Model Predictive Control for NAO Locomotion

RoboCup P&S, 2025

1 Introduction

Model Predictive Control (MPC) is an advanced control strategy used in systems where decision-making must account for constraints, time-varying dynamics, and optimization objectives. Here's a breakdown of how it works:

1. **Model-Based Prediction:** MPC uses a mathematical model of the system (eg. differential equations) to predict how the system will evolve over time. The model can be linear or nonlinear, depending on the system's complexity.
2. **Define an Optimization Problem:**
   - **Objective Function:** The goal, such as minimizing energy or tracking a desired trajectory.
   - **Constraints:** Limits on system states (eg. one foot should always be on the ground), and control inputs (eg. don't move the robot joints too fast, otherwise they will break).

$$
\begin{aligned}
\min_{\{x_i\},\{u_i\}} \quad & \sum_{i=0}^{N-1} l(x_i, u_i) & \\
\text{s.t.} \quad & x_{i+1} = f(x_i, u_i) & \text{system model} \\
& x_i \in \mathcal{X} & \text{state constraints} \\
& u_i \in \mathcal{U} & \text{input constraints} \\
& x_0 = x(k) & \text{measurement/initialization}
\end{aligned}
$$

Figure 1: MPC Optimization Problem

3. **Finite Horizon Planning:** MPC operates over a finite time horizon (eg. 0.5 seconds or "the next two foot steps"). The optimization problem is solved for this horizon to determine the best sequence of actions.
4. **Receding Horizon Approach:** From the optimized sequence of actions, only the first action is applied to the system. After applying this control, the system state is updated, and the optimization is solved again over a new horizon starting from the current state.
5. **Iterative Process:** Repeat the prediction, optimization, and application steps at a fast rate (eg. every 20 milliseconds). Like this, the MPC becomes robust to modeling uncertainties, external disturbances, and changes in the environment.

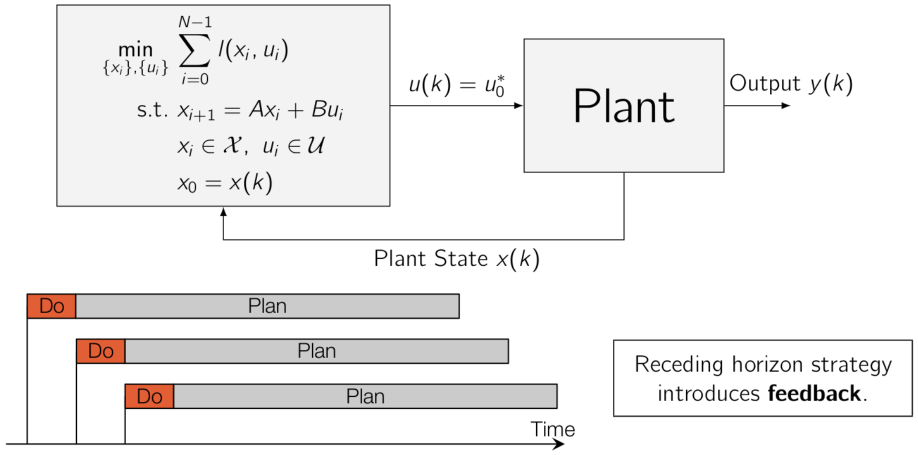

$\min_{\{x_i\},\{u_i\}} \sum_{i=0}^{N-1} l(x_i, u_i)$
s.t. $x_{i+1} = Ax_i + Bu_i$
$x_i \in \mathcal{X},\ u_i \in \mathcal{U}$
$x_0 = x(k)$
$u(k) = u_0^*$
Plant
Output $y(k)$
Plant State $x(k)$
Do
Plan
Do
Plan
Do
Plan
Time
Receding horizon strategy introduces **feedback**.

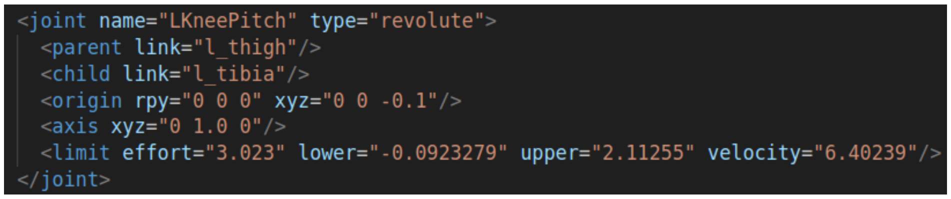
Figure 2: MPC Closed Loop

2 Humanoid Robot Models

To use MPC for humanoid locomotion we first need a model of the robot. The most common way of representing robots for control tasks is with the **Unified Robot Description Format (URDF)**. This is a file that describes the robot's physical properties:

- **Links:** The rigid parts of the robot (like legs, arms, torso). Each link is described by its total mass, center of mass, and inertia (mass distribution).
- **Joints:** The connections between links that allow movement. These joints are what we can control, by providing them with desired positions, velocities, or even torques.
- **Frames:** These identify specific 3D points of the robot, like the heel of the foot. They are useful for formulating constraints like "the foot should land here at this time".
- **Visual and Collision Geometry:** How the robot looks and how it interacts with the environment.

Figure 3 displays a section of the NAO robot's URDF that describes the knee joint. It specifies which links are the parent and child of this joint, the relative position and rotation with regards to the parent link (with `origin` and `axis`), as well as the joint limits (torque, position, velocity).



```
<joint name="LKneePitch" type="revolute">
  <parent link="l_thigh"/>
  <child link="l_tibia"/>
  <origin rpy="0 0 0" xyz="0 0 -0.1"/>
  <axis xyz="0 1.0 0"/>
  <limit effort="3.023" lower="-0.0923279" upper="2.11255" velocity="6.40239"/>
</joint>
```

Figure 3: URDF description of a the NAO robot's knee joint

The URDF provides a detailed description of the robot's kinematic and inertial properties. However, it does not describe how the robot's state changes over time, given input commands to the joints. Therefore, to control the robot with MPC, we need to derive dynamic models. Figure 4 describes the most commonly used humanoid models. Let's go through them one by one:

- **Whole-body Dynamics:** This is the most complete model, using the full Equations of Motion (EoM) to describe the dynamics of every link and joint in the robot. This provides high accuracy but with significant computational cost.

- **Centroidal Dynamics:** This model focuses on the motion of the robot's center of mass (CoM) and its total angular momentum, simplifying the dynamics while still capturing essential balance and locomotion behaviors.
- **Single Rigid Body:** This model treats the entire robot as a single rigid object with constant inertia (mass distribution). It is especially useful for robots with very light limbs compared to the body, such as quadrupeds.
- **Inverted Pendulum:** This highly simplified model represents the robot as a point mass (no inertia), balanced on a single contact point with the ground. This model is primarily used to approximate balance in the forward-backward direction. The leg length can be fixed (Linear Inverted Pendulum) or modeled with a spring (Spring-Loaded Inverted Pendulum).

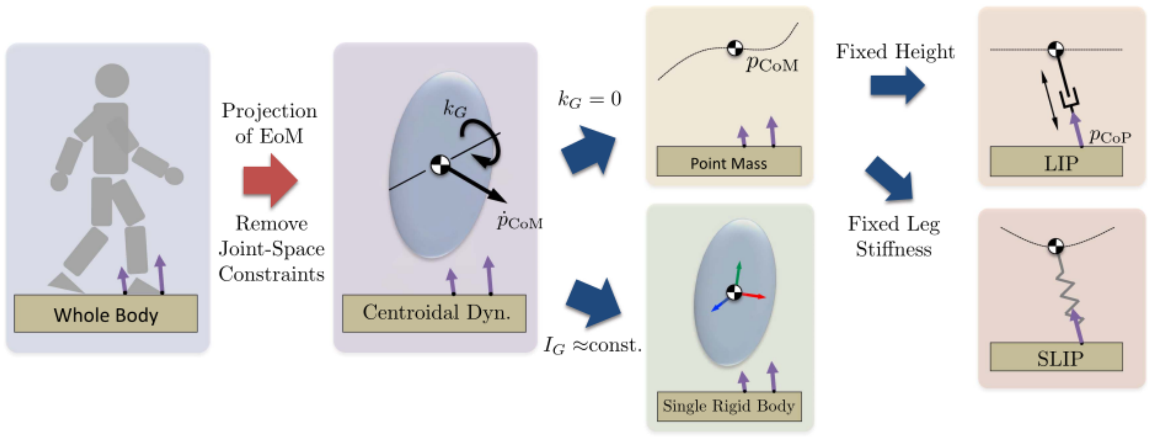


Figure 4: Humanoid models overview

3 MPC Formulation

3.1 Centroidal Dynamics

We will use centroidal dynamics for our MPC, since it strikes a good balance between computational efficiency and model accuracy. To describe the motion of the CoM we use the variable $\boldsymbol{h}_{\mathrm{com}} = [\boldsymbol{p}_{\mathrm{com}}, \boldsymbol{l}_{\mathrm{com}}] \in \mathbb{R}^6$ which defines the linear and angular momentum of the CoM (aka. centroidal momentum). Intuitively, the robot's interaction with the ground is what enables it to walk, so we need to describe how the ground contact forces $\boldsymbol{F}_c$ affect the CoM:

$$\dot{\boldsymbol{h}}_{\mathrm{com}} = \begin{bmatrix} \dot{\boldsymbol{p}}_{\mathrm{com}} \\ \dot{\boldsymbol{l}}_{\mathrm{com}} \end{bmatrix} = \begin{bmatrix} \sum_{c \in \mathcal{C}} \boldsymbol{F}_c + m\boldsymbol{g} \\ \sum_{c \in \mathcal{C}} \boldsymbol{r}_{\mathrm{com},c} \times \boldsymbol{F}_c \end{bmatrix} \tag{1}$$

where, for each contact point c the 3D force vector $\boldsymbol{F}_c$ is acting on the robot at the location $\boldsymbol{r}_{\mathrm{com},c}$ (relative vector to the CoM).

Furthermore, we also want to describe how the robot's joint configuration affects the CoM. Imagine you are standing on one leg. If you lose balance, you instinctively move your arms to stabilize. This works, because your joint configuration affects your mass distribution, and hence how your CoM moves.

In mathematical terms, we describe the robot's joint and torso configuration with the so-called "generalized coordinates" $\boldsymbol{q} = [\boldsymbol{q}_b, \boldsymbol{q}_j] \in \mathbb{R}^{7+n_j}$. Here, $\boldsymbol{q}_b$ describes the torso (or base) through its 3D position and 4D quaternion orientation. The quaternion is a clever way of representing the orientation, avoiding singularities that occur when we use Euler angles for the orientation. The joint configuration is given by $\boldsymbol{q}_j$, noting the angle of each joint relative to some default angle.

The current robot configuration $\boldsymbol{q}$ and its respective velocity $\boldsymbol{v}$ affect the centroidal momentum in the following way:

$$\boldsymbol{h}_{\text{com}} = \boldsymbol{A}(\boldsymbol{q})\boldsymbol{v} = \begin{bmatrix}\boldsymbol{A}_b(\boldsymbol{q}) & \boldsymbol{A}_j(\boldsymbol{q})\end{bmatrix}\begin{bmatrix}\boldsymbol{v}_b \\ \boldsymbol{v}_j\end{bmatrix} \tag{2}$$

where $\boldsymbol{A}(\boldsymbol{q})$ is the centroidal momentum matrix (CMM).

3.2 MPC States and Inputs

To formulate our MPC problem, we need to define what the states and inputs of our centroidal-based system are. For the state, we use the centroidal momentum and current configuration of the robot:

$$\boldsymbol{x} = \begin{bmatrix}\boldsymbol{h}_{\text{com}}, \boldsymbol{q}\end{bmatrix} \in \mathbb{R}^{6+n_q}$$

It is important to define our state as something that can be measured/estimated on the robot, since whenever we re-solve the MPC we initialize it with the current state. For the joint angles this is easy because each joint has a sensor that reads this value. For the base position/rotation and the CoM linear/angular momentum these values are estimated from IMU sensor readings, which can be less accurate.

For the inputs, we need to define variables that we can directly/indirectly affect to control the state of the robot. We use the joint velocity commands and the desired ground contact forces:

$$\boldsymbol{u} = \begin{bmatrix}\boldsymbol{v}_j, \{\boldsymbol{F}_c\}_{\forall c \in \mathcal{C}}\end{bmatrix} \in \mathbb{R}^{n_j+n_f}$$

The joint velocity commands we can pass directly to the joint motors to track them. The ground reaction forces are usually controlled with a low-level "whole body controller" that uses the whole body dynamics described in Section 2 to compute motor torques. We will not consider this, and instead assume the forces we compute can be tracked.

3.3 Discretized Dynamics

Since MPC is always done for discretized systems, we need to define our discrete dynamics as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

In Section 3.1 we covered the continuous centroidal dynamics, which can be summarized in:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$$

TASK: Write out $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$.

Then we use Euler integration to approximate the discretized dynamics:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)dt$$

3.4 Constraints

Now we will formulate the constraints on our states and inputs. To do this we need to separate when each foot is in contact with the ground, and when it is in the swing phase. We define the contact parameter $\sigma_{c,k} \in \{0, 1\}$, where for $\sigma_{c,k} = 1$ the contact point c is on the ground at time step k, and for $\sigma_{c,k} = 0$ it is in swing. These values can be multiplied with constraints to activate or deactivate them.

Force constraints: We define the contact forces $\{\boldsymbol{F}_c\}_{\forall c \in \mathcal{C}}$ at the 4 corners of each foot. When the foot is in contact with the ground, the normal force in z-direction must be positive, and the friction cone constraint must hold:

$$\sigma_{c,k}\boldsymbol{F}_{c,k,z} \geq 0$$

$$\sigma_{c,k}\mu^2\boldsymbol{F}^2_{c,k,z} \geq \sigma_{c,k}(\boldsymbol{F}^2_{c,k,x} + \boldsymbol{F}^2_{c,k,y})$$

where μ is the friction coefficient.

When the foot is not in contact with the ground, the force must be zero:

$$(1 - \sigma_{c,k})\boldsymbol{F}_{c,k} = \boldsymbol{0}$$

Velocity Constraints: Since we don't want to add velocity constraints on each corner of each foot (this would become redundant), we define the velocity of the center of each foot as $\boldsymbol{v}_{c,k}$. This contains both linear and angular components $\boldsymbol{v}_{\text{lin},c,k}$ and $\boldsymbol{v}_{\text{ang},c,k}$. During contact, both linear and angular velocities must be zero:

$$\sigma_{c,k}\boldsymbol{v}_{\text{lin},c,k} = \mathbf{0}$$
$$\sigma_{c,k}\boldsymbol{v}_{\text{ang},c,k} = \mathbf{0}$$

During the swing phase, the linear velocity must track a reference trajectory in z-direction:

$$(1 - \sigma_{c,k})\boldsymbol{v}_{\text{lin},c,k,z} = \boldsymbol{v}_z^{\text{ref}}$$

This reference trajectory is computed using a Bezier curve (already implemented for you).

The angular velocity during the swing phase should be zero in x/y-directions, keeping the foot flat while it is in the air:

$$(1 - \sigma_{c,k})\boldsymbol{v}_{\text{ang},c,k,x/y} = \mathbf{0}$$

The fact that the linear x/y-velocities are not constrained during swing means that the MPC can optimize the foot trajectories in the air and determine the most effective location for the next step.

3.5 Cost Function

We define a quadratic cost function for the MPC, which penalizes deviations from a desired state and input at each time step:

$$J = \sum_{k=0}^{N-1} \left(\left\| \boldsymbol{x}_k - \boldsymbol{x}^{\text{des}} \right\|_{\mathbf{Q}}^2 + \left\| \boldsymbol{u}_k - \boldsymbol{u}^{\text{des}} \right\|_{\mathbf{R}}^2 \right) + \left\| \boldsymbol{x}_N - \boldsymbol{x}^{\text{des}} \right\|_{\mathbf{Q}}^2 \tag{3}$$

where $\mathbf{Q} \succeq 0$ and $\mathbf{R} \succ 0$ are weight matrices for states and inputs, respectively. For the state, the desired $\boldsymbol{h}_{\text{com}}$ tracks a linear/angular velocity specified by the user, and the desired $\boldsymbol{q}$ is the nominal robot configuration with all default joint angles. For the input, the desired joint velocities are zero, and the desired contact forces evenly distribute the robot's weight. Note that the terminal cost on $\boldsymbol{x}_N$ is the same as the stage cost on $\boldsymbol{x}_k$.