Institute for Cognitive Systems
Technische Universität München
M.Sc. Simon Armleder

Modeling and control of legged robots
Tutorial 6: Linear Model Predictive Control for Walking

1 Introduction

In this tutorial we will explore a simple linear model predictive controller (LMPC) to generate stable walking motions. The goal is to convert a given sequence of footsteps into feasible trajectories for the center of mass (CoM) and zero moment point (ZMP). These trajectories can then later be tacked by a whole-body controller to generate the appropriate joint motions.

2 Bipedal Walking

As explained during the lecture, the complex high dimensional non-linear dynamics of a humanoid robot

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} + \sum_{k=\{l,r\}} \mathbf{J}_k^\mathsf{T}\mathbf{W} \tag{1}$$

is usually not suitable for direct optimization. Instead it is quite common to reduce this model to the six coordinates of the under actuated floating base

$$\begin{bmatrix} m\ddot{\mathbf{c}} \\ \dot{\mathbf{L}}(\mathbf{c}) \end{bmatrix} = \begin{bmatrix} \mathbf{f}_\Sigma \\ \tau(\mathbf{c}) \end{bmatrix} + \begin{bmatrix} m\mathbf{g} \\ \mathbf{0} \end{bmatrix} \tag{2}$$

where on the left hand side we have the change in linear and angular momentum around the center of mass as a results of all the external forces and torques acting on our robot (right hand side).
Here, $\ddot{\mathbf{c}}$ denotes the acceleration of the center of mass and $\mathbf{L}(\mathbf{c})$ the net angular momentum expressed at the center of mass $\mathbf{c}$. The external forces are: the resultant force $\mathbf{f}_\Sigma$ that is exerted by the robot's feet against the ground, the moment $\boldsymbol{\tau}(\mathbf{c})$ around the CoM, also induced by the feet, and finally gravity $\mathbf{g}$.

To simplify even further lets assume that

1. There is no angular momentum around the center of mass: $\mathbf{L}(\mathbf{c}) = \mathbf{0}$
2. The height of the center of mass is kept constant: $\mathbf{c}_z = h, \dot{\mathbf{c}}_z = \ddot{\mathbf{c}}_z = 0$

Reasoning about the stability of the robot can in this case be simplified by introducing the Zero Moment Point (ZMP). This is a point $\mathbf{p}$ inside the robot's support polygon at which all contact moments vanish $\boldsymbol{\tau}_{xy}(\mathbf{p}) = \mathbf{0}$ (= no tilting of the robot). On flat ground it corresponds to the center of pressure at the robot's foot

$$\mathbf{p}_{xy} = \frac{\sum_i \mathbf{p}_{i,xy} f_{z,i}}{\sum_i f_{i,z}} \tag{3}$$

where $f_{z,i}$ are the normal forces in the foot sole acting at position $\mathbf{p}_{i,xy}$.

Relating the ZMP with the dynamics of the CoM in Eq. (2) and making use of the aforementioned two assumptions leads to

$$\ddot{c}_{xy} = \frac{g}{h}\left(c_{xy} - p_{xy}\right) \tag{4}$$

This is the famous **ZMP Equation**. It relates the position of the center of mass $\mathbf{c}$ and zero moment point $\mathbf{p}$ (both easy to measure) with the acceleration of the robot's center of mass. This relation is quit powerful because it means that by changing the position of these two points we can effectivity control the dynamics of the center of mass.

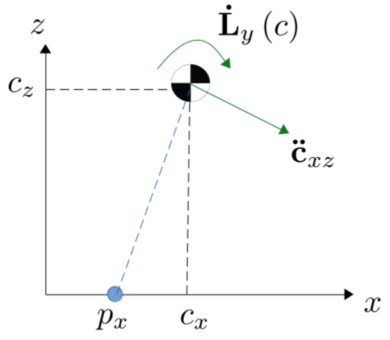

Figure 1: Simple walking model.

This walking model is also called the **Linear inverted pendulum (LIP) mode**. Kajita et al. (2001).

a Trajectory Generation

The simple nature of Eq. (4) makes it useful for the planning of walking trajectories. In a second step, a whole-body controller (acting on the full dynamics of Eq. (1)) than tracks these low dimensional trajectories.

a.1 Trajectory Optimization

In the general case trajectory generation can be formulated as an optimization problem of the form

$$\begin{aligned} \min_{\mathbf{x}_{1:T},\mathbf{u}_{0:T-1}} \quad & \sum_{k=0}^{T-1} l_k\left(\mathbf{x}_k,\mathbf{u}_k\right) + L_T\left(\mathbf{x}_T\right) \\ s.t. \quad & \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}\left(\mathbf{x}_k,\mathbf{u}_k\right)\delta t \\ & \mathbf{x}_0 = \mathbf{x}_{\text{initial}} \\ & \mathbf{x}_T = \mathbf{x}_{\text{terminal}} \\ & \dots \end{aligned} \tag{5}$$

here $\mathbf{x}_k$ corresponds to the system's state (e.g. position and velocity) and $\mathbf{u}_k$ to the control signal (e.g. actuator torques). The control horizon is denoted by T and the goal is to find the optimal state $\mathbf{x}_{1:T}$ and

control $\mathbf{u}_{1:T-1}$ over this horizon (our decision variables).
The function $l_k(\mathbf{x}_k, \mathbf{u}_k)$ encodes the cost at each step (e.g. control effort) and $L_T(\mathbf{x}_T)$ the terminal cost at goal state (e.g. distance to the goal state).
The constraint $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\,\delta t$ makes sure that the solver respects the systems dynamic $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ when euler integrating at step size δt.
The initial and final state can be set with the last two constraints in Eq. (5)

a.2 Center of Mass Trajectory Optimization

In our special case we want to use trajectory optimization based on the model in Eq. (4). Lets start by getting this system into a more familiar shape:

Model

First, since the ZMP-equation is linear it can be analytically integrated to

$$\begin{bmatrix} c_{xy}^{+} \\ \dot{c}_{xy}^{+} \end{bmatrix} = \begin{bmatrix} \cosh(\omega\Delta T) & \omega^{-1}\sinh(\omega\Delta T) \\ \omega\sinh(\omega\Delta T) & \cosh(\omega\Delta T) \end{bmatrix} \begin{bmatrix} c_{xy}^{-} \\ \dot{c}_{xy}^{-} \end{bmatrix} + \begin{bmatrix} 1 - \cosh(\omega\Delta T) \\ -\omega\sinh(\omega\Delta T) \end{bmatrix} p_{xy} \tag{6}$$

where $\begin{bmatrix} c_{xy}^{-} & \dot{c}_{xy}^{-} \end{bmatrix}$ is the position and velocity at time t and $\begin{bmatrix} c_{xy}^{+} & \dot{c}_{xy}^{+} \end{bmatrix}$ at time $t + \Delta T$. The variable $\omega = \sqrt{g/h}$ is a constant (the ratio of gravity and center of mass height).

Now, this has the familiar form of a linear system and is called the discretized system (since we are computing $\mathbf{x}^{+}$ and not $\dot{\mathbf{x}}$)

$$\mathbf{x}^{+} = \mathbf{A}_d\mathbf{x}^{-} + \mathbf{B}_d\mathbf{u} \tag{7}$$

where our state $\mathbf{x} = \begin{bmatrix} c_{xy} & \dot{c}_{xy} \end{bmatrix}^{\mathsf{T}}$ is the position and velocity of the CoM and the control input is the ZMP $u = p_{xy}$.

Cost Function

The cost function is based on a preplanned sequence of N footstpes with positions $\{\mathbf{p}_n\}_{1:N}$.
Since our goal is to prevent the robot from falling it's ZMP values should always be close to the center of the footsole (= the position of our footsteps). This means that the cost function can be formulated to keep the ZMP close to current footstep position:

$$\left\| \mathbf{u}_k - \mathbf{p}_k^{ref} \right\|^2 \tag{8}$$

where $\mathbf{p}_k^{ref}$ are the planned (reference) footstep positions. Furthermore, since the trajectories should be smooth it is common to minimize velocity and therefore add the cost

$$\|\dot{\mathbf{c}}_k\|^2 \tag{9}$$

Constraints

There are two important constraints, besides the initial and terminal conditions. The first one makes sure that the solution respects the discretized dynamics in Eq. (6)

$$\mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k \tag{10}$$

The second one, comes form the "no tilting" stability characterized by the ZMP position: The ZMP should never reach the boundaries of the foot sole. If the footprint of the robot has dimensions $s = \{w, l\}$ than the ZMP constraint become

$$\mathbf{p}_k^{ref} - \frac{s}{2} \leq \mathbf{u}_k \leq \mathbf{p}_k^{ref} + \frac{s}{2} \tag{11}$$

All together the planning problem is

$$\begin{aligned} \min_{\mathbf{x}_{1:T}, \mathbf{u}_{0:T-1}} \quad & \sum_{k=0}^{T-1} \alpha \left\| \mathbf{u}_k - \mathbf{p}_k^{ref} \right\|^2 + \gamma \left\| \dot{\mathbf{c}}_k \right\|^2 \\ s.t. \quad & \mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d \mathbf{u}_k \\ & \mathbf{p}_k^{ref} - \frac{s}{2} \leq \mathbf{u}_k \leq \mathbf{p}_k^{ref} + \frac{s}{2} \\ & \mathbf{x}_0 = \mathbf{x}_{\text{init}} \\ & \mathbf{x}_T = \mathbf{x}_{\text{terminal}} \end{aligned} \tag{12}$$

Here α and γ are two weights that balance smoothness vs the ZMP tracking error.
Our goal will be to solve this problem and generate some plans.

3 Exercises

We will use the python bindings of a framework called drake to implement Eq. (12).

1. Install pydrake on your system: stable-releases
2. Download the template folder of the tutorial, containing the python scripts

a Task 1: Getting familiar

This task will be completed inside **example_2_pydrake.py** and will show you how to use some of the functions that pydrake offers.
This code is almost finished and only requires very little changes. We will simulate the standard pendulum swingup problem: Find the commanded torques $u_{0:T-1} = \tau$ such that the pendulum ends up in its upward position with zero velocity $\mathbf{x}_T = \begin{bmatrix} q_T & \dot{q}_T \end{bmatrix}^\mathsf{T} = \begin{bmatrix} \pi & 0 \end{bmatrix}^\mathsf{T}$.

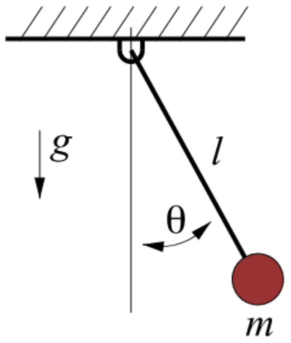

Figure 2: The simple pendulum.

The equation of motion of the pendulum can be derived through the Lagrange formalism (Wikipedia) and gives the familiar equation:

$$ml^2\ddot{q}(t) + mgl\sin\left(q\left(t\right)\right) = -b\dot{q}(t) + u(t) \tag{13}$$

where m is the mass, l the length, b the friction and g gravity.

Look for the `»TODO:` that indicate some missing code in **example_2_pydrake.py**

1. Rewrite the second order differential equation Eq. (13) as a two dimensional system of first order equation in the form $\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, u\right)$
2. Implement `pendulum_continous_dynamics()` that returns $\dot{\mathbf{x}}$
3. Now implement `pendulum_discretized_dynamics()` that uses euler integration of $\dot{\mathbf{x}}$ to discretize the system
4. Run and read the main program in the script. Answer the questions:
   - What is the meaning of the variables h?
   - What happens if we add constraints on the magnitude of $u(t)$?
   - What is the difference of this pendulum problem compared to the CoM trajectory optimization (hint look at their dynamics equations)?
5. Add some constraints that limit $u_{lb} \leq u(t) \leq u_{ub}$ and observe the outcome.

b Task 2: Linear inverted pendulum

This task will be completed inside file **ocp_lipm_2ord.py** and deals with the foot step planning problem of .
Look for the »»`TODO:` that indicate some missing code.

1. Complete the function `generate_foot_steps()` that generates footsteps in the x-direction for a given step length.

2. Complete the function `plot_foot_steps()` that takes this plan and plots the rectangular footprint of each step

3. Complete the function `generate_zmp_reference()` that returns a the an array with zmp references $\mathbf{p}^{ref}$ at all T time steps

4. Complete the `LIP_dynamics()` function that returns the ZMP Equation as Matrix $\mathbf{A}, \mathbf{B}$

5. Complete the `discrete_LIP_dynamics()` function that returns the discretized system Matrices $\mathbf{A}_d, \mathbf{B}_d$

6. In the main program setup the mathematical program, decision variables, constraints and cost functions (use the previous exercise as an example).

7. Solve the problem and plot all time plots: CoM position, velocity, acceleration, ZMP reference, ZMP bounds. Also generate a 2d plot of the CoM, ZMP and Footsteps

Some plots of the expected outcome:

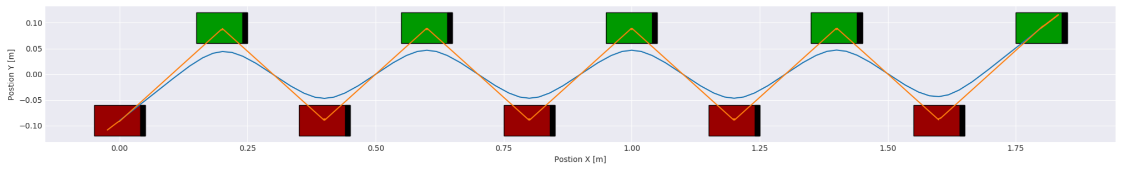

Figure 3: Footsteps and trajectories.

c Task 3: Linear Model Predictive Control

In a real application we would not solve the optimal control problem of the previous exercise for the entire footstep plan in one go. Instead the solution to Eq. (12) is computed continuously (typically at a frequency of 10 Hz). Each time we get a new vector of optimal controls $\mathbf{u}_{0:T-1}$ for the entire horizon T. But we only apply the first control $\mathbf{u}_0$ to the system. In between MPC updates we simulate the LIP dynamics at a higher rate using the latest command of the solver. This way the optimal control problem of Task 2 is turned into an model predictive controller (MPC).

Look for the »»`TODO:` that indicate some missing code inside file **mpc_lipm_2ord.py**

1. Start by copying over all the finished functions (footstep planner, dynamics, etc.) of the task 2

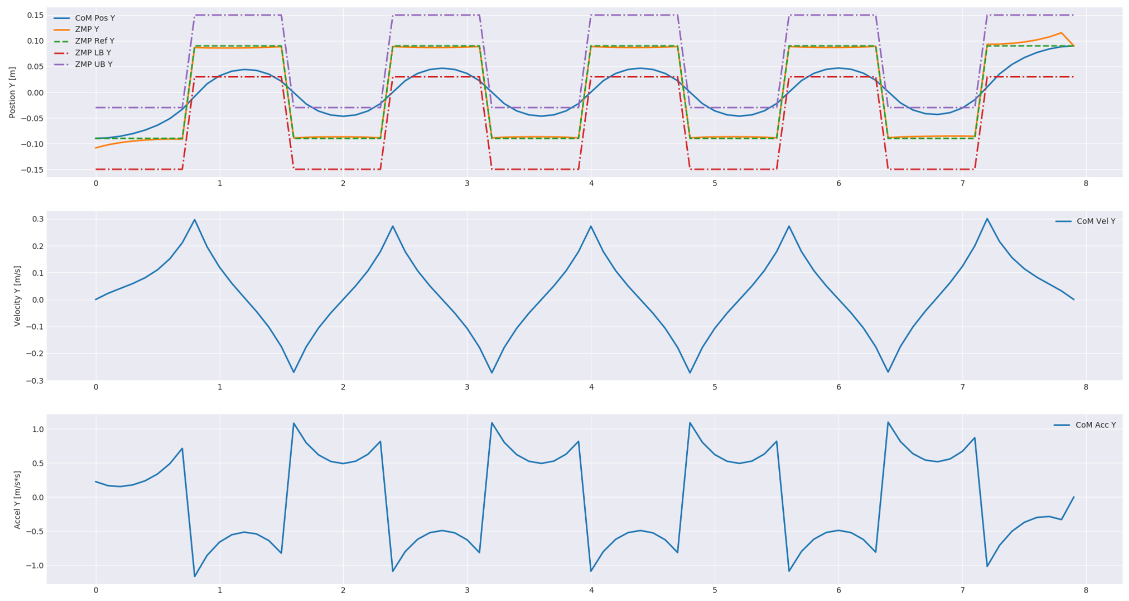

Figure 4: Signals over time in y-direction.

2. Now finish the class `MPC` by implementing the function `buildSolve()` which computes the solution for the sample problem of task 2 but starting form x_k and using reference `ZMP_ref_K` instead.
3. Next finish the class `Simulator` by implementing the function `update()` that takes the control u and updates the Linear inverted pendulum state through Euler integration and the continuous dynamics of Eq. (4)
4. Move on to the main function. Here we instantiate the Simulator and MPC. In the loop we continuously update the Simulator and every T_{mpc} we solve the MPC to compute the next command u_0. Follow the TODOs to implement this functionality.
5. When everything is running fine, add a small disturbance on the pendulum's acceleration to see the effect on the MPC.
6. Create plots similar to the last exercise, highlight the disturbance