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>
<thead>
<tr>
<th>1. Limits of Instantaneous Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Linear Inverted Pendulum Model (LIPM)</td>
</tr>
<tr>
<td>3. Center of Mass Trajectory Optimization with LIPM</td>
</tr>
<tr>
<td>4. Foot-step Planning</td>
</tr>
<tr>
<td>5. Implementation in Python (exploiting existing library)</td>
</tr>
<tr>
<td>6. Connection with TSID</td>
</tr>
</tbody>
</table>
Limits of Instantaneous Control
Limits of TSID

TSID is good for tracking trajectories, but...
TSID is good for tracking trajectories, but...

...no notion of future state!
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 → Nonsmooth optimization problem!
Trajectory Optimization through Contacts

Traj-opt for locomotion/manipulation is really hard!

**Option 1: Rigid Contacts**
Hybrid dynamical system → Nonsmooth optimization problem!

**Option 2: Soft (but stiff) Contacts**
Stiff differential equations → 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!)
Even with fixed contacts, traj-opt is hard because high-dimensional.
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.
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)
Newton equation (center-of-mass dynamics):

\[ m(\ddot{c} + g) = \sum_i f_i \]  

(1)

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
Newton equation (center-of-mass dynamics):

\[ m(\ddot{c} + g) = \sum_i f_i \]  \hspace{1cm} (1)

Euler equation (angular-momentum dynamics):

\[ \dot{i} = \sum_i (p_i - c) \times f_i \]  \hspace{1cm} (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
Center of Mass and Angular Momentum

Newton equation (center-of-mass dynamics):
\[ m(\ddot{c} + g) = \sum_i f_i \]  (1)

Euler equation (angular-momentum dynamics):
\[ \dot{i} = \sum_i (p_i - c) \times f_i \]  (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: \( \dot{i} = 0 \)
- constant CoM height: \( \dot{c}^z = \ddot{c}^z = 0 \)
Assume:

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

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

\[
c^{xy} - \frac{c^z}{g^z} \ddot{c}^{xy} = \frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z} \]

Center of Pressure

(3)
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):

\[
c^{xy} - \frac{c^z}{g^z} \dot{c}^{xy} = \frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z}
\]

\[
f_i^z \geq 0
\]
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):

$$c^{xy} - \frac{c^z}{g^z} \dot{c}^{xy} = \frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z}$$

$$f_i^z \geq 0 \iff z^{xy} \triangleq \frac{\sum_i f_i^z p_i^{xy}}{\sum_i f_i^z} \in \text{conv}(p_i^{xy})$$
Rearrange (3) as:

\[ \ddot{c}^{xy} = \frac{g^z}{C^z} (c^{xy} - z^{xy}) \]  

(4)
Rearrange (3) as:

\[ \ddot{c}^{xy} = \frac{g^z}{C^z}(c^{xy} - z^{xy}) \]  

(4)

**Interpretation**

CoM acc \( \ddot{c}^{xy} \) given by force pushing CoM \( c^{xy} \) away from CoP \( z^{xy} \)
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 \(z_{xy}\) → **UNSTABLE!**
Rearrange (3) as:

\[ \ddot{c}_{xy} = \frac{g^z}{c^z} (c_{xy} - z_{xy}) \] (4)

**Interpretation**

CoM acc \( \ddot{c}_{xy} \) given by force pushing CoM \( c_{xy} \) away from CoP \( z_{xy} \) → **UNSTABLE!**

Same dynamics as linearized Inverted Pendulum.
LIPM as Linear Dynamical System

Rewrite (3) as:

\[
\begin{bmatrix}
\dot{c}_{xy} \\
\ddot{c}_{xy}
\end{bmatrix}
= \begin{bmatrix}
0 & I \\
\omega^2 & 0
\end{bmatrix}
\begin{bmatrix}
c_{xy} \\
\dot{c}_{xy}
\end{bmatrix}
+ \begin{bmatrix}
0 \\
-\omega^2
\end{bmatrix} z_{xy}
\]

(5)

where \( \omega^2 \triangleq \frac{g^z}{c^z} \).
Rewrite (3) as:

\[
\begin{bmatrix}
\dot{c}_{xy} \\
\ddot{c}_{xy}
\end{bmatrix} =
\begin{bmatrix}
0 & I \\
\omega^2 & 0
\end{bmatrix}
\begin{bmatrix}
c_{xy} \\
\dot{c}_{xy}
\end{bmatrix} +
\begin{bmatrix}
0 \\
-\omega^2
\end{bmatrix} z_{xy} + u
\]

(5)

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

Discretize with time step \( \delta t \):

\[
x^+ =
\begin{bmatrix}
\cosh(\omega \delta t) & \omega^{-1} \sinh(\omega \delta t) \\
\omega \sinh(\omega \delta t) & \cosh(\omega \delta t)
\end{bmatrix}
x +
\begin{bmatrix}
1 - \cosh(\omega \delta t) \\
-\omega \sinh(\omega \delta t)
\end{bmatrix} u
\]

(6)
Center of Mass Trajectory Optimization with LIPM
Key Idea

Follow reference trajectory of:

- CoP \( P = \begin{bmatrix} p_0 & \ldots & p_{N-1} \end{bmatrix} \) (i.e. foot steps),
- CoM position \( C^{\text{ref}} = \begin{bmatrix} c_0^{\text{ref}} & \ldots & c_N^{\text{ref}} \end{bmatrix} \)
- CoM velocity \( \dot{C}^{\text{ref}} = \begin{bmatrix} \dot{c}_0^{\text{ref}} & \ldots & \dot{c}_N^{\text{ref}} \end{bmatrix} \)
Follow reference trajectory of:

- CoP $P = [p_0 \ldots p_{N-1}]$ (i.e. foot steps),
- CoM position $C^{ref} = [c_{0}^{ref} \ldots c_{N}^{ref}]$
- CoM velocity $\dot{C}^{ref} = [\dot{c}_{0}^{ref} \ldots \dot{c}_{N}^{ref}]$

Foot steps and timing $P$ predefined by user.
Key Idea

Follow reference trajectory of:

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

Foot steps and timing $P$ predefined by user.

Keep CoP close to foot center for robustness.
Follow reference trajectory of:

- **CoP** $P = [p_0 \ldots p_{N-1}]$ (i.e. foot steps),
- **CoM position** $C^{ref} = [c^{ref}_0 \ldots c^{ref}_N]$
- **CoM velocity** $\dot{C}^{ref} = [\dot{c}^{ref}_0 \ldots \dot{c}^{ref}_N]$

Foot steps and timing $P$ predefined by user.

Keep CoP close to foot center for robustness.

$C^{ref}$ could be straight line.
Formulation

\[
\begin{align*}
\text{minimize} & \quad \sum_k \frac{\beta}{2} \| c_k - c_k^{\text{ref}} \|^2 + \frac{\gamma}{2} \| \dot{c} - \dot{c}^{\text{ref}} \|^2 + \frac{\alpha}{2} \| u_k - p_k \|^2 \\
\text{subject to} & \quad p_k - \frac{s}{2} \leq u_k \leq p_k + \frac{s}{2} \quad \forall k = 0 \ldots N - 1 \\
& \quad x_{k+1} = Ax_k + Bu_k \quad \forall k = 0 \ldots N - 1 \\
& \quad x_0 = x_{\text{initial}} \\
& \quad x_N = x_{\text{final}}
\end{align*}
\]

where:

- \( s \in \mathbb{R}^2 \) = foot size in \( x \) and \( y \) directions
- \( C = \begin{bmatrix} c_0 & \ldots & c_N \end{bmatrix} \)
- \( \dot{C} = \begin{bmatrix} \dot{c}_0 & \ldots & \dot{c}_N \end{bmatrix} \)
- \( x_k = (c_k, \dot{c}_k) \)
- \( \alpha, \beta, \gamma \) = user-defined weights
Problem (7) can be expressed as QP:

\[
\min_U \quad \frac{1}{2} U^\top Q U + g^\top U \\
\text{subject to} \quad A_{in} U \leq b_{in} \\
A_{eq} U = b_{eq}
\]
Problem (7) can be expressed as QP:

\[
\min_U \quad \frac{1}{2} U^\top QU + g^\top U
\]

subject to \quad A_{in} U \leq b_{in}

\[A_{eq} U = b_{eq}\]

(8)

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

\[
C = P_{ps} c_0 + P_{pu} U
\]

\[
\dot{C} = P_{vs} \dot{c}_0 + P_{vu} U
\]

(9)
Foot-step Planning
Optimize for foot step positions, but...
Optimize for foot step positions, but...

...foot step timing remains fixed.
Optimize for foot step positions, but...

...foot step timing remains fixed.

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

\[
\begin{align*}
\text{minimize} & \quad \sum_k \frac{\beta}{2} \left\| c_k - c_k^{ref} \right\|^2 + \frac{\gamma}{2} \left\| \dot{c} - \dot{c}^{ref} \right\|^2 + \frac{\alpha}{2} \left\| u_k - p_k \right\|^2 \\
\text{subject to} & \quad p_k - \frac{s}{2} \leq u_k \leq p_k + \frac{s}{2} \\
& \quad x_{k+1} = Ax_k + Bu_k \quad k = 0 \ldots N - 1 \\
& \quad x_0 = x_{initial} \\
& \quad x_N = x_{final} \\
& \quad p_{k+1} - p_k \in \mathcal{P}_k \quad k = 0 \ldots N - 1 \\
\end{align*}
\]
Implementation in Python (exploiting existing library)
Open-source Python library:

Main developer: Ahmad Gazar (currently PhD student at Max-Planck Institute).
# Inverted pendulum parameters:
# ----------------------------
foot_length = conf.lxn + conf.lxp  # foot size in x direction
foot_width = conf.lyn + conf.lyp  # foot size in y direction
nb_dt_per_step = int(conf.T_step/conf.dt_mpc)
N = conf.nb_steps * nb_dt_per_step  # nb of time steps
# CoM initial state: \([x_0, xdot_0].T\)
#
# \([y_0, ydot_0].T\)
#
# ----------------------------------

```python
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])
```
# 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
# 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.
# compute CoP reference trajectory:
cop_ref = create_CoP_trajectory(conf.nb_steps,
    foot_steps, N, nb_dt_per_step)
# 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.
# 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
# construct preview system
# \( C_{1:N} = P_{ps} c_0 + P_{pu} U \)
# \( C^\dot{}_{1:N} = P_{vs} c^\dot{}_0 + P_{vu} U \)

\[
[P_{ps}, P_{vs}, P_{pu}, P_{vu}] = \text{compute_recursive_matrices}(
    \text{conf.dt_mpc, conf.g, conf.h, N})
\]
# construct preview system
# \( C_{1:N} = P_ps \, c_0 + P_{pu} \, U \)
# \( C^{\dot{}}_{1:N} = P_{vs} \, c^{\dot{0}} + P_{vu} \, U \)

\[
[P_ps, P_vs, P_{pu}, P_{vu}] = \text{compute_recursive_matrices}(
    \text{conf.dt_mpc, conf.g, conf.h, N})
\]

Beware of condition number:

```python
>>> np.log10(np.max(np.abs(P_ps)) / np.min(np.abs(P_ps)))
9.2
```
# compute cost function terms

\[ [Q, p_k] = \text{compute_objective_terms}(\text{conf}.\alpha, \text{conf}.\beta, \text{conf}.\gamma, \text{conf}.T_\text{step}, \text{nb}_\text{dt}_\text{per}_\text{step}, N, \text{conf}.\text{step}_\text{length}, \text{step}_\text{width}, P_{ps}, P_{pu}, P_{vs}, P_{vu}, x_0, y_0, \text{cop}_\text{ref}) \]
# compute cost function terms

```python
[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 $Q$:

```python
>>> np.max(np.abs(Q))
1.2e+16
>>> np.min(np.abs(Q))
0.0
```
# 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, PPu, P_vu)

A = np.concatenate((A_terminal, A_zmp), axis = 0)
b = np.concatenate((b_terminal, b_zmp), axis = 0)
# 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)
Update code before running scripts:

cd devel/src/tsid/exercises

git pull

python ex_4_plan_LIPM_romeo.py
Connection with TSID
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.

\[
\begin{align*}
\dot{c} + & \begin{bmatrix}
\cosh(\omega \delta t) \\
\omega - 1 \\
\omega \sinh(\omega \delta t) \\
\cosh(\omega \delta t)
\end{bmatrix} \\
& + \begin{bmatrix}
1 - \cosh(\omega \delta t) - \omega \sinh(\omega \delta t)
\end{bmatrix} u
\end{align*}
\]

\[\ddot{c} = g_z c_z (c - z) \]

**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:

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

Compute acc with:

\[\dddot{c} = g_z c_z (c - z) \quad (12)\]
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:

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

Compute acc with:

\[
\dddot{c} = \frac{g^z}{c^z} (c - z)
\] (12)
Common choice: polynomials.
Foot Trajectories

Common choice: polynomials.

For instance: 3rd order with constraints:

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