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

Modeling and control of legged robots
Tutorial 7: Walking Control

1 Introduction

In this last tutorial we will combine parts of the previous tutorials to build the most basis walking algorithm. Our goal is to generate whole-body motions that allows our robot to step along a strait line.

2 Walking Control

As explained during the lecture it is quite common to design walking controllers in a hierarchical manner. A low dimensional model such as the Linear Inverted Pendulum (LIP) is used for planning over some future horizon. The planned references are then feed into a whole-body controller that computes the required body motions based on the full dimensional robot model.
Of cause the detailed structure involves many more modules. An overview of a possible walking controller architecture is show in Figure 1.

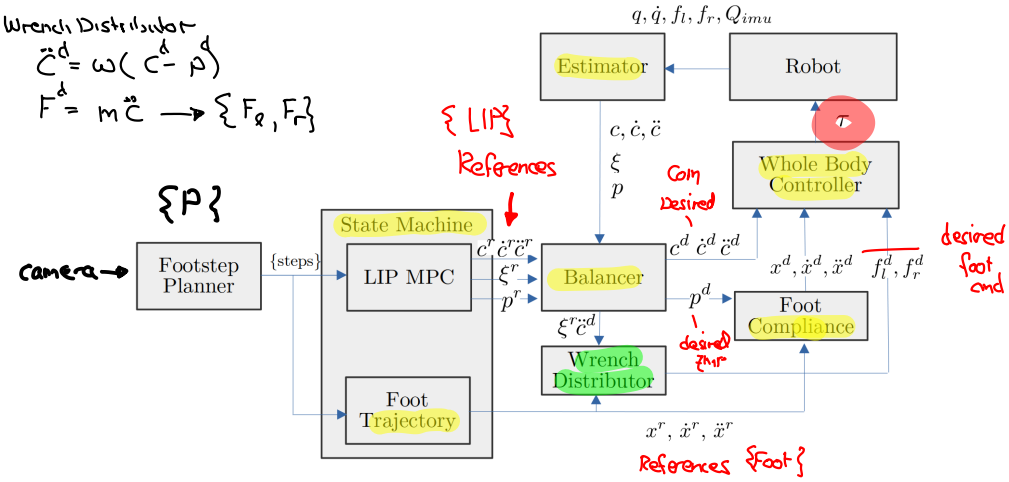


Figure 1: Walking controller architecture.

A **footstep planner** plans a sequence of steps (foot positions) the robot should take. This information if forwarded into a **state machine** to coordinate when to add/break foot contacts. It contains a **LIP MPC** that translates steps into valid reference motions for the center of mass $c^r, \dot{c}^r, \ddot{c}^r$, the divergent component of motion ξ^r, and zero moment point p^r. The **foot trajectory** module takes two steps and interpolates their required stepping motion $x^r, \dot{x}^r, \ddot{x}^r$.
The **balancer** takes the planned references and real feedback from the robots **estimators** to compute

desired commands that should stabilize the robot. The desired com motion $c^d, \dot{c}^d, \dot{c}^d$ is send to the com task inside the **whole body controller**. The desired zmp p^d can be used to further adapt foot poses by the **foot compliance** module. The **wrench distributors** job is to decide how much force to put on each foot. It outputs the desired foot forces f_l^d, f_r^d that are realized by the contact tasks inside the **whole body controller**. Robot measurements such as joint state, forces, inertial measurement unit $q, \dot{q}, f_l, f_r, Q_{imu}$ enter **estimators** to compute the quantities that can not directly be observed (real center of mass, real DCM, real ZMP).

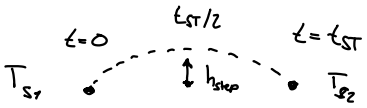
3 Exercises

Implementing the complete structure in Figure 1 requires a lot of time. Instead we only focus on some of the modules. To simplify we assume:

- Floor is flat, so foot trajectories can be implemented as polynomials and don't need to be adapted during runtime
- Torque controlled robot, so the compliance between ground and feet doesn't need to be manually generated (no Foot Compliance Module)
- Ground truth center of mass position is available (we use pybullet) and doesn't need to be estimated (no Floating-base estimator)
- No disturbances, so we can remove the balancer (however, if you want to test it you can add it in)

As a bare minimum we will need the **footstep planner**, **LIP MPC**, **foot trajectory**, **whole body controller** and a **state machine**. Luckily you already have some of them.

a Task 1: Implement Modules



a.1 Foot Trajectory

Step one is to generate foot motions that can move the feet from one contact point to another. Finish the file **foot_trajectory.py** that uses a polynomial to interpolate positions.
The input parameters are the step duration t_{step} and the step height h_{step}. The step height is the height above the ground at the middle

$$z_{\text{foot}}(0.5t_{\text{step}}) = h_{\text{step}} \tag{1}$$

The initial and final velocity and acceleration of the planned trajectory should be zero

$$\dot{\mathbf{x}}_{\text{foot}}(0) = \ddot{\mathbf{x}}_{\text{foot}}(0) = \dot{\mathbf{x}}_{\text{foot}}(t_{\text{step}}) = \ddot{\mathbf{x}}_{\text{foot}}(t_{\text{step}}) = \mathbf{0} \tag{2}$$

You can use any library you like to create the polynomial. If you want to compute it by yourself check out Polynomial regression.

- Finish **foot_trajectory.py**
- Write a short test function to check and plot the trajectories linear position $\mathbf{x}(t)$, linear velocity $\dot{\mathbf{x}}(t)$ and linear acceleration $\ddot{\mathbf{x}}(t)$.

Note: You can assume a constant foot orientation.

a.2 Foot Step planner

Next step is to generate foot steps. For this, work in the **footstep_planner.py** module. The goal is to produce a list of footsteps.

- Finish **footstep_planner.py**
- Write a short test function that plots footsteps

Note: the robot should start and stop with both feet next to each other.

a.3 LIP MPC

Next module is the linear inverted pendulum mpc of last tutorial.

- Use your previous solutions to finish **lip_mpc.py**

a.4 Robot

Lets modify our usual robot class in **talos.py** a little bit to add some special functions for walking control. Like always we want this class to publish the robot state to rviz for visualization. Here, we further add publishers for the zmp p and the dcm ξ. Additionally, we also want to see the wrenches at the foot sole of the robot.
Furthermore, we add some logic to activate and deactivate the contacts and motion tasks on the feet. For walking the feet of the robot are usually referred to as support foot and swing foot. The support foot is in contact with the ground and remains static. The swing foot is in the air and moving towards the next contact.

- Implement the publishers for visualization
- Implement the `setSupportFoot()`, `setSwingFoot()`, `updateSwingFootRef()` functions.
  Note: You may have to change the `TSIDWrapper` class for this
- Like always quickly instantiate the robot and check if standing works fine

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

b Task 2: Implement the Walking Cycle

Now that we have some basic modules, lets combine them into a walking simulation. For this we work in the file **walking.py**. Your goal is to implement the main loop that gets this going (this is essentially the **state machine** module in the figure above). Before we can start walking we need to get center of mass into the right starting position. The procedure before walking is like this:

- Create the plan and ZMP references
- Initialize the MPC and LIP Interpolator
- Shift the COM over the initial support foot
- Wait some time for the COM to complete the shift (3 seconds)

Now we generate the walking cycle, the procedure is like this:

- Our main loop updates as fast as the simulator (e.g. 1 khz)
- When ever its time to update the mpc (e.g. 10 Hz, see `conf.no_sim_per_mpc`):
  1. Get the current LIP state x_k from the interpolator
  2. Extract the ZMP reference `ZMP_ref_k` over the current horizon from the ZMP reference vector
  3. Solve the mpc and get the first control u_k
  4. increment mpc counter
- When ever its time to take a new footstep (e.g. every 0.8 seconds, see `conf.no_sim_per_step`):
  1. Get the next step location for the swing foot from the plan
  2. Set the swing foot of the robot depending on the `side` of the next step
  3. Set the support foot fo the robot depending on the other `side`
  4. Get the current location of the swing foot
  5. Plan a foot trajectory between current and next foot pose
  6. increment step counter
- In every iteration of the simulator when walking:
  1. update the foot trajectory with current step time and set the new pose, velocity and acceleration reference to the swing foot
  2. update the interpolator with the latest command u_k computed by the mpc
  3. feed the com tasks with the new com reference pos, vel, acc
  4. increment elapsed footstep time
- In every iteration of the simulator:
  1. update the simulator
  2. update the robot

Finally, plot everything you logged through out the simulation. Some example are show below.
Note: If you want to test the logic without the robot you can instead first plot the evolution of com, zmp, dcm, feet trajectories within pybullet.

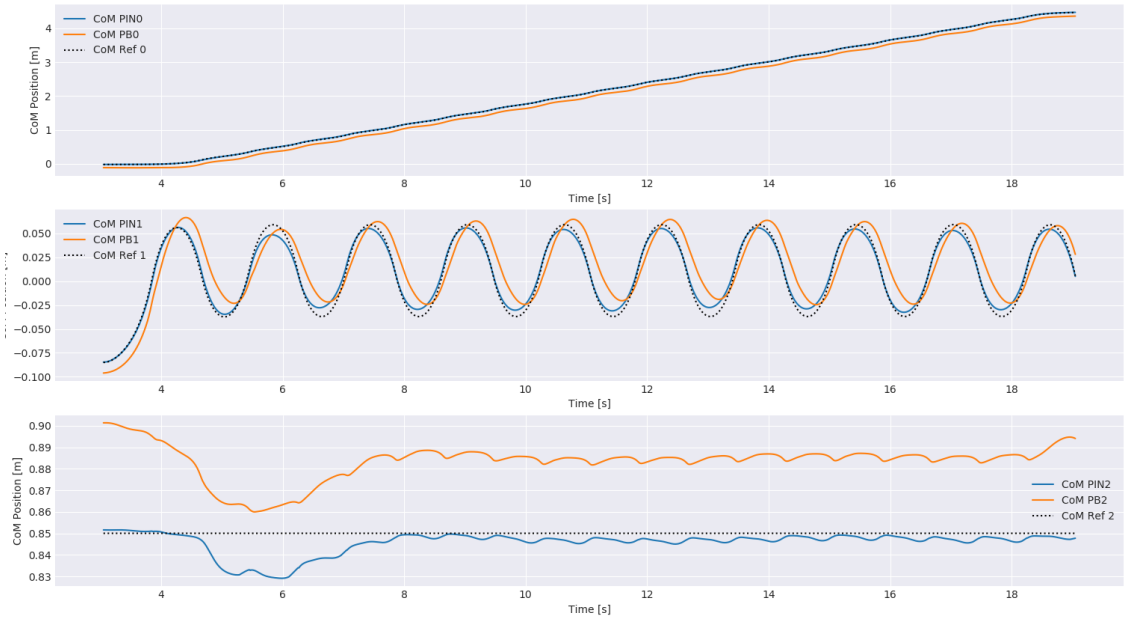


Figure 2: Center of mass position.

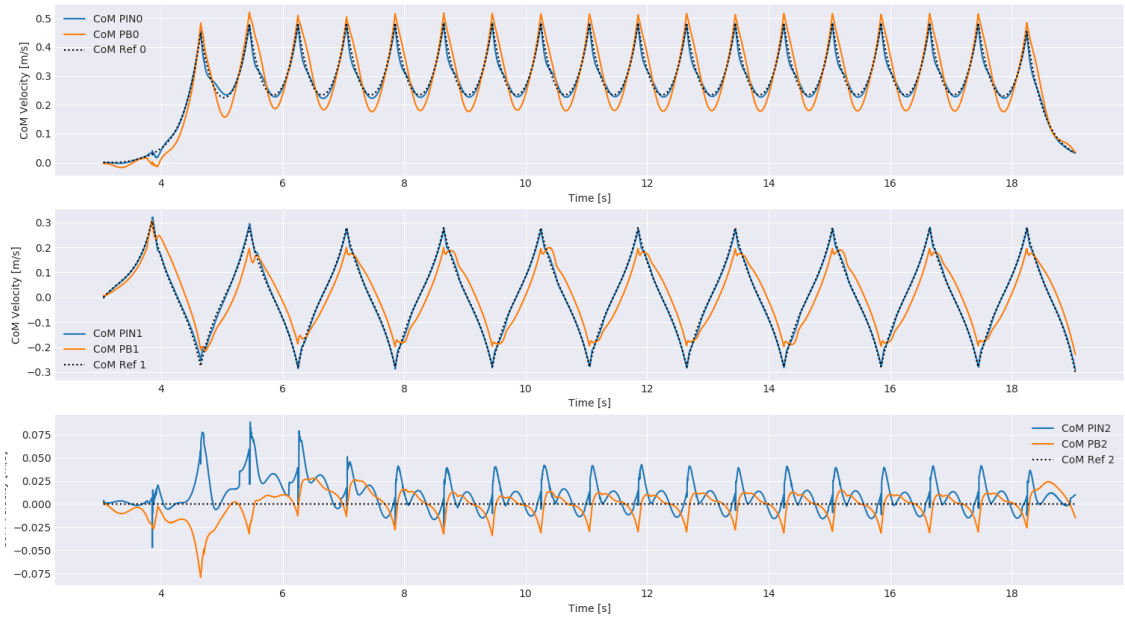


Figure 3: Center of mass velocity.

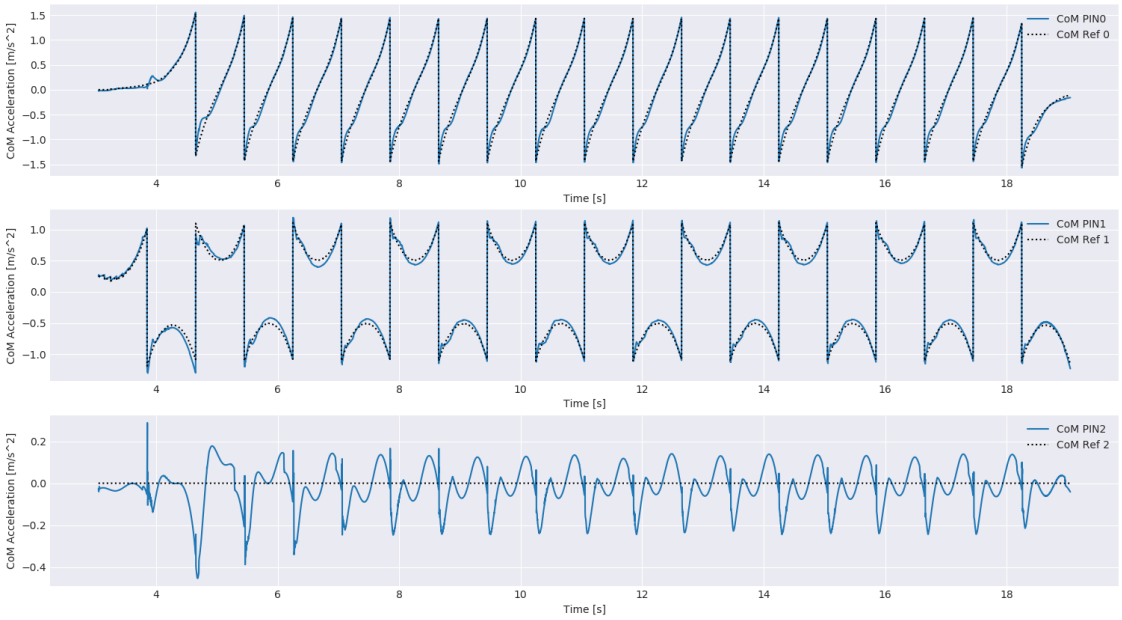


Figure 4: Center of mass acceleration.

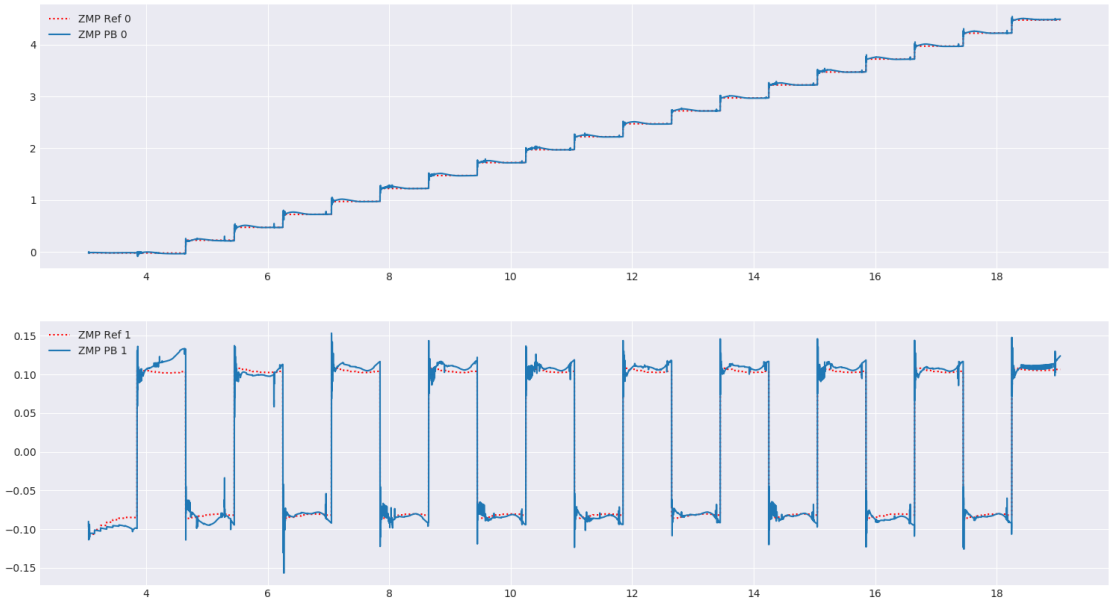


Figure 5: Zero Moment Point

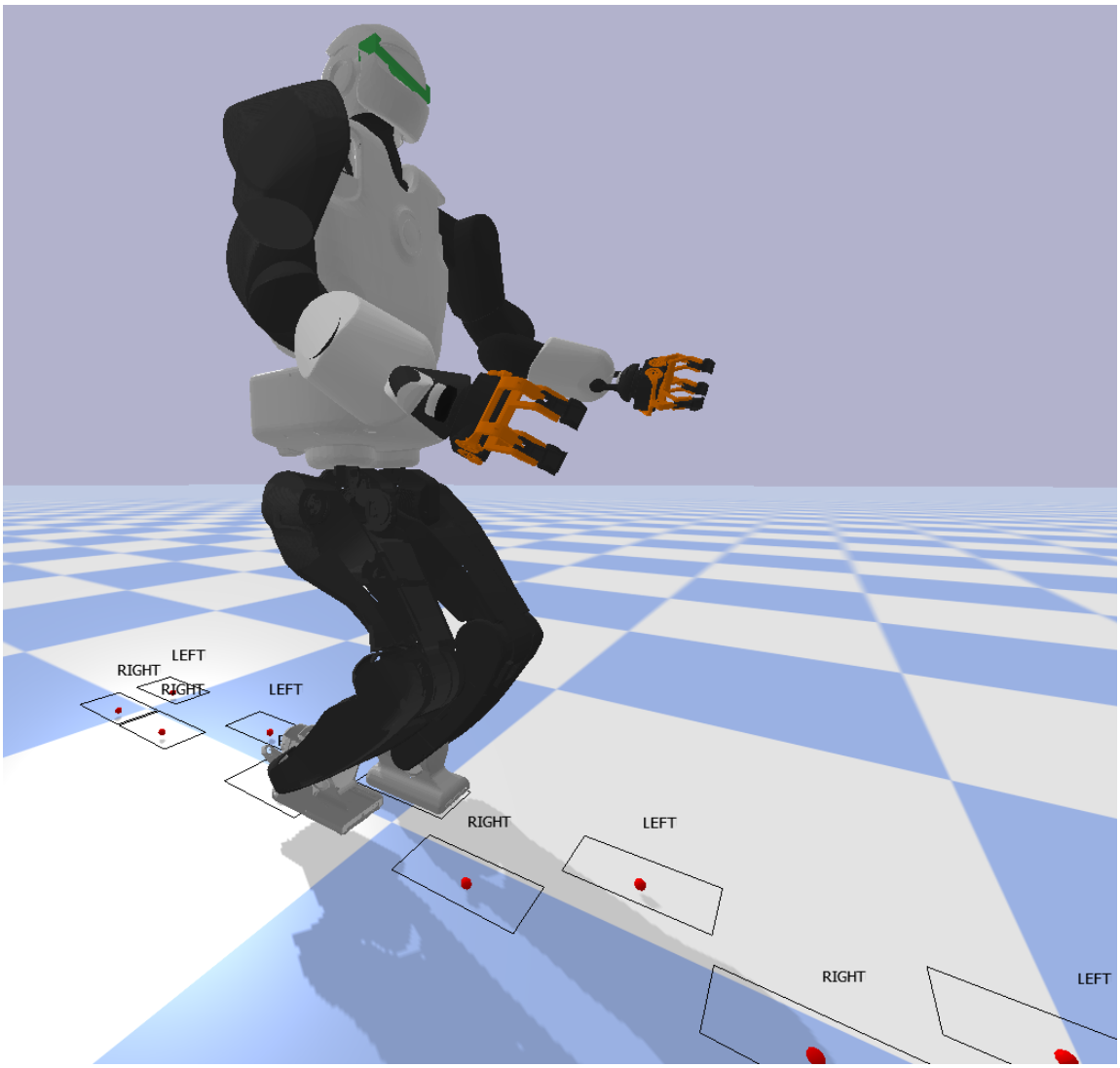


Figure 6: Talos Walking