v}_q + \dot{J}v_q - \ddot{y}^* &= -Ke - D\dot{e} \\ \underbrace{J}_{A_v}\dot{v}_q &= \underbrace{\ddot{y}^* - \dot{J}v_q - Ke - D\dot{e}}_{a} \end{aligned} \tag{4}$$

Again, we got an affine function of the accelerations $\dot{v}_q$.

Configuration Task-Function

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

Let us impose a second-order linear dynamic:

$$\begin{aligned} \ddot{e} &= -Ke - D\dot{e} \\ J\dot{v}_q + \dot{J}v_q - \ddot{y}^* &= -Ke - D\dot{e} \\ \underbrace{J}_{A_v}\dot{v}_q &= \underbrace{\ddot{y}^* - \dot{J}v_q - Ke - D\dot{e}}_{a} \end{aligned} \tag{4}$$

Again, we got an affine function of the accelerations $\dot{v}_q$.

N.B.

We could also impose a nonlinear dynamic, but in practice a linear dynamic is ok for most cases.

Task-Function Types: Summary

Task functions can depend either on u, or on $x \triangleq (q, v_q)$.

Task-Function Types: Summary

Task functions can depend either on u, or on $x \triangleq (q, v_q)$.

Functions of u must be affine.

Task-Function Types: Summary

Task functions can depend either on u, or on $x \triangleq (q, v_q)$.

Functions of u must be affine.

Functions of x can be nonlinear, but cannot be directly imposed.

Task-Function Types: Summary

Task functions can depend either on u, or on $x \triangleq (q, v_q)$.

Functions of u must be affine.

Functions of x can be nonlinear, but cannot be directly imposed.

- For functions of v_q we can impose first derivative.

Task-Function Types: Summary

Task functions can depend either on u, or on $x \triangleq (q, v_q)$.

Functions of u must be affine.

Functions of x can be nonlinear, but cannot be directly imposed.

- For functions of v_q we can impose first derivative.
- For functions of q we can impose second derivative.

Task-Function Types: Summary

Task functions can depend either on u, or on $x \triangleq (q, v_q)$.

Functions of u must be affine.

Functions of x can be nonlinear, but cannot be directly imposed.

- For functions of v_q we can impose first derivative.
- For functions of q we can impose second derivative.

In any case, we end up with an affine function of $\dot{v}_q$ and u:

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

Optimization-Based Control

Control as an Optimization Problem

IDEA: formulate control problem as an optimization problem (similar to optimal control).

Control as an Optimization Problem

IDEA: formulate control problem as an optimization problem (similar to optimal control).

Key elements are:

- state: $x \triangleq (q, v_q)$

Control as an Optimization Problem

IDEA: formulate control problem as an optimization problem (similar to optimal control).

Key elements are:

- state: $x \triangleq (q, v_q)$
- control: $u \triangleq \tau$

Control as an Optimization Problem

IDEA: formulate control problem as an optimization problem (similar to optimal control).

Key elements are:

- state: $x \triangleq (q, v_q)$
- control: $u \triangleq \tau$
- dynamic (no contacts): $M\dot{v}_q + h = S^\top \tau$

Control as an Optimization Problem

IDEA: formulate control problem as an optimization problem (similar to optimal control).

Key elements are:

- state: $x \triangleq (q, v_q)$
- control: $u \triangleq \tau$
- dynamic (no contacts): $M\dot{v}_q + h = S^\top \tau$
- task function to minimize: $||g(y)||^2 \triangleq ||Ay - a||^2$

Task-Space Inverse Dynamics (TSID)

Formulate optimization problem to find control inputs that minimize task function:

$$\begin{aligned} \underset{y=(\dot{v}_q,\tau)}{\text{minimize}} \quad & ||Ay - a||^2 \\ \text{subject to} \quad & \begin{bmatrix} M & -S^\top \end{bmatrix} y = -h \end{aligned} \tag{5}$$

Task-Space Inverse Dynamics (TSID)

Formulate optimization problem to find control inputs that minimize task function:

$$\begin{aligned} \underset{y=(\dot{v}_q,\tau)}{\text{minimize}} \quad & ||Ay - a||^2 \\ \text{subject to} \quad & \begin{bmatrix} M & -S^\top \end{bmatrix} y = -h \end{aligned} \tag{5}$$

Equality constraints are affine and cost function is convex quadratic
$\rightarrow$ Problem is a Quadratic Program (QP).

Task-Space Inverse Dynamics (TSID)

Formulate optimization problem to find control inputs that minimize task function:

$$\begin{aligned} \underset{y=(\dot{v}_q,\tau)}{\text{minimize}} \quad & ||Ay - a||^2 \\ \text{subject to} \quad & \begin{bmatrix} M & -S^\top \end{bmatrix} y = -h \end{aligned} \tag{5}$$

Equality constraints are affine and cost function is convex quadratic
$\rightarrow$ Problem is a Quadratic Program (QP).

N.B.

To be precise, cost function is 2-norm of affine function, which is a special kind of convex quadratic function (linear term $A^\top a$ is in range space of Hessian $A^\top A$) $\rightarrow$ Problem is a Least-Squares Problem (LSP).

TSID for Robots in Soft Contact

If system is in contact with environment, its dynamic must account for contact forces f.

TSID for Robots in Soft Contact

If system is in contact with environment, its dynamic must account for contact forces f.

If contacts are soft, measured/estimated contact forces $\hat{f}$ can be easily included:

$$\begin{aligned} \underset{y=(\dot{v}_q,\tau)}{\text{minimize}} \quad & ||Ay - a||^2 \\ \text{subject to} \quad & \begin{bmatrix} M & -S^\top \end{bmatrix} y = -h + J^\top \hat{f} \end{aligned} \tag{6}$$

TSID for Robots in Rigid Contact

If contacts are rigid, they constrain the motion.

TSID for Robots in Rigid Contact

If contacts are rigid, they constrain the motion. Let us model the rigid contact constraints as nonlinear functions

$$c(q) = 0 \qquad \Longleftrightarrow \qquad \text{Contact points do not move}$$

TSID for Robots in Rigid Contact

If contacts are rigid, they constrain the motion. Let us model the rigid contact constraints as nonlinear functions

$$c(q) = 0 \qquad \Longleftrightarrow \qquad \text{Contact points do not move}$$

To express the constraints as functions of the problem variables we must differentiate them twice:

$$J v_q = 0 \qquad \Longleftrightarrow \qquad \text{Contact point velocities are null}$$

$$J \dot{v}_q + \dot{J} v_q = 0 \qquad \Longleftrightarrow \qquad \text{Contact point accelerations are null}$$

TSID for Robots in Rigid Contact

If contacts are rigid, they constrain the motion. Let us model the rigid contact constraints as nonlinear functions

$$c(q) = 0 \qquad \Longleftrightarrow \qquad \text{Contact points do not move}$$

To express the constraints as functions of the problem variables we must differentiate them twice:

$$\begin{aligned} J v_q = 0 \qquad &\Longleftrightarrow \qquad \text{Contact point velocities are null} \\ J \dot{v}_q + \dot{J} v_q = 0 \qquad &\Longleftrightarrow \qquad \text{Contact point accelerations are null} \end{aligned}$$

Introduce contact forces and contact constraints in optimization problem:

$$\begin{aligned} \underset{y=(\dot{v}_q, f, \tau)}{\text{minimize}} \quad & ||Ay - a||^2 \\ \text{subject to} \quad & \begin{bmatrix} J & 0 & 0 \\ M & -J^\top & -S^\top \end{bmatrix} y = \begin{bmatrix} -\dot{J} v_q \\ -h \end{bmatrix} \end{aligned} \tag{7}$$

QP vs Pseudo-inverses

So far we have seen only Equality-Constrained LSP (ECLSP).

QP vs Pseudo-inverses

So far we have seen only Equality-Constrained LSP (ECLSP).

Unconstrained LSP can be solved using pseudo-inverses, for instance:

$$y^* = \underset{y}{\operatorname{argmin}} \, ||Ay - a||^2 \quad \iff \quad y^* = A^\dagger a$$

QP vs Pseudo-inverses

So far we have seen only Equality-Constrained LSP (ECLSP).

Unconstrained LSP can be solved using pseudo-inverses, for instance:

$$y^* = \underset{y}{\operatorname{argmin}} ||Ay - a||^2 \quad \Longleftrightarrow \quad y^* = A^\dagger a$$

Also ECLSP can be solved using pseudo-inverses, for instance:

$$\begin{aligned} y^* = \quad & \underset{y}{\operatorname{argmin}} \quad ||Ay - a||^2 \quad \Longleftrightarrow \quad y^* = B^\dagger b + N_B (A N_B)^\dagger (a - A B^\dagger b) \\ & \text{subject to} \quad By = b \end{aligned}$$

where $N_B = I - B^\dagger B$ is the null-space projector of B.

QP vs Pseudo-inverses

So far we have seen only Equality-Constrained LSP (ECLSP).

Unconstrained LSP can be solved using pseudo-inverses, for instance:

$$y^* = \underset{y}{\operatorname{argmin}} ||Ay - a||^2 \quad \Longleftrightarrow \quad y^* = A^\dagger a$$

Also ECLSP can be solved using pseudo-inverses, for instance:

$$\begin{aligned} y^* = \quad & \underset{y}{\operatorname{argmin}} \quad ||Ay - a||^2 \quad \Longleftrightarrow \quad y^* = B^\dagger b + N_B(AN_B)^\dagger(a - AB^\dagger b) \\ & \text{subject to} \quad By = b \end{aligned}$$

where $N_B = I - B^\dagger B$ is the null-space projector of B.

QUESTION: if we can solve ECLSP with pseudo-inverses, why should we use a QP solver?

Inequality Constraints

Main benefit of QP solvers (over pseudo-inverses) is that they can handle inequality constraints.

Inequality Constraints

Main benefit of QP solvers (over pseudo-inverses) is that they can handle inequality constraints.

We can account for any inequality affine in problem variables y, such as:

- joint torque bounds: $\tau^{min} \leq \tau \leq \tau^{max}$
- (linearized) force friction cones: $Bf \leq 0$
- joint position-velocity bounds (after nontrivial transformation into acceleration bounds [1]): $\dot{v}_q^{min} \leq \dot{v}_q \leq \dot{v}_q^{max}$

Multi-Task Control

Multi-Objective Optimization

Complex robots are typically redundant with respect to the main task they must perform

Multi-Objective Optimization

Complex robots are typically redundant with respect to the main task they must perform, for instance:

- a 7-DoF manipulator that has to control its end-effector placement (6 DoFs) has 1 DoF of redundancy

Multi-Objective Optimization

Complex robots are typically redundant with respect to the main task they must perform, for instance:

- a 7-DoF manipulator that has to control its end-effector placement (6 DoFs) has 1 DoF of redundancy
- an 18-DoF biped robot that has to control the placement of its two feet (12 DoFs) has 6 DoFs of redundancy

Multi-Objective Optimization

Complex robots are typically redundant with respect to the main task they must perform, for instance:

- a 7-DoF manipulator that has to control its end-effector placement (6 DoFs) has 1 DoF of redundancy
- an 18-DoF biped robot that has to control the placement of its two feet (12 DoFs) has 6 DoFs of redundancy

Redundancy can be used to execute secondary tasks, but how to incorporate them in the optimization problem?

Weighted Multi-Objective Optimization

Assume robot must perform N tasks, each defined by a task function

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

Weighted Multi-Objective Optimization

Assume robot must perform N tasks, each defined by a task function

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

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

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

Weighted Multi-Objective Optimization

Assume robot must perform N tasks, each defined by a task function

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

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

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

PROS Problem remains standard computationally-efficient LSP.

Weighted Multi-Objective Optimization

Assume robot must perform N tasks, each defined by a task function

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

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

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

PROS Problem remains standard computationally-efficient LSP.

CONS Finding proper weights can be hard, too large/small weights can lead to numerical issues.

Hierarchical Multi-Objective Optimization

Alternative strategy: order task functions according to priority, that is

Hierarchical Multi-Objective Optimization

Alternative strategy: order task functions according to priority, that is

- task 1 is infinitely more important than task 2

Hierarchical Multi-Objective Optimization

Alternative strategy: order task functions according to priority, that is

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

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

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

Hierarchical Multi-Objective Optimization

Alternative strategy: order task functions according to priority, that is

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

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

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

PROS Finding priorities is easier than finding weights.

Hierarchical Multi-Objective Optimization

Alternative strategy: order task functions according to priority, that is

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

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

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

PROS Finding priorities is easier than finding weights.

CONS Solving several QPs can be too computationally expensive.

Computational Aspects

Computational Complexity of TSID

TSID needs to solve a QP at each control loop (embedded optimization, same spirit as MPC).

Computational Complexity of TSID

TSID needs to solve a QP at each control loop (embedded optimization, same spirit as MPC). $\rightarrow$ Limited computation time (1-10 ms).

Computational Complexity of TSID

TSID needs to solve a QP at each control loop (embedded optimization, same spirit as MPC). $\rightarrow$ Limited computation time (1-10 ms).

For n_v DoFs, n_{va} motors, and n_f contact constraints the QP has:

- $n_v + n_{va} + n_f$ variables ($\approx$ 70 for humanoid)
- $n_v + n_f$ equality constraints ($\approx$ 40 for humanoid)
- $n_v + n_{va} + \frac{4}{3} n_f$ inequality constraints (*assuming friction cones are approximated with 4-sided pyramids*)

Computational Complexity of TSID

TSID needs to solve a QP at each control loop (embedded optimization, same spirit as MPC). $\rightarrow$ Limited computation time (1-10 ms).

For n_v DoFs, n_{va} motors, and n_f contact constraints the QP has:

- $n_v + n_{va} + n_f$ variables (≈ 70 for humanoid)
- $n_v + n_f$ equality constraints (≈ 40 for humanoid)
- $n_v + n_{va} + \frac{4}{3} n_f$ inequality constraints (*assuming friction cones are approximated with 4-sided pyramids*)

Computational cost dominated by Hessian (Cholesky) decomposition: $\mathcal{O}(n^3)$, with n the number of variables.

Computational Complexity of TSID

TSID needs to solve a QP at each control loop (embedded optimization, same spirit as MPC). $\rightarrow$ Limited computation time (1-10 ms).

For n_v DoFs, n_{va} motors, and n_f contact constraints the QP has:

- $n_v + n_{va} + n_f$ variables (≈ 70 for humanoid)
- $n_v + n_f$ equality constraints (≈ 40 for humanoid)
- $n_v + n_{va} + \frac{4}{3} n_f$ inequality constraints (*assuming friction cones are approximated with 4-sided pyramids*)

Computational cost dominated by Hessian (Cholesky) decomposition: $\mathcal{O}(n^3)$, with n the number of variables.

QUESTIONS

- Can we solve such a problem in 1 ms?
- Is there a way to speed up computation?

Reformulating Optimization Problem

IDEA: Exploit structure of problem to make computation faster.

Reformulating Optimization Problem

IDEA: Exploit structure of problem to make computation faster.

Equality constraints have special structure:

$$\begin{bmatrix} J & 0 & 0 \\ M_u & -J_u^\top & -0 \\ M_a & -J_a^\top & -I \end{bmatrix} \begin{bmatrix} \dot{v}_q \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -\dot{J} v_q \\ -h_u \\ -h_a \end{bmatrix}$$

Reformulating Optimization Problem

IDEA: Exploit structure of problem to make computation faster.

Equality constraints have special structure:

$$\begin{bmatrix} J & 0 & 0 \\ M_u & -J_u^\top & -0 \\ M_a & -J_a^\top & -I \end{bmatrix} \begin{bmatrix} \dot{v}_q \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -\dot{J} v_q \\ -h_u \\ -h_a \end{bmatrix}$$

Identity matrix is easy to invert $\rightarrow$ We can easily express τ as affine function of other variables.

$$\underbrace{\begin{bmatrix} \dot{v}_q \\ f \\ \tau \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} I & 0 \\ 0 & I \\ M_a & -J_a^\top \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} \dot{v}_q \\ f \end{bmatrix}}_{\bar{y}} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ h_a \end{bmatrix}}_{d}$$

Reformulating Optimization Problem

Original problem:

$$
\begin{aligned}
\underset{y}{\text{minimize}} \quad & ||Ay - a||^2 \\
\text{subject to} \quad & By \leq b \\
& \begin{bmatrix} J & 0 & 0 \\ M_u & -J_u^\top & -0 \\ M_a & -J_a^\top & -I \end{bmatrix} \begin{bmatrix} \dot{v}_q \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -\dot{J} v_q \\ -h_u \\ -h_a \end{bmatrix}
\end{aligned}
$$

Reformulating Optimization Problem

Original problem:

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & ||Ay - a||^2 \\ \text{subject to} \quad & By \leq b \\ & \begin{bmatrix} J & 0 & 0 \\ M_u & -J_u^\top & -0 \\ M_a & -J_a^\top & -I \end{bmatrix} \begin{bmatrix} \dot{v}_q \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -\dot{J}v_q \\ -h_u \\ -h_a \end{bmatrix} \end{aligned}$$

Use $y = D\bar{y} + d$ to reformulate problem [5]:

$$\begin{aligned} \underset{\bar{y}}{\text{minimize}} \quad & ||AD\bar{y} + Ad - a||^2 \\ \text{subject to} \quad & BD\bar{y} \leq b - Bd \\ & \begin{bmatrix} J & 0 \\ M_u & -J_u^\top \end{bmatrix} \begin{bmatrix} \dot{v}_q \\ f \end{bmatrix} = \begin{bmatrix} -\dot{J}v_q \\ -h_u \end{bmatrix} \end{aligned}$$

Reformulating Optimization Problem

Original problem:

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & ||Ay - a||^2 \\ \text{subject to} \quad & By \leq b \\ & \begin{bmatrix} J & 0 & 0 \\ M_u & -J_u^\top & -0 \\ M_a & -J_a^\top & -I \end{bmatrix} \begin{bmatrix} \dot{v}_q \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -\dot{J} v_q \\ -h_u \\ -h_a \end{bmatrix} \end{aligned}$$

Use $y = D\bar{y} + d$ to reformulate problem [5]:

$$\begin{aligned} \underset{\bar{y}}{\text{minimize}} \quad & ||AD\bar{y} + Ad - a||^2 \\ \text{subject to} \quad & BD\bar{y} \leq b - Bd \\ & \begin{bmatrix} J & 0 \\ M_u & -J_u^\top \end{bmatrix} \begin{bmatrix} \dot{v}_q \\ f \end{bmatrix} = \begin{bmatrix} -\dot{J} v_q \\ -h_u \end{bmatrix} \end{aligned}$$

We have removed n_{va} variables and n_{va} equality constraints.

Reformulating Optimization Problem: Can We Do Better?

Can we improve even more?

Reformulating Optimization Problem: Can We Do Better?

Can we improve even more?

In theory, yes:

- for floating-base robots, remove first 6 variables of $\dot{v}_q$ exploiting structure of first 6 columns of M_u
- remove (either all [7, 9] or some [2]) force variables by projecting dynamics in null space of J

Reformulating Optimization Problem: Can We Do Better?

Can we improve even more?

In theory, yes:

- for floating-base robots, remove first 6 variables of $\dot{v}_q$ exploiting structure of first 6 columns of M_u
- remove (either all [7, 9] or some [2]) force variables by projecting dynamics in null space of J

BUT these tricks either limit the expressiveness of the problem, or lead to small improvements (while making the software more complex).

Reformulating Optimization Problem: Can We Do Better?

Can we improve even more?

In theory, yes:

- for floating-base robots, remove first 6 variables of $\dot{v}_q$ exploiting structure of first 6 columns of M_u
- remove (either all [7, 9] or some [2]) force variables by projecting dynamics in null space of J

BUT these tricks either limit the expressiveness of the problem, or lead to small improvements (while making the software more complex).

My opinion: probably not worth it!

From Euclidian Spaces to Lie Groups

So far we have assumed output function $y(x, u) \in \mathbb{R}^m$.

From Euclidian Spaces to Lie Groups

So far we have assumed output function $y(x, u) \in \mathbb{R}^m$.

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

From Euclidian Spaces to Lie Groups

So far we have assumed output function $y(x, u) \in \mathbb{R}^m$.

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

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

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

where log is the pseudo-inverse operation of the matrix exponential (i.e. exponential map): it transforms a displacement into a twist.

References i

A. Del Prete.
Joint Position and Velocity Bounds in Discrete-Time Acceleration / Torque Control of Robot Manipulators.
IEEE Robotics and Automation Letters, 3(1), 2018.

A. Del Prete, N. Mansard, F. Nori, G. Metta, and L. Natale.
Partial Force Control of Constrained Floating-Base Robots.
In *Intelligent Robots and Systems (IROS 2014), IEEE International Conference on*, 2014.

A. Del Prete, F. Nori, G. Metta, and L. Natale.
Prioritized Motion-Force Control of Constrained Fully-Actuated Robots: "Task Space Inverse Dynamics".
Robotics and Autonomous Systems, 63:150–157, 2015.

A. Escande, N. Mansard, and P.-B. Wieber.
Hierarchical Quadratic Programming: Fast Online Humanoid-Robot Motion Generation.
International Journal of Robotics Research, 33(7):1006–1028, 2014.

A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti.
Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid.
Autonomous Robots, 40(3):473–491, 2016.

O. Khatib.
A unified approach for motion and force control of robot manipulators: The operational space formulation.
IEEE Journal on Robotics and Automation, 3(1):43–53, feb 1987.

- M. Mistry, J. Buchli, and S. Schaal.
  Inverse dynamics control of floating base systems using orthogonal decomposition.
  2010 IEEE International Conference on Robotics and Automation, (3):3406–3412, may 2010.
- L. Righetti, J. Buchli, M. Mistry, M. Kalakrishnan, and S. Schaal.
  Optimal distribution of contact forces with inverse dynamics control.
  The International Journal of Robotics Research, (January), jan 2013.
- L. Righetti, J. Buchli, M. Mistry, and S. Schaal.
  Inverse dynamics control of floating-base robots with external constraints: A unified view.
  2011 IEEE International Conference on Robotics and Automation, pages 1085–1090, may 2011.

L. Saab, O. E. Ramos, N. Mansard, P. Soueres, and J.-y. Fourquet.
Dynamic Whole-Body Motion Generation under Rigid Contacts and other Unilateral Constraints.
IEEE Transactions on Robotics, 29(2):346–362, 2013.

L. Sentis and O. Khatib.
Synthesis of whole-body behaviors through hierarchical control of behavioral primitives.
International Journal of Humanoid Robotics, 2(4):505–518, 2005.

Historical Notes: Dynamically-Consistent Pseudo-Inverses

Early literature on TSID is rich of misleading claims, supported by convoluted math.

Historical Notes: Dynamically-Consistent Pseudo-Inverses

Early literature on TSID is rich of misleading claims, supported by convoluted math.

For instance, when using pseudo-inverses, it was believed that the only way to ensure consistency with dynamic was to use M^{-1} as weight matrix.

Historical Notes: Dynamically-Consistent Pseudo-Inverses

Early literature on TSID is rich of misleading claims, supported by convoluted math.

For instance, when using pseudo-inverses, it was believed that the only way to ensure consistency with dynamic was to use M^{-1} as weight matrix.

This has been shown not to be the case, but not everybody is aware of/agrees with this, so...beware!