Task-Space Inverse Dynamics: Implementation

Optimization-based Robot Control

Andrea Del Prete

University of Trento
Table of contents

1. Introduction

2. Details

3. Python Example

4. Exercises
Introduction
Key Concepts

Task

- Motion
- Force
- Actuation
Key Concepts

Task
- Motion
- Force
- Actuation

Rigid Contact
- similar to Task, but
- associated to reaction forces
Key Concepts

Task
- Motion
- Force
- Actuation

Rigid Contact
- similar to Task, but
- associated to reaction forces

Inverse Dynamics Formulation
- collects Tasks and RigidContacts
- translates them into HQP
Key Concepts

Task
- Motion
- Force
- Actuation

Inverse Dynamics Formulation
- collects Tasks and RigidContacts
- translates them into HQP

Rigid Contact
- similar to Task, but
- associated to reaction forces

HQP Solver
- solves a HQP
Other Concepts

**Constraint**

- affine function
- purely mathematical
- used to represent HQP
Other Concepts

**Constraint**

- affine function
- purely mathematical
- used to represent HQP

**Robot Wrapper**

- contains robot model
- provides utility functions to compute robot quantities
- e.g., mass matrix, Jacobians
Other Concepts

**Constraint**
- affine function
- purely mathematical
- used to represent HQP

**Robot Wrapper**
- contains robot model
- provides utility functions to compute robot quantities
- e.g., mass matrix, Jacobians

**Trajectory**
- maps time to vector values
- pos, vel, acc
- position and velocity can have different sizes (Lie groups)
Details
ConstraintBase

- A linear (affine) function
ConstraintBase

- A linear (affine) function
- Purely mathematical object
• A linear (affine) function
• Purely mathematical object
• “Unaware” of what the function represents

Three kinds of constraints:
Constraint Base

- A linear (affine) function
- Purely mathematical object
- “Unaware” of what the function represents

Three kinds of constraints:

- Equalities, represented by matrix $A$ and vector $a$:
  \[ A x = a \]
ConstraintBase

- A linear (affine) function
- Purely mathematical object
- “Unaware” of what the function represents

Three kinds of constraints:

- Equalities, represented by matrix $A$ and vector $a$:
  \[ Ax = a \]

- Inequalities, represented by matrix $A$ and vectors $lb$ and $ub$:
  \[ lb \leq Ax \leq ub \]
• A linear (affine) function
• Purely mathematical object
• “Unaware” of what the function represents

Three kinds of constraints:

• Equalities, represented by matrix $A$ and vector $a$:

$$Ax = a$$

• Inequalities, represented by matrix $A$ and vectors $lb$ and $ub$:

$$lb \leq Ax \leq ub$$

• Bounds, represented by vectors $lb$ and $ub$:

$$lb \leq x \leq ub$$
ConstraintBase

ConstraintBase(string name, int rows, int cols);

bool isEquality();
bool isInequality();
bool isBound();

Matrix matrix();
Vector vector();
Vector lowerBound();
Vector upperBound();

bool setMatrix(Matrix A);
bool setVector(Vector b);
bool setLowerBound(Vector lb);
bool setUpperBound(Vector ub);

bool checkConstraint(Vector x);
Interface of TaskBase:

TaskBase(string name, Model model);
Constraint compute(double t, Vector q, Vector v, Data data);
Interface of TaskBase:

TaskBase(string name, Model model);
Constraint compute(double t, Vector q, Vector v, Data data);

Three kinds of task:

- TaskMotion: linear function of robot accelerations
- TaskContactForce: linear function of contact forces
- TaskActuation: linear function of joint torques
TaskBase

Interface of TaskBase:

TaskBase(string name, Model model);
Constraint compute(double t, Vector q, Vector v, Data data);

Three kinds of task:

- TaskMotion: linear function of robot accelerations
- TaskContactForce: linear function of contact forces
- TaskActuation: linear function of joint torques

Tasks can compute either:

- equality constraints, e.g., TaskComEquality, TaskJointPosture, TaskSE3Equality
TaskBase

Interface of TaskBase:
TaskBase(string name, Model model);
Constraint compute(double t, Vector q, Vector v, Data data);

Three kinds of task:
- TaskMotion: linear function of robot accelerations
- TaskContactForce: linear function of contact forces
- TaskActuation: linear function of joint torques

Tasks can compute either:
- equality constraints, e.g., TaskComEquality, TaskJointPosture, TaskSE3Equality
- bounds, e.g., TaskJointBounds
Interface of TaskBase:

TaskBase(string name, Model model);
Constraint compute(double t, Vector q, Vector v, Data data);

Three kinds of task:

- TaskMotion: linear function of robot accelerations
- TaskContactForce: linear function of contact forces
- TaskActuation: linear function of joint torques

Tasks can compute either:

- equality constraints, e.g., TaskComEquality, TaskJointPosture, TaskSE3Equality
- bounds, e.g., TaskJointBounds
- inequality constraints, e.g., friction cones
ContactBase

Interface of ContactBase:

ContactBase(name, Kp, Kd, bodyName, regWeight);
ConstraintBase computeMotionTask(t, q, v, data);
InequalityConstraint computeForceTask(t, q, v, data);
ConstraintBase computeForceRegularizationTask(t, q, v, data);
Matrix computeForceGeneratorMatrix();

Motion task:
• represents motion constraint caused by rigid contact
• \( J \dot{v} = -\dot{J}v - Kp e - Kd \dot{e} \)

Force task:
• represents inequality constraints acting on contact forces
• e.g., friction cone constraints
• \( A_f \leq a \)
ContactBase

Interface of ContactBase:

ContactBase(name, Kp, Kd, bodyName, regWeight);
ConstraintBase computeMotionTask(t, q, v, data);
InequalityConstraint computeForceTask(t, q, v, data);
ConstraintBase computeForceRegularizationTask(t, q, v, data);
Matrix computeForceGeneratorMatrix();

Motion task:

- represents motion constraint caused by rigid contact
- $J\dot{v} = -\dot{J}v$
ContactBase

Interface of ContactBase:

ContactBase(name, Kp, Kd, bodyName, regWeight);
ConstraintBase computeMotionTask(t, q, v, data);
InequalityConstraint computeForceTask(t, q, v, data);
ConstraintBase computeForceRegularizationTask(t, q, v, data);
Matrix computeForceGeneratorMatrix();

Motion task:

• represents motion constraint caused by rigid contact
• \( J\dot{v} = -\dot{J}v - K_p e - K_d \dot{e} \)
Interface of ContactBase:

ContactBase(name, Kp, Kd, bodyName, regWeight);
ConstraintBase computeMotionTask(t, q, v, data);
InequalityConstraint computeForceTask(t, q, v, data);
ConstraintBase computeForceRegularizationTask(t, q, v, data);
Matrix computeForceGeneratorMatrix();

Motion task:

- represents motion constraint caused by rigid contact
- $J\dot{v} = -\dot{J}v - K_p e - K_d \dot{e}$

Force task:

- represents inequality constraints acting on contact forces
- e.g., friction cone constraints
- $Af \leq a$
Interface of ContactBase:

ContactBase(name, Kp, Kd, bodyName, regWeight);
ConstraintBase computeMotionTask(t, q, v, data);
InequalityConstraint computeForceTask(t, q, v, data);
ConstraintBase computeForceRegularizationTask(t, q, v, data);
Matrix computeForceGeneratorMatrix();

Force Regularization task:
- regularizes contact forces
- e.g., keep them close to friction cone center
ContactBase

Interface of ContactBase:

ContactBase(name, Kp, Kd, bodyName, regWeight);
ConstraintBase computeMotionTask(t, q, v, data);
InequalityConstraint computeForceTask(t, q, v, data);
ConstraintBase computeForceRegularizationTask(t, q, v, data);
Matrix computeForceGeneratorMatrix();

Force Regularization task:
• regularizes contact forces
• e.g., keep them close to friction cone center

Force-Generator matrix $T$:
• maps force variables to motion constraint representation
• Dynamic: $M\ddot{\nu} + h = S^\top \tau + J^\top Tf$
• Motion constraint: $J\dot{\nu} = -\dot{J}\nu$
• Friction cones: $Af \leq a$
Contact6d

- unilateral plane contact $\rightarrow$ 6d motion constraint
- minimal force representation $\rightarrow$ 6d (3d force + 3d moment)

PROBLEM

- hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
- easy to write friction constraints if force represented as collection of 3d forces applied at vertices of contact surface
- redundant representation, e.g., 4-vertex surface $\rightarrow$ 12 variables
- redundancy is an issue for motion constraint if solver does not handle redundant constraints (as \texttt{eiQuadProg}).

SOLUTION

- use 6d representation for motion constraint $J\dot{v} = -\dot{J}v \in \mathbb{R}^6$
- but 12d representation for force variable $f \in \mathbb{R}^{12}$
- force-generator matrix $T \in \mathbb{R}^{6 \times 12}$ defines mapping between two representations:
  
  $\tau_{\text{contact}} = J^\top Tf$
Contact6d

- unilateral plane contact $\rightarrow$ 6d motion constraint
- minimal force representation $\rightarrow$ 6d (3d force $+$ 3d moment)

**PROBLEM**
- hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
Contact6d

- unilateral plane contact $\rightarrow$ 6d motion constraint
- minimal force representation $\rightarrow$ 6d (3d force $+$ 3d moment)

**PROBLEM**

- hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
- easy to write friction constraints if force represented as collection of 3d forces applied at vertices of contact surface
Contact6d

- unilateral plane contact $\rightarrow$ 6d motion constraint
- minimal force representation $\rightarrow$ 6d (3d force + 3d moment)

**PROBLEM**
- hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
- easy to write friction constraints if force represented as collection of 3d forces applied at vertices of contact surface
  - redundant representation, e.g., 4-vertex surface $\rightarrow$ 12 variables

**SOLUTION**
- use 6d representation for motion constraint $\dot{J}v = -\dot{J}v \in \mathbb{R}^{6}$
- but 12d representation for force variable $f \in \mathbb{R}^{12}$
- force-generator matrix $T \in \mathbb{R}^{6 \times 12}$ defines mapping between two representations: $\tau_{\text{contact}} = J^\top T f$
Contact6d

- unilateral plane contact $\rightarrow$ 6d motion constraint
- minimal force representation $\rightarrow$ 6d (3d force $+$ 3d moment)

**PROBLEM**

- hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
- easy to write friction constraints if force represented as collection of 3d forces applied at vertices of contact surface
  
  - redundant representation, e.g., 4-vertex surface $\rightarrow$ 12 variables

- redundancy is an issue for motion constraint if solver does not handle redundant constraints (as eiQuadProg).
Contact6d

• unilateral plane contact → 6d motion constraint
• minimal force representation → 6d (3d force + 3d moment)

PROBLEM
• hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
• easy to write friction constraints if force represented as collection of 3d forces applied at vertices of contact surface
  • redundant representation, e.g., 4-vertex surface → 12 variables
• redundancy is an issue for motion constraint if solver does not handle redundant constraints (as eiQuadProg).

SOLUTION
• use 6d representation for motion constraint $\dot{J}\dot{v} = -\dot{J}v \in \mathbb{R}^6$
Contact 6d

- unilateral plane contact → 6d motion constraint
- minimal force representation → 6d (3d force + 3d moment)

**PROBLEM**

- hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
- easy to write friction constraints if force represented as collection of 3d forces applied at vertices of contact surface
  - redundant representation, e.g., 4-vertex surface → 12 variables
- redundancy is an issue for motion constraint if solver does not handle redundant constraints (as eiQuadProg).

**SOLUTION**

- use 6d representation for motion constraint $J\dot{v} = -\dot{J}v \in \mathbb{R}^6$
- but 12d representation for force variable $f \in \mathbb{R}^{12}$

- force-generator matrix $T \in \mathbb{R}^{6 \times 12}$ defines mapping between two representations: $\tau_{\text{contact}} = J^\top Tf$
Contact6d

- unilateral plane contact $\rightarrow$ 6d motion constraint
- minimal force representation $\rightarrow$ 6d (3d force + 3d moment)

PROBLEM
- hard to write friction constraints with 6d representation (especially for non-rectangular shapes)
- easy to write friction constraints if force represented as collection of 3d forces applied at vertices of contact surface
  - redundant representation, e.g., 4-vertex surface $\rightarrow$ 12 variables
- redundancy is an issue for motion constraint if solver does not handle redundant constraints (as eiQuadProg).

SOLUTION
- use 6d representation for motion constraint $J\dot{v} = -\dot{J}v \in \mathbb{R}^6$
- but 12d representation for force variable $f \in \mathbb{R}^{12}$
- force-generator matrix $T \in \mathbb{R}^{6 \times 12}$ defines mapping between two representations: $\tau_{contact} = J^\top Tf$
Central class of the whole library

Methods to add tasks:

- `addMotionTask(MotionTask task, double weight, int priority)`;
- `addForceTask(ForceTask task, double weight, int priority)`;
- `addTorqueTask(TorqueTask task, double weight, int priority)`;
Central class of the whole library

Methods to add tasks:

addMotionTask(MotionTask task, double weight, int priority);
addForceTask(ForceTask task, double weight, int priority);
addTorqueTask(TorqueTask task, double weight, int priority);

Method to add rigid contacts:

addRigidContact(RigidContact contact);
Central class of the whole library

Methods to add tasks:

addMotionTask(MotionTask task, double weight, int priority);
addForceTask(ForceTask task, double weight, int priority);
addTorqueTask(TorqueTask task, double weight, int priority);

Method to add rigid contacts:
addRigidContact(RigidContact contact);

Methods to convert TSID problem into (Hierarchical) QP:
HqpData computeProblemData(double time, Vector q, Vector v);
Central class of the whole library

Methods to add tasks:

addMotionTask(MotionTask task, double weight, int priority);
addForceTask(ForceTask task, double weight, int priority);
addTorqueTask(TorqueTask task, double weight, int priority);

Method to add rigid contacts:

addRigidContact(RigidContact contact);

Methods to convert TSID problem into (Hierarchical) QP:

HqpData computeProblemData(double time, Vector q, Vector v);

**HqpData defined as:**

```cpp
#define vector<pair<double, ConstraintBase>> ConstraintLevel
#define vector<ConstraintLevel> HqpData
```
Python Example
Ex 1 UR5

- Robot manipulator
• Robot manipulator
• end-effector control
• Robot manipulator
• end-effector control
• torque limits
- Robot manipulator
- end-effector control
- torque limits
- joint velocity limits
Description

- Code snippets
• Code snippets
• **Biped** robot with both feet on the ground (double support)
Description

- Code snippets
- **Biped** robot with both feet on the ground (double support)
- Control center of mass (CoM) for balance
• Code snippets
• **Biped** robot with both feet on the ground (double support)
• Control center of mass (**CoM**) for balance
• Control joint angles (**posture**) for whole-body stability
• Code snippets
• **Biped** robot with both feet on the ground (double support)
• Control center of mass (**CoM**) for balance
• Control joint angles (**posture**) for whole-body stability
• Good starting point before moving to more complex scenarios
Create Robot Wrapper

```python
import pinocchio as se3
from tsid import RobotWrapper, ...

...  
robot = RobotWrapper(urdf, vec, se3.JointModelFreeFlyer(), False)
```
invdyn = InverseDynamicsFormulationAccForce("tsid", robot, False)

q = ...
v = ...
invdyn.computeProblemData(t, q, v)
contactRF = Contact6d("contact_rfoot", robot, rf_frame_name, contact_points, contact_normal, mu, fMin, fMax, w_forceReg)

contactRF.setKp(...)
contactRF.setKd(...)

H_rf_ref = ...
contactRF.setReference(H_rf_ref)

invdyn.addRigidContact(contactRF)

# repeat for other contact(s)
Create Center-of-Mass Task

```python
comTask = TaskComEquality("task-com", robot)
comTask.setKp(...)  
comTask.setKd(...)  

invdyn.addMotionTask(comTask, w_com, 1, 0.0)
```
postureTask = TaskJointPosture("task-posture", robot)

postureTask.setKp(...)
postureTask.setKd(...)

invdyn.addMotionTask(postureTask, w_posture, 1, 0.0)
com_ref = robot.com(data)
trajCom = TrajectoryEuclidianConstant("traj_com", com_ref)

q_ref = q[7:]
trajPosture = TrajectoryEuclidianConstant("traj_joint", q_ref)
Create HQP Solver

```python
solver = SolverHQuadProg("qp solver")
solver.resize(invdyn.nVar, invdyn.nEq, invdyn.nIn)
```
Control Loop

for i in range(0, N_SIMULATION_STEPS):
    comTask.setReference(trajCom.computeNext())
    postureTask.setReference(trajPosture.computeNext())

    # get current state estimation
    (q, v) = ...

    HQPData = invdyn.computeProblemData(t, q, v)

    sol = solver.solve(HQPData)
    tau = invdyn.getActuatorForces(sol)

    # send desired joint torques (tau) to actuators
    ...

for i in range(0, N_SIMULATION_STEPS):
    ...

    # assuming perfect torque-acceleration tracking...
    dv = invdyn.getAccelerations(sol)

    # integrate desired accelerations
    q = se3.integrate(robot.model(), q, dt*v)
    v += dt*dv

    # increase time
    t += dt
Exercises
Exercise 2: CoM Sinusoidal Tracking

Run provided code (tsid/exercizes/ex_2.py) and check the sinusoidal reference CoM tracking
Exercise 2: CoM Sinusoidal Tracking

Run provided code (`tsid/exercizes/ex_2.py`) and check the sinusoidal reference CoM tracking

- Change CoM/posture gains and see effect

- Increase CoM frequency until tracking gets bad. Why does that happen?

- Set reference CoM outside support polygon (e.g., 20 cm to the side), what happens? Why?
Exercise 2: CoM Sinusoidal Tracking

Run provided code (tsid/exercises/ex_2.py) and check the sinusoidal reference CoM tracking

- Change CoM/posture gains and see effect
- Change CoM/posture weights and see effect
Run provided code (tsid/exercizes/ex_2.py) and check the sinusoidal reference CoM tracking

- Change CoM/posture gains and see effect
- Change CoM/posture weights and see effect
- Set reference CoM outside support polygon (e.g., 20 cm to the side), what happens? Why?
Exercise 2: CoM Sinusoidal Tracking

Run provided code (tsid/exercises/ex_2.py) and check the sinusoidal reference CoM tracking

- Change CoM/posture gains and see effect
- Change CoM/posture weights and see effect
- Set reference CoM outside support polygon (e.g., 20 cm to the side), what happens? Why?
- Increase CoM frequency until tracking gets bad. Why does that happen?
Exercise 2: CoM Sinusoidal Tracking

Run provided code (tsid/exercises/ex_2.py) and check the sinusoidal reference CoM tracking

- Change CoM/posture gains and see effect
- Change CoM/posture weights and see effect
- Set reference CoM outside support polygon (e.g., 20 cm to the side), what happens? Why?
- Increase CoM frequency until tracking gets bad. Why does that happen?
- Add contact on hand
Exercise 3: Balancing

Run provided code
(tsid/exercizes/ex_3_biped_balance_with_gui.py)
Run provided code
(tsid/exercises/ex_3_biped_balance_with_gui.py)

- Move reference CoM position
Exercise 3: Balancing

Run provided code
(tsid/exercises/ex_3_biped_balance_with_gui.py)

- Move reference CoM position
- Push robot and check reaction
Exercise 3: Balancing

Run provided code
(tsid/exercises/ex_3_biped_balance_with_gui.py)

- Move reference CoM position
- Push robot and check reaction
- Move CoM over left foot
Exercise 3: Balancing

Run provided code
(tsid/exercizes/ex_3_biped_balance_with_gui.py)

- Move reference CoM position
- Push robot and check reaction
- Move CoM over left foot
- Break contact with right foot
Exercise 3: Balancing

Run provided code
(tsid/exercises/ex_3_biped_balance_with_gui.py)

- Move reference CoM position
- Push robot and check reaction
- Move CoM over left foot
- Break contact with right foot
- Move reference right foot