Underactuated Robots: Humanoid Robot Control

Nicola Scianca

December 2025

humanoid robot control architectures

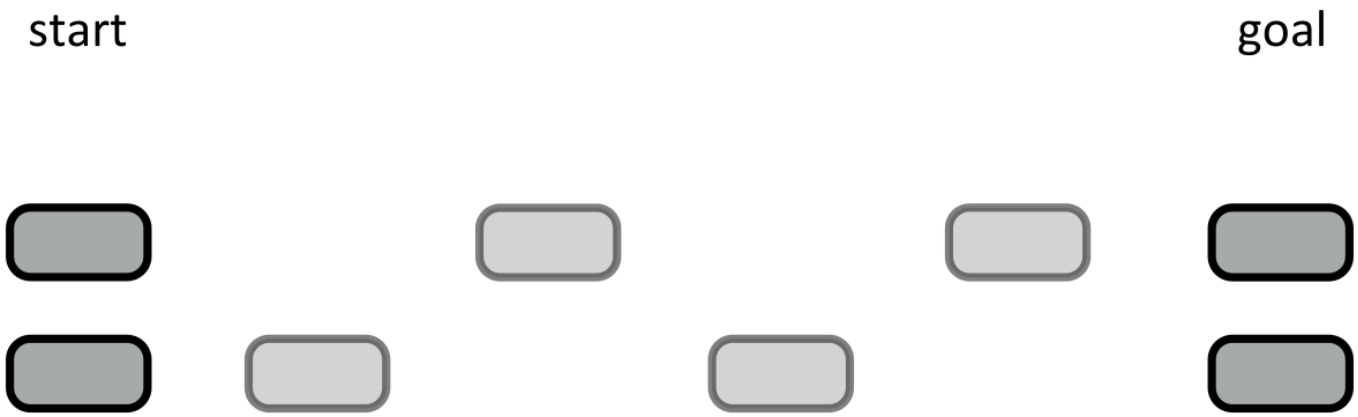
- in this lecture we will focus on **hierarchical** architectures based on **model predictive control** (MPC)
- MPC is used to optimize a trajectory for a **simplified model** of the dynamics
- a whole-body controller is used to generate commands for the robot

what is a gait?

- a **gait** is the way a person **walks**
- in humanoid robotics, **gait generation** the process by which we generate a trajectory for the **Center of Mass** (CoM) of the robot
- assume we want to go from a start position to a goal position: let us now look at the conceptual steps we have to go through in order to achieve this

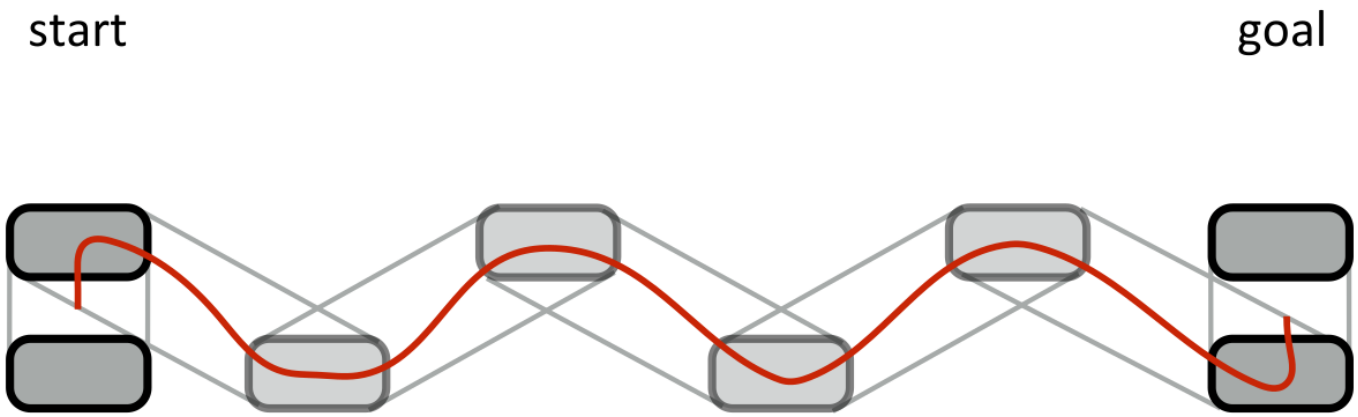
gait generation

- the first thing to do is **plan** a sequence of **footsteps** (e.g., using RRT); this defines the shape and position of the **support polygon** at each time during the walk



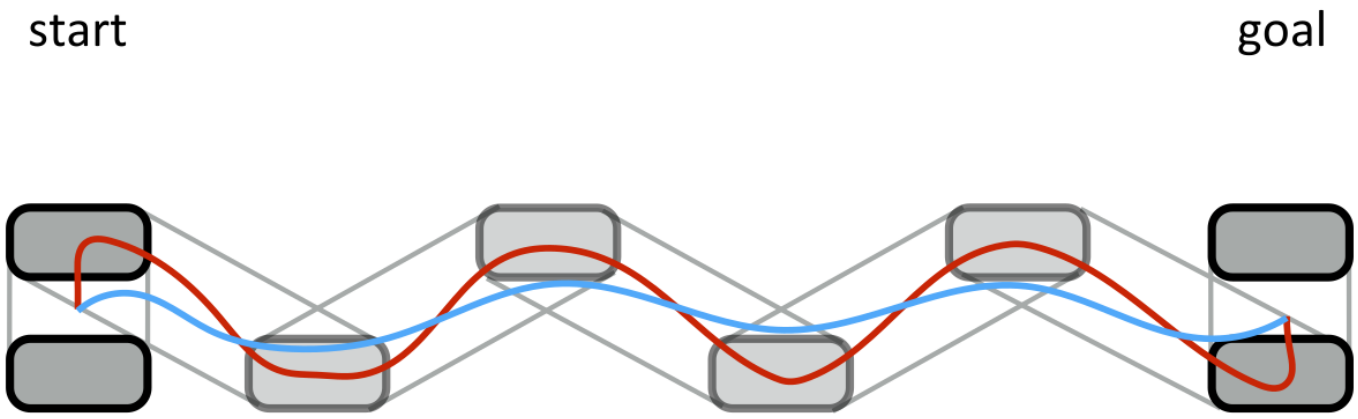
gait generation

- now we can design a **ZMP trajectory** that is always inside the **support polygon**: it uses double support phases to transition between consecutive footsteps



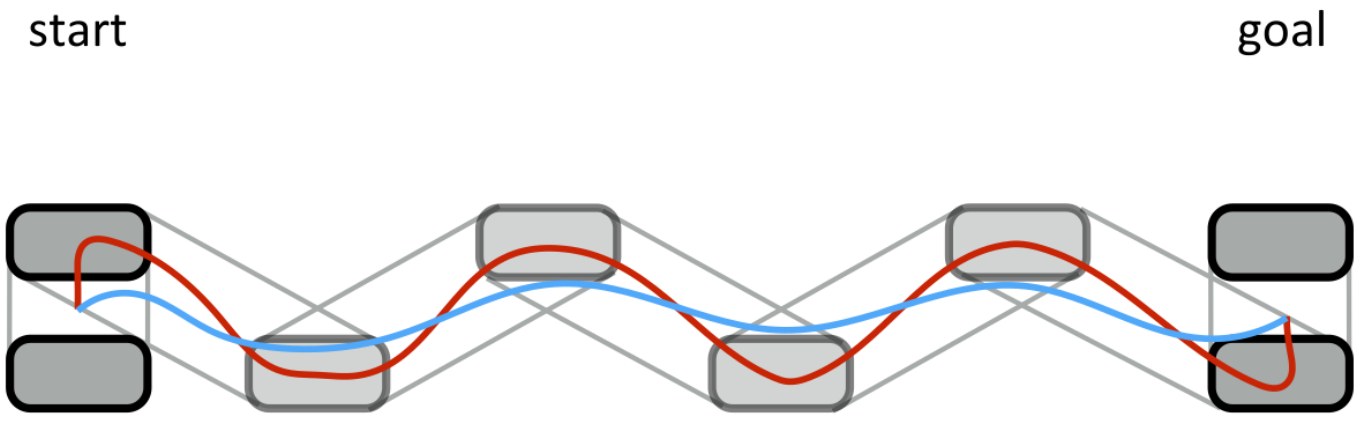
gait generation

- the next step is to generate a **CoM trajectory** that satisfies the dynamics of the **simplified model** we chose to use (e.g., the LIP model)



gait generation

- now we generate the other trajectories that we need (e.g., feet, torso orientation, ...) and track them using a **whole-body controller**



gait generation

- there are two complementary ways of viewing the same model:
- the ZMP p_z is the input, and the state is position and velocity of the center of mass p_c

$$\textbf{ZMP} \implies \boxed{\ddot{\boldsymbol{p}}_c = \eta^2(\boldsymbol{p}_c - \boldsymbol{p}_z)} \implies \textbf{CoM}$$

- in this view, we could design an arbitrary ZMP trajectory $p_z(t)$, plug it inside the model and **integrate** forward to obtain the CoM trajectory $p_c(t)$
- however, the LIP is **unstable**! if we do not keep this into account we would get a **diverging** CoM trajectory (we can solve this by adding a constraint to the MPC)

gait generation

- the other possibility is to view the CoM acceleration $\ddot{p}_c$ as the input, and the ZMP p_z as the output

$$\textbf{CoM acc} \implies \boxed{\ddot{\boldsymbol{p}}_c = \boldsymbol{u}, \quad \boldsymbol{p}_z = \boldsymbol{p}_c - \frac{1}{\eta^2}\ddot{\boldsymbol{p}}_c} \implies \textbf{ZMP}$$

- this is typically referred to as the **Cart-Table** (CT) model: here we have an output ZMP trajectory that we want to **track**

- we can have an **optimization-based controller** trying to minimize the distance between the ZMP and the reference trajectory

model predictive control

- starting from the current time-step k, we want to predict a **discrete** sequence of input and states over N future time-steps

$$\boldsymbol{x}_k, \boldsymbol{x}_{k+1}, \ldots, \boldsymbol{x}_{k+N-1}, \boldsymbol{x}_{k+N}$$
$$\boldsymbol{u}_k, \boldsymbol{u}_{k+1}, \ldots, \boldsymbol{u}_{k+N-1}$$

- these states and inputs are connected via the systems **dynamics**, which we write in discrete form

$$\boldsymbol{x}_{i+1} = f(\boldsymbol{x}_i, \boldsymbol{u}_i)$$

- for physical systems, which have continuous dynamics, we assume to have discretized them in some way (e.g., Euler, Runge-Kutta, ...)

model predictive control

- at time-step k, an **MPC** computes an optimal control sequence by solving an optimization problem of the form

$$\begin{aligned}
&\min \sum_{i=k}^{k+N-1} l(\boldsymbol{x}_i, \boldsymbol{u}_i) + l_N(\boldsymbol{x}_{k+N}) \\
&\text{s.t.} \quad \boldsymbol{x}_k = \bar{\boldsymbol{x}}_k \\
&\qquad \boldsymbol{x}_{i+1} = f(\boldsymbol{x}_i, \boldsymbol{u}_i) \quad \text{for } i = k, \ldots, k+N-1 \\
&h(\boldsymbol{x}_i, \boldsymbol{u}_i) = 0 \quad \text{for } i = k, \ldots, k+N \\
&g(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq 0 \quad \text{for } i = k, \ldots, k+N
\end{aligned}$$

- the initial state $\boldsymbol{x}_k$ of the predicted sequence is set equal to the **measured state** $\bar{\boldsymbol{x}}_k$ (**feedback!**)
- other constraints ensure that the **dynamics** are satisfied, and enforce other requirements on states and inputs

model predictive control

- generic optimization problems can be **computationally heavy**, but an MPC controller must give us solutions at a rate of about 100 Hz
- to keep computation time manageable, we typically formulate this as a **Quadratic Program** (QP), i.e., an optimization problem with **quadratic cost** and **linear constraints**
- these problems can be solved using off-the-shelf libraries in **milliseconds**

preview control

- MPC gait generation using the LIP model was introduced with the **preview controller** [Kajita et al.]
- this technique used a cost function to track a **reference ZMP** trajectory, designed in such a way to always be inside the support polygon
- it did not make use of **constraints**, therefore the solution was given in **closed form** (without the need to use a QP solver)

Kajita et al., "Biped walking pattern generation by using preview control of zero-moment point", International Conference on Robotics and Automation, 2003

preview control

- the dynamics is represented having the **CoM jerk** $\dddot{p}_c$ (third time derivative) as the input, and the **ZMP** as the output

$$\underbrace{\frac{d}{dt}\begin{pmatrix} p_c \\ \dot{p}_c \\ \ddot{p}_c \end{pmatrix}}_{\dot{\boldsymbol{x}}} = \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} p_c \\ \dot{p}_c \\ \ddot{p}_c \end{pmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{\boldsymbol{B}} \underbrace{\dddot{p}_c}_{\boldsymbol{u}}$$

$$\underbrace{p_z}_{\boldsymbol{y}} = \underbrace{\begin{pmatrix} 1 & 0 & -1/\eta^2 \end{pmatrix}}_{\boldsymbol{C}} \underbrace{\begin{pmatrix} p_c \\ \dot{p}_c \\ \ddot{p}_c \end{pmatrix}}_{\boldsymbol{x}}$$

- the jerk $\dddot{p}_c$ is used instead of the acceleration $\ddot{p}_c$ to avoid the output directly depending on the input, which might cause discontinuous ZMP trajectories

preview control

- the optimization problem can now be written as

$$\min_{\boldsymbol{x}_i, \boldsymbol{u}_i} \sum_{i=k}^{k+N-1} \left(\left(\dddot{p}_c^i \right)^2 + \beta \left(p_z^i - p_z^{i,\text{ref}} \right)^2 \right) + \beta \left(p_z^{k+N} - p_z^{k+N,\text{ref}} \right)^2$$

$$\text{s.t.} \quad \boldsymbol{x}_k = \bar{\boldsymbol{x}}_k$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i \quad \text{for } i = 0, \ldots, N-1$$

- there are constraints to enforce the consistency of the predicted sequence with the **dynamics** of the model
- β is a weight that modulates how strongly we want to **track** the reference

preview control

- the predicted states can be expressed in terms of the measured current state $\bar{\boldsymbol{x}}_k$ and the predicted input sequence $\boldsymbol{u}_k, \ldots, \boldsymbol{u}_{k+N-1}$ by performing **algebraic substitutions**

$$
\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{A}\bar{\boldsymbol{x}}_k + \boldsymbol{B}\boldsymbol{u}_k \\
\boldsymbol{x}_{k+2} &= \boldsymbol{A}\boldsymbol{x}_{k+1} + \boldsymbol{B}\boldsymbol{u}_{k+1} = \boldsymbol{A}(\boldsymbol{A}\bar{\boldsymbol{x}}_k + \boldsymbol{B}\boldsymbol{u}_k) + \boldsymbol{B}\boldsymbol{u}_{k+1} \\
&= \boldsymbol{A}^2\bar{\boldsymbol{x}}_k + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{B}\boldsymbol{u}_{k+1} \\
\boldsymbol{x}_{k+3} &= \boldsymbol{A}\boldsymbol{x}_{k+2} + \boldsymbol{B}\boldsymbol{u}_{k+2} = \boldsymbol{A}(\boldsymbol{A}^2\bar{\boldsymbol{x}}_k + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{B}\boldsymbol{u}_{k+1}) + \boldsymbol{B}\boldsymbol{u}_{k+2} \\
&= \boldsymbol{A}^3\bar{\boldsymbol{x}}_k + \boldsymbol{A}^2\boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_{k+1} + \boldsymbol{B}\boldsymbol{u}_{k+2} \\
&\vdots \\
\boldsymbol{x}_{k+N} &= \boldsymbol{A}^N\bar{\boldsymbol{x}}_k + \boldsymbol{A}^{N-1}\boldsymbol{B}\boldsymbol{u}_k + \cdots + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_{k+N-2} + \boldsymbol{B}\boldsymbol{u}_{k+N-1}
\end{aligned}
$$

preview control

- preview control is essentially using **unconstrained MPC** to realize **tracking** of a predefined ZMP trajectory
- the absence of constraints implies that this method is quite fast, because the solution can be recovered in closed form (we have to invert a large matrix but only once)
- however, without constraints we cannot guarantee that the **ZMP** will always be inside the **support polygon**

constrained MPC

- since we know the footstep plan, it is possible to enforce **constraints** that will ensure that the ZMP is always contained in the support polygon

$$p_z^{k+i,\min} \leq \boldsymbol{C}\boldsymbol{x}_{k+i} \leq p_z^{k+i,\max}$$

- if the robot is in **single support** $p_z^{k+i,\min}$ and $p_z^{k+i,\max}$ represent the lower and upper edge of the foot

- if the robot is in **double support** the expression is a bit more involved, but still linear

Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations", International Conference on Humanoid Robots, 2006

constrained MPC

- having constrained the ZMP, we do not need to explicitly track the reference in the cost function (but we still minimize the square of the input to make the problem convex)

$$\min_{\boldsymbol{x}_i, \boldsymbol{u}_i} \sum_{i=k}^{k+N-1} \left(\dddot{p}_c^i \right)^2$$

$$\text{s.t.} \quad \boldsymbol{x}_k = \bar{\boldsymbol{x}}_k$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i \quad \text{for } i = 0, \ldots, N-1$$

$$p_z^{k+i,\min} \leq \boldsymbol{C}\boldsymbol{x}_{k+i} \leq p_z^{k+i,\max} \quad \text{for } i = 1, \ldots, N$$

- this in principle means that the MPC is free to choose the "best" trajectory, although in practice we might still want to include a ZMP tracking term if it gives better results

MPC on the LIP

- when we derived the LIP model, we saw that it can be written in state-space form by choosing the **ZMP** as the **input**

- to avoid discontinuous ZMP trajectories we can **dynamically extend** the system with the ZMP derivative $\dot{p}_z$ as new input

$$\underbrace{\frac{d}{dt}\begin{pmatrix} p_c \\ \dot{p}_c \\ p_z \end{pmatrix}}_{\dot{\boldsymbol{x}}} = \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ \eta^2 & 0 & -\eta^2 \\ 0 & 0 & 0 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} p_c \\ \dot{p}_c \\ p_z \end{pmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{\boldsymbol{B}} \underbrace{\dot{p}_z}_{\boldsymbol{u}}$$

MPC locomotion example

- a working implementation of this last MPC scheme is available at https://github.com/DIAG-Robotics-Lab/ismpc
- it is written in Python using the DART robotics toolkit
- the optimization problems are solved with OSQP, interfaced using CasADi

MPC locomotion example

automatic footstep placement

- when using a linear simplified model, the balance condition is expressed as a **linear constraint** on the **ZMP**

$$p_z^{k+i,\text{foot}} \leq \boldsymbol{C}\boldsymbol{x}_{k+i} \leq p_z^{k+i,\max}$$

- if the footprint has size $2d$, its edges can be written as the foot position p_f^{k+i} plus or minus half the foot size

$$p_f^{k+i} - d \leq \boldsymbol{C}\boldsymbol{x}_{k+i} \leq p_f^{k+i} + d$$

- this means that if we consider the foot position as an **optimization variable**, the ZMP constraint is still **linear**

$$-d \leq \boldsymbol{C}\boldsymbol{x}_{k+i} - p_f^{k+i} \leq d$$

automatic footstep placement

- we can formulate an MPC with with the footstep positions as additional optimization variables [Herdt at al.], that still only has linear constraints

- if balance can't be maintained by simply changing the ZMP and CoM trajectory, the MPC can decide to **move the the foot** to a different location

- this achieves **automatic footstep placement**, and allows the robot to be more robust to disturbances

Herdt et al., "Walking without thinking about it",
International Conference on Intelligent Robots and Systems, 2010

automatic footstep placement

- an example of automatic footstep placement: the robot adapts after getting pushed (not from the previous paper)