Task-Space Inverse Dynamics: Implementation (Joint Space)

Optimization-based Robot Control

Andrea Del Prete

University of Trento

Table of contents

Introduction

Introduction

This document explains the implementation of the control framework Task-Space Inverse Dynamics (TSID).

[1] https://github.com/stack-of-tasks/tsid

Introduction

This document explains the implementation of the control framework Task-Space Inverse Dynamics (TSID).

To simplify the job we rely on the open-source C++ library TSID[1].

[1] https://github.com/stack-of-tasks/tsid

Introduction

This document explains the implementation of the control framework Task-Space Inverse Dynamics (TSID).

To simplify the job we rely on the open-source C++ library TSID[1].

TSID (currently) relies on:

- Eigen for linear algebra
- Pinocchio for multi-body dynamics computations
- Eiquadprog for solving Quadratic Programs

[1] https://github.com/stack-of-tasks/tsid

Main features: Pros & Cons

CONS

- Not mature (Feb 2017)
- Many missing features
  - Hierarchy
  - Joint pos limits
  - Bilateral contacts
  - Line contacts
  - ...

Main features: Pros & Cons

CONS

- Not mature (Feb 2017)
- Many missing features
  - Hierarchy
  - Joint pos limits
  - Bilateral contacts
  - Line contacts
  - ...

PROS

- Efficient ($<$0.6 ms for humanoid)
- Tested in simulation & on HRP-2
- Open source
- Modular design
  - $\rightarrow$ easy to extend
- Python bindings
- No alternative (AFAIK)

Key Concepts

Task

- JointPosture
- JointVelLimits
- JointTorqueLimits

Key Concepts

Task

- JointPosture
- JointVelLimits
- JointTorqueLimits

Robot Wrapper

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

Key Concepts

Task

- JointPosture
- JointVelLimits
- JointTorqueLimits

Robot Wrapper

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

Inverse Dynamics Formulation

- collects `Tasks` and ...
- translates them into LSP

Key Concepts

Task

- JointPosture
- JointVelLimits
- JointTorqueLimits

Robot Wrapper

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

Inverse Dynamics Formulation

- collects `Tasks` and ...
- translates them into LSP

HQP Solver

- solves HQP (LSP)

Details

Robot Wrapper

Interface for computing robot-related quantities:

```
RobotWrapper(string filename, vector<string> package_dirs,
             JointModelVariant rootJoint);

int nq(); // size of configuration vector q
int nv(); // size of velocity vector v

Model & model(); // reference to robot model (Pinocchio)

// Compute all quantities and store them into data
void computeAllTerms(Data &data, Vector q, Vector v);

Matrix mass(Data data);
Vector nonLinearEffects(Data data);
```

InverseDynamicsFormulationBase

Central class of the whole library

InverseDynamicsFormulationBase

Central class of the whole library

Method to add tasks:

```
addMotionTask(MotionTask task, double weight, int priority);
```

InverseDynamicsFormulationBase

Central class of the whole library

Method to add tasks:

```
addMotionTask(MotionTask task, double weight, int priority);
```

Method to convert TSID problem into (Hierarchical) QP:

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

InverseDynamicsFormulationBase

Central class of the whole library

Method to add tasks:

```
addMotionTask(MotionTask task, double weight, int priority);
```

Method to convert TSID problem into (Hierarchical) QP:

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

HqpData defined as:

```
#typedef vector<pair<double, ConstraintBase>> ConstraintLevel
#typedef vector<ConstraintLevel> HqpData
```

HQP Solvers

Using `InverseDynamicsFormulationBase` you get an `HqpData` object.

HQP Solvers

Using `InverseDynamicsFormulationBase` you get an `HqpData` object.

Then you need to solve this HQP.

HQP Solvers

Using `InverseDynamicsFormulationBase` you get an `HqpData` object.

Then you need to solve this HQP.

All HQP solvers implement this interface (`SolverHQPBase`):

```
void resize(int nVar, int nEq, int nIn);
HqpOutput solve(HqpData data);
```

HQP Solvers

Using `InverseDynamicsFormulationBase` you get an `HqpData` object.

Then you need to solve this HQP.

All HQP solvers implement this interface (`SolverHQPBase`):

```
void resize(int nVar, int nEq, int nIn);
HqpOutput solve(HqpData data);
```

`HqpOutput` is defined as:

```
class HqpOutput
{
    QpStatusFlag flag;
    Vector x, lambda;
}
```

Available HQP Solvers

- Several solvers currently implemented

Available HQP Solvers

- Several solvers currently implemented
- None of them supports hierarchy

Available HQP Solvers

- Several solvers currently implemented
- None of them supports hierarchy
- $\rightarrow$ HQP problems can only have two hierarchy levels.

Available HQP Solvers

- Several solvers currently implemented
- None of them supports hierarchy
- $\rightarrow$ HQP problems can only have two hierarchy levels.
- All solvers based on `EiQuadProg`: a modified version of `uQuadProg++` working with `Eigen`

Available HQP Solvers

- Several solvers currently implemented
- None of them supports hierarchy
- $\rightarrow$ HQP problems can only have two hierarchy levels.
- All solvers based on `EiQuadProg`: a modified version of `uQuadProg++` working with `Eigen`
- To improve efficiency, two optimized versions have been developed:

Available HQP Solvers

- Several solvers currently implemented
- None of them supports hierarchy
- $\rightarrow$ HQP problems can only have two hierarchy levels.
- All solvers based on `EiQuadProg`: a modified version of `uQuadProg++` working with `Eigen`
- To improve efficiency, two optimized versions have been developed:
  - `EiquadprogRealTime`: the fastest, but matrix sizes known at compile time

Available HQP Solvers

- Several solvers currently implemented
- None of them supports hierarchy
- $\rightarrow$ HQP problems can only have two hierarchy levels.
- All solvers based on `EiQuadProg`: a modified version of `uQuadProg++` working with `Eigen`
- To improve efficiency, two optimized versions have been developed:
  - `EiquadprogRealTime`: the fastest, but matrix sizes known at compile time
  - `EiquadprogFast`: dynamic matrix sizes (memory allocation performed only when resizing)

Available HQP Solvers

- Several solvers currently implemented
- None of them supports hierarchy
- $\rightarrow$ HQP problems can only have two hierarchy levels.
- All solvers based on `EiQuadProg`: a modified version of `uQuadProg++` working with `Eigen`
- To improve efficiency, two optimized versions have been developed:
  - `EiquadprogRealTime`: the fastest, but matrix sizes known at compile time
  - `EiquadprogFast`: dynamic matrix sizes (memory allocation performed only when resizing)

Results on HRP-2's computer (very old):

```
60 variables, 18 equalities, 40 inequalities
*** PROFILING RESULTS [ms]     (min - avg - max ) ***
Eiquadprog ..................... 0.651 0.704 0.870
Eiquadprog Fast ................ 0.563 0.605 0.810
Eiquadprog Real Time ......... 0.543 0.592 0.712

active inequalities .... 16.0 19.8 26.0
```

Exercises

Exercize 0

Open Virtual Machine.

Open Terminal and execute:

```
cd devel/src/tsid
git pull
spyder&
```

Open file
`/home/student/devel/src/tsid/exercizes/ex_0_ur5_joint_space_control.py`

Press F5 to run file.