Trajectory Optimization for Walking

Optimization-based Robot Control

Andrea Del Prete

University of Trento

Introduction

Task-Space Inverse Dynamics needs reference trajectories.

How to compute them for a walking robot?

Table of contents

Limits of Instantaneous Control

Limits of TSID

TSID is good for tracking trajectories, but...

Limits of TSID

TSID is good for tracking trajectories, but...

...no notion of future state!

Limits of TSID

TSID is good for tracking trajectories, but...

...no notion of future state!

Hard to anticipate constraint violations (e.g., joint limits).

Limits of TSID

TSID is good for tracking trajectories, but...

...no notion of future state!

Hard to anticipate constraint violations (e.g., joint limits).

Example of car moving towards wall.

Need for Trajectory Optimization

Trajectory Optimization $\approx$ TSID with preview horizon.

Need for Trajectory Optimization

Trajectory Optimization $\approx$ TSID with preview horizon.

PROS: Account for future constraints/cost in current decisions.

Need for Trajectory Optimization

Trajectory Optimization $\approx$ TSID with preview horizon.

PROS: Account for future constraints/cost in current decisions.

CONS: More computationally expensive.

Need for Trajectory Optimization

Trajectory Optimization $\approx$ TSID with preview horizon.

PROS: Account for future constraints/cost in current decisions.

CONS: More computationally expensive.

Solution

Use traj-opt offline to compute reference trajectory.

Use TSID online to track reference trajectory.

Trajectory Optimization through Contacts

Traj-opt for locomotion/manipulation is really hard!

Trajectory Optimization through Contacts

Traj-opt for locomotion/manipulation is really hard!

Option 1: Rigid Contacts

Hybrid dynamical system $\rightarrow$ Nonsmooth optimization problem!

Trajectory Optimization through Contacts

Traj-opt for locomotion/manipulation is really hard!

Option 1: Rigid Contacts

Hybrid dynamical system $\rightarrow$ Nonsmooth optimization problem!

Option 2: Soft (but stiff) Contacts

Stiff differential equations $\rightarrow$ Veeeery slow!

Trajectory Optimization through Contacts

Traj-opt for locomotion/manipulation is really hard!

Option 1: Rigid Contacts

Hybrid dynamical system $\rightarrow$ Nonsmooth optimization problem!

Option 2: Soft (but stiff) Contacts

Stiff differential equations $\rightarrow$ Veeeery slow!

Solution

Use rigid contacts, but fix contact sequence $\rightarrow$ Time-varying dynamical system (not hybrid!)

Trajectory Optimization for locomotion

Even with fixed contacts, traj-opt is hard because high-dimensional.

Trajectory Optimization for locomotion

Even with fixed contacts, traj-opt is hard because high-dimensional.

Use simplified models to speed it up.

Trajectory Optimization for locomotion

Even with fixed contacts, traj-opt is hard because high-dimensional.

Use simplified models to speed it up.

Potential use as Model Predictive Control.

Trajectory Optimization for locomotion

Even with fixed contacts, traj-opt is hard because high-dimensional.

Use simplified models to speed it up.

Potential use as Model Predictive Control.

Common models for locomotion:

- Inverted Pendulum
- Linear Inverted Pendulum
- Centroidal Dynamics (i.e. single rigid body dynamics)

Linear Inverted Pendulum Model (LIPM)

Center of Mass and Angular Momentum

Newton equation (center-of-mass dynamics):

$$m(\ddot{c} + g) = \sum_i f_i \tag{1}$$

Center of Mass and Angular Momentum

Newton equation (center-of-mass dynamics):

$$m(\ddot{c} + g) = \sum_i f_i \tag{1}$$

Euler equation (angular-momentum dynamics):

$$\dot{l} = \sum_i (p_i - c) \times f_i \tag{2}$$

Center of Mass and Angular Momentum

Newton equation (center-of-mass dynamics):

$$m(\ddot{c} + g) = \sum_i f_i \tag{1}$$

Euler equation (angular-momentum dynamics):

$$\dot{l} = \sum_i (p_i - c) \times f_i \tag{2}$$

where:

- c: center of mass (CoM)
- l: angular momentum (expressed at CoM)
- m: robot mass
- g: gravity acceleration
- f_i: i-th contact force
- p_i: i-th contact point

Flat Ground (1/2)

Assume:

- contacts with flat ground: $p_i^z = 0$
- constant angular momentum: $\dot{l} = 0$
- constant CoM height: $\dot{c}^z = \ddot{c}^z = 0$

Flat Ground (1/2)

Assume:

- contacts with flat ground: $p_i^z = 0$
- constant angular momentum: $\dot{l} = 0$
- constant CoM height: $\dot{c}^z = \ddot{c}^z = 0$

Then we get (Wieber, Tedrake, and Kuindersma 2015):

$$c^{xy} - \frac{c^z}{g^z}\ddot{c}^{xy} = \underbrace{\frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z}}_{\text{Center of Pressure}} \tag{3}$$

Flat Ground (1/2)

Assume:

- contacts with flat ground: $p_i^z = 0$
- constant angular momentum: $\dot{l} = 0$
- constant CoM height: $\dot{c}^z = \ddot{c}^z = 0$

Then we get (Wieber, Tedrake, and Kuindersma 2015):

$$c^{xy} - \frac{c^z}{g^z}\ddot{c}^{xy} = \underbrace{\frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z}}_{\text{Center of Pressure}} \tag{3}$$

$f_i^z \geq 0$

Flat Ground (1/2)

Assume:

- contacts with flat ground: $p_i^z = 0$
- constant angular momentum: $\dot{l} = 0$
- constant CoM height: $\dot{c}^z = \ddot{c}^z = 0$

Then we get (Wieber, Tedrake, and Kuindersma 2015):

$$c^{xy} - \frac{c^z}{g^z}\ddot{c}^{xy} = \underbrace{\frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z}}_{\text{Center of Pressure}} \tag{3}$$

$f_i^z \geq 0 \impliedby z^{xy} \triangleq \frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z} \in \text{conv}(p_i^{xy})$

Flat Ground (2/2)

Rearrange (3) as:

$$\ddot{c}^{xy} = \frac{g^z}{c^z}(c^{xy} - z^{xy}) \tag{4}$$

Flat Ground (2/2)

Rearrange (3) as:

$$\ddot{c}^{xy} = \frac{g^z}{c^z}(c^{xy} - z^{xy}) \tag{4}$$

Interpretation

CoM acc $\ddot{c}^{xy}$ given by force pushing CoM c^{xy} away from CoP z^{xy}

Flat Ground (2/2)

Rearrange (3) as:

$$\ddot{c}^{xy} = \frac{g^z}{c^z}(c^{xy} - z^{xy}) \tag{4}$$

Interpretation

CoM acc $\ddot{c}^{xy}$ given by force pushing CoM c^{xy} away from CoP z^{xy} $\rightarrow$ UNSTABLE!

Flat Ground (2/2)

Rearrange (3) as:

$$\ddot{c}^{xy} = \frac{g^z}{c^z}(c^{xy} - z^{xy}) \tag{4}$$

Interpretation

CoM acc $\ddot{c}^{xy}$ given by force pushing CoM c^{xy} away from CoP z^{xy} $\rightarrow$ UNSTABLE!

Same dynamics as linearized Inverted Pendulum.

LIPM as Linear Dynamical System

Rewrite (3) as:

$$\underbrace{\begin{bmatrix} \dot{c}^{xy} \\ \ddot{c}^{xy} \end{bmatrix}}_{\dot{x}} = \begin{bmatrix} 0 & I \\ \omega^2 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} c^{xy} \\ \dot{c}^{xy} \end{bmatrix}}_{x} + \begin{bmatrix} 0 \\ -\omega^2 \end{bmatrix} \underbrace{z^{xy}}_{u} \tag{5}$$

where $\omega^2 \triangleq \frac{g^z}{c^z}$.

LIPM as Linear Dynamical System

Rewrite (3) as:

$$\underbrace{\begin{bmatrix} \dot{c}^{xy} \\ \ddot{c}^{xy} \end{bmatrix}}_{\dot{x}} = \begin{bmatrix} 0 & I \\ \omega^2 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} c^{xy} \\ \dot{c}^{xy} \end{bmatrix}}_{x} + \begin{bmatrix} 0 \\ -\omega^2 \end{bmatrix} \underbrace{z^{xy}}_{u} \tag{5}$$

where $\omega^2 \triangleq \frac{g^z}{c^z}$.

Discretize with time step δt:

$$x^+ = \underbrace{\begin{bmatrix} \cosh(\omega \delta t) & \omega^{-1} \sinh(\omega \delta t) \\ \omega \sinh(\omega \delta t) & \cosh(\omega \delta t) \end{bmatrix}}_{A} x + \underbrace{\begin{bmatrix} 1 - \cosh(\omega \delta t) \\ -\omega \sinh(\omega \delta t) \end{bmatrix}}_{B} u \tag{6}$$

Center of Mass Trajectory Optimization with LIPM

Key Idea

Follow reference trajectory of:

- CoP $P = \begin{bmatrix} p_0 & \dots & p_{N-1} \end{bmatrix}$ (i.e. foot steps),
- CoM position $C^{ref} = \begin{bmatrix} c_0^{ref} & \dots & c_N^{ref} \end{bmatrix}$
- CoM velocity $\dot{C}^{ref} = \begin{bmatrix} \dot{c}_0^{ref} & \dots & \dot{c}_N^{ref} \end{bmatrix}$

Key Idea

Follow reference trajectory of:

- CoP $P = \begin{bmatrix} p_0 & \dots & p_{N-1} \end{bmatrix}$ (i.e. foot steps),
- CoM position $C^{ref} = \begin{bmatrix} c_0^{ref} & \dots & c_N^{ref} \end{bmatrix}$
- CoM velocity $\dot{C}^{ref} = \begin{bmatrix} \dot{c}_0^{ref} & \dots & \dot{c}_N^{ref} \end{bmatrix}$

Foot steps and timing P predefined by user.

Key Idea

Follow reference trajectory of:

- CoP $P = \begin{bmatrix} p_0 & \dots & p_{N-1} \end{bmatrix}$ (i.e. foot steps),
- CoM position $C^{ref} = \begin{bmatrix} c_0^{ref} & \dots & c_N^{ref} \end{bmatrix}$
- CoM velocity $\dot{C}^{ref} = \begin{bmatrix} \dot{c}_0^{ref} & \dots & \dot{c}_N^{ref} \end{bmatrix}$

Foot steps and timing P predefined by user.

Keep CoP close to foot center for robustness.

Key Idea

Follow reference trajectory of:

- CoP $P = \begin{bmatrix} p_0 & \dots & p_{N-1} \end{bmatrix}$ (i.e. foot steps),
- CoM position $C^{ref} = \begin{bmatrix} c_0^{ref} & \dots & c_N^{ref} \end{bmatrix}$
- CoM velocity $\dot{C}^{ref} = \begin{bmatrix} \dot{c}_0^{ref} & \dots & \dot{c}_N^{ref} \end{bmatrix}$

Foot steps and timing P predefined by user.

Keep CoP close to foot center for robustness.

C^{ref} could be straight line.

Formulation

$$
\begin{aligned}
\underset{C,\dot{C},U}{\text{minimize}} \quad & \sum_k \frac{\beta}{2}||c_k - c_k^{ref}||^2 + \frac{\gamma}{2}||\dot{c} - \dot{c}^{ref}||^2 + \qquad \frac{\alpha}{2}||u_k - p_k||^2 \\
\text{subject to} \quad & p_k - \frac{s}{2} \leq u_k \leq p_k + \frac{s}{2} \qquad\qquad k = 0 \dots N-1 \\
& x_{k+1} = Ax_k + Bu_k \qquad\qquad k = 0 \dots N-1 \\
& x_0 = x_{initial} \\
& x_N = x_{final}
\end{aligned} \tag{7}
$$

where:

- $s \in \mathbb{R}^2$ = foot size in x and y directions
- $C = \begin{bmatrix} c_0 & \dots & c_N \end{bmatrix}$
- $\dot{C} = \begin{bmatrix} \dot{c}_0 & \dots & \dot{c}_N \end{bmatrix}$
- $x_k = (c_k, \dot{c}_k)$
- α, β, γ = user-defined weights

Quadratic Program

Problem (7) can be expressed as QP:

$$\begin{aligned}
\min_{U} \quad & \frac{1}{2}U^\top QU + g^\top U \\
\text{subject to} \quad & A_{in}U \leq b_{in} \\
& A_{eq}U = b_{eq}
\end{aligned} \tag{8}$$

Quadratic Program

Problem (7) can be expressed as QP:

$$\begin{aligned} \min_{U} \quad & \frac{1}{2} U^\top Q U + g^\top U \\ \text{subject to} \quad & A_{in} U \leq b_{in} \\ & A_{eq} U = b_{eq} \end{aligned} \tag{8}$$

where we express $C, \dot{C}$ as functions of U (*shooting*):

$$\begin{aligned} C &= P_{ps} c_0 + P_{pu} U \\ \dot{C} &= P_{vs} \dot{c}_0 + P_{vu} U \end{aligned} \tag{9}$$

Foot-step Planning

Key Idea

Optimize for foot step positions, but...

Key Idea

Optimize for foot step positions, but...

...foot step timing remains fixed.

Key Idea

Optimize for foot step positions, but...

...foot step timing remains fixed.

Add P to decision variables $\rightarrow$ Problem remains QP! (Herdt et al. 2010)

Key Idea

Optimize for foot step positions, but...

...foot step timing remains fixed.

Add P to decision variables $\rightarrow$ Problem remains QP! (Herdt et al. 2010)

Bound distance between successive foot steps.

CoM Trajectory Optimization with Foot-Step Planning

$$
\begin{aligned}
\underset{C,\dot{C},U,P}{\text{minimize}} \quad & \sum_k \frac{\beta}{2}||c_k - c_k^{ref}||^2 + \frac{\gamma}{2}||\dot{c} - \dot{c}^{ref}||^2 + \quad \frac{\alpha}{2}||u_k - p_k||^2 \\
\text{subject to} \quad & p_k - \frac{s}{2} \leq u_k \leq p_k + \frac{s}{2} && k = 0 \dots N-1 \\
& x_{k+1} = Ax_k + Bu_k && k = 0 \dots N-1 \\
& x_0 = x_{initial} \\
& x_N = x_{final} \\
& p_{k+1} - p_k \in \mathcal{P}_k && k = 0 \dots N-1
\end{aligned} \tag{10}
$$

Implementation in Python (exploiting existing library)

Library

Open-source Python library:
`https://github.com/machines-in-motion/lmpc_walking`.

Main developer: Ahmad Gazar (currently PhD student at Max-Planck Institute).

Code

```
# Inverted pendulum parameters:
# ----------------------------
foot_length = conf.lxn + conf.lxp   # foot size in x direction
foot_width  = conf.lyn + conf.lyp   # foot size in y direciton
nb_dt_per_step        = int(conf.T_step/conf.dt_mpc)
N  = conf.nb_steps * nb_dt_per_step  # nb of time steps
```

Code

```
# CoM initial state: [x_0, xdot_0].T
#                    [y_0, ydot_0].T
# ----------------------------------
x_0 = np.array([conf.foot_step_0[0], 0.0])
y_0 = np.array([conf.foot_step_0[1], 0.0])

step_width = 2*np.absolute(y_0[0])
```

Code

```
# compute foot steps
foot_steps  = manual_foot_placement(conf.foot_step_0,
                                    conf.step_length, conf.nb_steps)
foot_steps[1:,0] -= conf.step_length
```

Code

```
# compute foot steps
foot_steps  = manual_foot_placement(conf.foot_step_0,
                                    conf.step_length, conf.nb_steps)
foot_steps[1:,0] -= conf.step_length
```

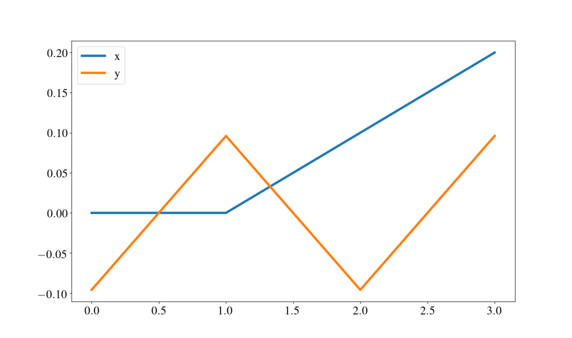

Figure 1: Foot steps.

Code

```
# compute CoP reference trajectory:
cop_ref = create_CoP_trajectory(conf.nb_steps,
                        foot_steps, N, nb_dt_per_step)
```

Code

```
# compute CoP reference trajectory:
cop_ref = create_CoP_trajectory(conf.nb_steps,
                                foot_steps, N, nb_dt_per_step)
```

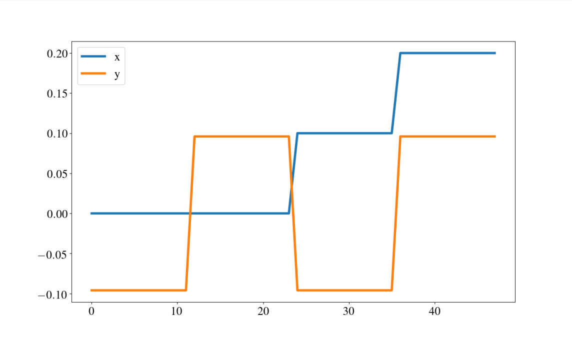

Figure 2: Foot steps and CoP.

Code

```
# terminal constraints
x_terminal = np.array([cop_ref[N-1, 0], 0.0])
y_terminal = np.array([cop_ref[N-1, 1], 0.0])
nb_terminal_constraints = 4
terminal_index = N-1
```

Code

```
# construct preview system
#  C_{1:N}     = P_ps c_0     + P_pu U
#  C^dot_{1:N} = P_vs c^dot_0 + P_vu U
[P_ps, P_vs, P_pu, P_vu] = compute_recursive_matrices(
                                        conf.dt_mpc, conf.g, conf.h, N)
```

Code

```
# construct preview system
#  C_{1:N}     = P_ps c_0     + P_pu U
#  C^dot_{1:N} = P_vs c^dot_0 + P_vu U
[P_ps, P_vs, P_pu, P_vu] = compute_recursive_matrices(
                                conf.dt_mpc, conf.g, conf.h, N)
```

Beware of condition number:

```
>>> np.log10(np.max(np.abs(P_ps)) / np.min(np.abs(P_ps)))
9.2
```

Code

```
# compute cost function terms
[Q, p_k] = compute_objective_terms(conf.alpha, conf.beta,
                   conf.gamma, conf.T_step, nb_dt_per_step,
                   N, conf.step_length, step_width,
                   P_ps, P_pu, P_vs, P_vu, x_0, y_0, cop_ref)
```

Code

```
# compute cost function terms
[Q, p_k] = compute_objective_terms(conf.alpha, conf.beta,
                     conf.gamma, conf.T_step, nb_dt_per_step,
                     N, conf.step_length, step_width,
                     P_ps, P_pu, P_vs, P_vu, x_0, y_0, cop_ref)
```

Beware of condition number of Q:

```
>>> np.max(np.abs(Q))
1.2e+16
>>> np.min(np.abs(Q))
0.0
```

Code

```
# create CoP (ZMP) and terminal constraints
[A_zmp, b_zmp] = add_ZMP_constraints(N, foot_length, foot_width,
                                     cop_ref)
[A_terminal, b_terminal] = add_terminal_constraints(N,
                           terminal_index, x_0, y_0, x_terminal,
                            y_terminal, P_ps, P_vs, P_pu, P_vu)
A = np.concatenate((A_terminal, A_zmp), axis = 0)
b = np.concatenate((b_terminal, b_zmp), axis = 0)
```

Code

```
# call QP solver:
U = solve_qp(Q, -p_k, A.T, b, nb_terminal_constraints)[0]
cop_x = U[0:N]
cop_y = U[N:2*N]

# Compute CoM trajectory from CoP
[com_state_x, com_state_y] = compute_recursive_dynamics(P_ps,
                                       P_vs, P_pu, P_vu, N,
                                       x_0, y_0, U)
```

Run script

Update code before running scripts:

```
cd devel/src/tsid/exercizes
git pull
python ex_4_plan_LIPM_romeo.py
```

Connection with TSID

LIPM to Whole-Body Model

Two issues:

1. Different time steps
2. Foot trajectories

Interpolation

Input: CoM (pos, vel) and CoP trajectories with traj-opt (large) time step.

Output: CoM (pos, vel, acc) with control (small) time step.

Interpolation

Input: CoM (pos, vel) and CoP trajectories with traj-opt (large) time step.

Output: CoM (pos, vel, acc) with control (small) time step.

Compute pos-vel with:

$$\begin{bmatrix} c \\ \dot{c} \end{bmatrix}^{+} = \begin{bmatrix} \cosh(\omega\delta t) & \omega^{-1}\sinh(\omega\delta t) \\ \omega\sinh(\omega\delta t) & \cosh(\omega\delta t) \end{bmatrix} \begin{bmatrix} c \\ \dot{c} \end{bmatrix} + \begin{bmatrix} 1 - \cosh(\omega\delta t) \\ -\omega\sinh(\omega\delta t) \end{bmatrix} u \quad (11)$$

Interpolation

Input: CoM (pos, vel) and CoP trajectories with traj-opt (large) time step.

Output: CoM (pos, vel, acc) with control (small) time step.

Compute pos-vel with:

$$\begin{bmatrix} c \\ \dot{c} \end{bmatrix}^{+} = \begin{bmatrix} \cosh(\omega\delta t) & \omega^{-1}\sinh(\omega\delta t) \\ \omega\sinh(\omega\delta t) & \cosh(\omega\delta t) \end{bmatrix} \begin{bmatrix} c \\ \dot{c} \end{bmatrix} + \begin{bmatrix} 1 - \cosh(\omega\delta t) \\ -\omega\sinh(\omega\delta t) \end{bmatrix} u \quad (11)$$

Compute acc with:

$$\ddot{c} = \frac{g^z}{c^z}(c - z) \quad (12)$$

Foot Trajectories

Common choice: polynomials.

Foot Trajectories

Common choice: polynomials.

For instance: 3rd order with constraints:

- initial pos
- initial vel (zero)
- final pos
- final vel (zero)

Foot Trajectories

Common choice: polynomials.

For instance: 3rd order with constraints:

- initial pos
- initial vel (zero)
- final pos
- final vel (zero)

Use higher order if you wanna add constraints (e.g., zero initial/final acc).

Run script

Update code and run scripts:

```
cd devel/src/tsid/exercizes
git pull
python ex_4_LIPM_to_TSID.py
python ex_4_walking.py
```

References

Herdt, Andrei et al. (2010). "Online Walking Motion Generation with Automatic Foot Step Placement". In: *Advanced Robotics* 24.5-6.

Wieber, Pierre-Brice, Russ Tedrake, and Scott Kuindersma (2015). "Modeling and Control of Legged Robots". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Khatib Oussama. 2nd. Chap. 48.