Exercise 3: Controlling a Legged Robot

Prof. Marco Hutter

Teaching Assistants:
Fabian Jenelten, Clemens Schwarke
Jia-Ruei Chiu, Turcan Tuna

November 16, 2022

Abstract

This exercise covers the implementation of control algorithms for a simplified legged robot. The objective is to implement two controllers which, respectively, enable the tracking of a desired motion for the torso (i.e. base) and desired motion and force for the arm (i.e. end-effector). We will employ a hierarchical optimization scheme that will enable the system to execute the aforementioned tasks while also respecting a set of necessary constraints. Computation of all kinematic and dynamic quantities are already provided, and only specification of a subset of optimization costs and constraints are to be implemented.

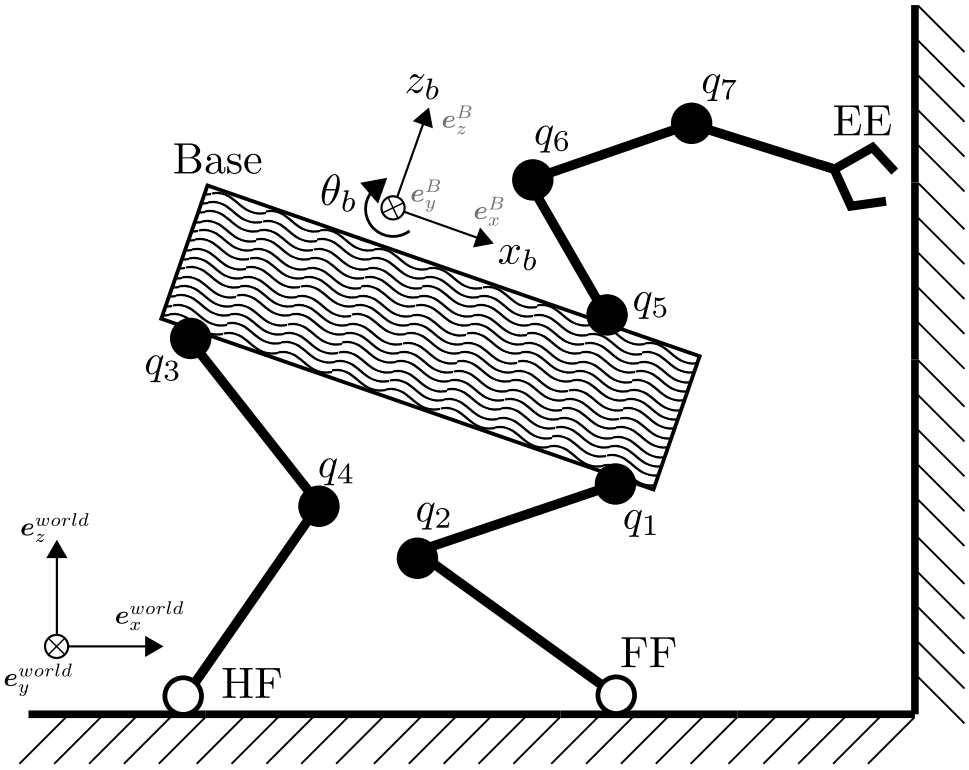

Figure 1: Schematic of a simplified legged robot in two dimensions. The morphology of the robot consists of a floating base (B) as well as the end-effector (EE), front foot (FF), and hind foot (HF) extremities.

1 Introduction

1.1 Setup

For this exercise, we provide MATLAB code which implements the simulation and visualization of the legged robot, as well as a skeleton for realizing the relevant controllers. Before we can run any experiments, however, we must first setup MATLAB's environment by running the `init_workspace.m` script. This will add all necessary files to MATLAB's path and is necessary for the code to work properly. The code is organized as follows:

- `simulate.m`: the *main* script that launches the visualization (if it's not open already) and configures and runs experiments. We will use this to run and test every controller in the next section.
- `controllers/`: the controllers to be implemented as MATLAB functions.
- `dynamics/`: contains the forward dynamics of the robot standing on the floor as well as a compliant contact model for the interactions with the wall.
- `common/`: a set of utilities used by the simulation and the controllers.
- `models/`: scripts for generating the kinematic and dynamic model of the system using the *proNeu* toolbox as well as the resulting MATLAB classes in binary form.
- `proneu/`: a MATLAB-based toolbox used for generating models of rigid multibody physics using both symbolic and numerical computations.

The exercises will essentially involve *modifying* the `floating_base_control.m` and `hybrid_force_motion_control.m` implementation files found in the `controllers/` directory, and running numerical simulation experiments using the `simulate.m` script. Fig. 2 demonstrates what the visualization should look like if the workspace has been set-up correctly.

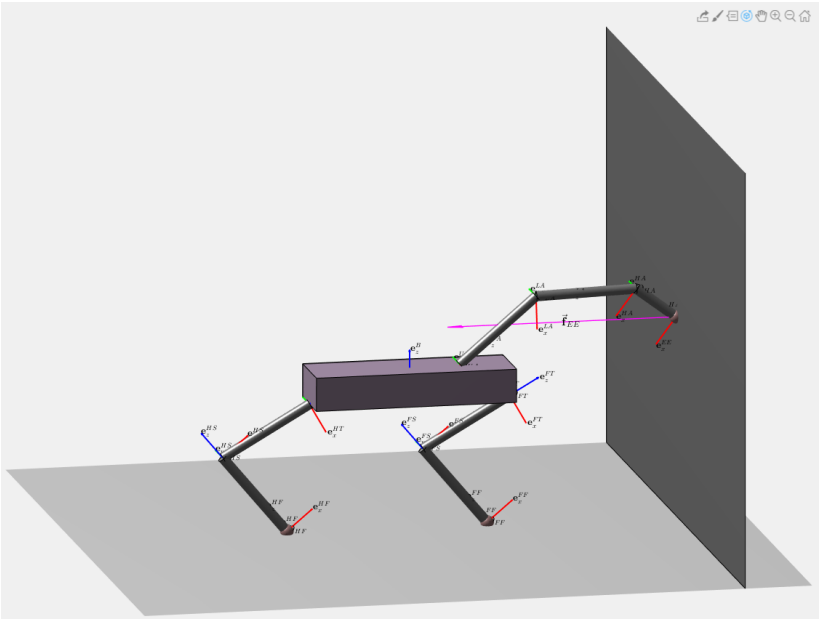

Figure 2: Rendering of the legged robot system interacting with the wall in MATLAB when running the `simulation.m` script.

1.2 Modelling

The legged robot, depicted in Fig. 1, is a planar simplification of quadrupedal systems such as ANYmal and Spot-Mini equipped with a manipulator. As we restrict motion to the X-Z plane, we model each pair of front and hind legs by a single articulated limb, which support a free-floating base. Additionally, we attach an articulated arm limb for manipulation tasks. Thus, the system has $n_q = 10$ total Degrees-of-Freedom (DoFs), $n_j = 7$ controllable (i.e. actuated) joints. The state of the system is defined by the vector of generalized coordinates

$$\mathbf{q} := \begin{bmatrix} \mathbf{q}_b \\ \mathbf{q}_j \end{bmatrix} , \mathbf{q}_b := \begin{bmatrix} x_B \\ z_B \\ \theta_B \end{bmatrix} , \mathbf{q}_j := \begin{bmatrix} q_1 \\ \vdots \\ q_7 \end{bmatrix} . \tag{1}$$

The Equations-of-Motion (EoM) of this system can therefore be written as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{S}^\top \boldsymbol{\tau}_j + \mathbf{J}_{\mathrm{FF}}(\mathbf{q})^\top \mathbf{f}_{\mathrm{FF}} + \mathbf{J}_{\mathrm{HF}}(\mathbf{q})^\top \mathbf{f}_{\mathrm{HF}} + \mathbf{J}_{\mathrm{EE}}(\mathbf{q})^\top \mathbf{f}_{\mathrm{EE}} , \tag{2}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n_q \times n_q}$ is the generalized inertia matrix, $\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n_q}$ is the vector of generalized non-linear forces, $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^{n_q}$ is the vector of generalized gravity forces, $\mathbf{S} \in \mathbb{R}^{n_q \times n_j}$ is the selection matrix, $\mathbf{J}_{\mathrm{FF}}(\mathbf{q})$, $\mathbf{J}_{\mathrm{HF}}(\mathbf{q})$, $\mathbf{J}_{\mathrm{EE}}(\mathbf{q})$ (all $\in \mathbb{R}^{3 \times n_q}$) are the *geometric* Jacobian matrices for the front and hind feet, and EE respectively, and $\mathbf{f}_{\mathrm{FF}}$, $\mathbf{f}_{\mathrm{HF}}$, and $\mathbf{f}_{\mathrm{EE}}$ are the corresponding wrenches applied at the feet and end-effector by the environment, respectively.

Please also take note of the following:

1. The motion of the base as well as all task spaces, have three degrees of freedom, namely x and z translation as well as single rotation about the y axis.
2. The resulting EoM expresses a system of n_q equations.
3. Due to the restriction of motion to the X-Z plane, we use $\dot{\mathbf{q}}$ instead of $\mathbf{u}$ for the generalized velocities, as the two are identical in this case.
4. All Cartesian quantities such as the pose, velocities and wrenches are reduced to $\mathbb{R}^2$ since no motion exists along the Y-axis.
5. As the joint torques $\boldsymbol{\tau}_j \in \mathbb{R}^{n_j}$ cannot directly influence the degrees of freedom of the base, the selection matrix is defined as $\mathbf{S} = \begin{bmatrix} \mathbf{0}_{n_j \times 3} & \mathbb{I}_{n_j \times n_j} \end{bmatrix}$.
6. We assume that the actuators at the joints are *perfect torque sources*, therefore, $\boldsymbol{\tau}_j = \boldsymbol{\tau}_j^*$, where $\boldsymbol{\tau}_j^*$ is the commanded torque generated by a controller.
7. We also define the constraint Jacobian $\mathbf{J}_c \in \mathbb{R}^{6 \times n_q}$ and respective constraint force vector $\mathbf{f}_c \in \mathbb{R}^6$, by simply stacking the geometric Jacobian matrices and forces of the feet respectively as $\mathbf{J}_c := \begin{bmatrix} \mathbf{J}_{\mathrm{FF}}^\top & \mathbf{J}_{\mathrm{HF}}^\top \end{bmatrix}^\top , \mathbf{f}_c := \begin{bmatrix} \mathbf{f}_{\mathrm{FF}}^\top & \mathbf{f}_{\mathrm{HF}}^\top \end{bmatrix}^\top$
8. Even though $\mathbf{J}_c \in \mathbb{R}^{6 \times n_q}$ and $\mathbf{f}_c \in \mathbb{R}^6$, in the optimization of the controller we may consider including *only* the linear components if we do not care to optimize the rotational motions and moments.

1.3 Control

Our objective is to design two model-based dynamic feedback controllers for tracking desired motions of the base and end-effector as well as forces for the latter. Our formulation, outlined in Fig. 3 in block-diagram form, consists of the following elements:

1. We assume that we are given instantaneous position and velocity references, in task-space, for the base and end-effector as well as 2D desired forces for the latter.

2. We employ HO, formulated using QPs to compute optimal torques, given a set of carefully selected costs and constraints[1].

3. The controllers will output a vector of desired joint torques $\boldsymbol{\tau}_j^*$ which we assume to be able to apply directly at the joints.

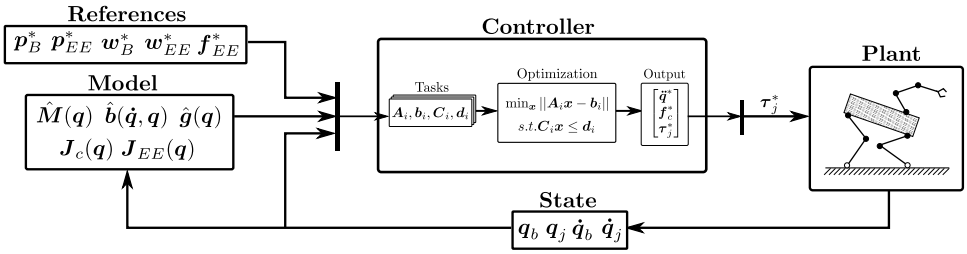

Figure 3: Schematic overview of the system: motion and force references together with estimates of the model parameters and state measurements are fed into the controller, which in turn uses these to formulate the task matrices. The resulting output of the optimization is used to select the joint-torque commands to the actuators of the plant.

1.4 Hierarchical Optimization

A detailed description of formulating Hierarchical Optimization (HO) problems as Quadratic Programs (QPs) can be found in Section 3.10.4 of the lecture notes. We briefly recapitulate the key elements of such a formulation in order to clarify how we will design controllers using this technique. Using the notation from the lecture, for each task i, we formulate a QP in the form

$$\min_{\mathbf{x}} ||\mathbf{A}_i\mathbf{x} - \mathbf{b}_i||_2 \tag{3}$$

$$\mathbf{C}_i\mathbf{x} \leq \mathbf{d}_i \tag{4}$$

Don't panic though, the hierarchical optimization procedure is already provided in the `common/hopt.m` MATLAB function. Thus, the actual work in this exercise will merely involve specifying numerical values for matrices $\mathbf{A}_i$ and vectors $\mathbf{b}_i$. The inequality constraint matrices and vectors $\mathbf{C}_i, \mathbf{d}_i$ are the same for each task in the hierarchy and are already implemented in each controller file.

[1] Compared to classical approaches based on direct Inverse Dynamics Control (IDC), this approach presents the advantage that using constrained optimization enables us to enforce constraints on the motion, forces and torques of the system, so to respect the physical limits of the available actuation.

2 Exercises

Floating-Base Motion Control

Exercise 2.1

The objective in this exercise is to implement a controller to track Cartesian pose and velocity references $\mathbf{p}_B^*$ and $\mathbf{w}_B^*$ for the base of the robot. These references should result in the base moving in a circular trajectory in the X-Z plane, while maintaining a constant attitude parallel with the floor.

The vector of decision variables for this HO are $\mathbf{x} = [\ddot{\mathbf{q}}^\top, \mathbf{f}_{c,xz}^\top, \boldsymbol{\tau}_j^\top]^\top$, where $\mathbf{f}_{c,xz}$ denotes the linear force components in the X-Z plane. You may assume that $\mathbf{f}_{\text{EE}} \equiv \mathbf{0}$, and that the floor does not introduce reaction moments, i.e. $\mathbf{f}_{\text{FF},\,\theta} = \mathbf{f}_{\text{HF},\,\theta} = \mathbf{0}$. The equations of motion for this particular setup can thus be written as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{S}^\top \boldsymbol{\tau}_j + \mathbf{J}_c(\mathbf{q})^\top \mathbf{f}_c \ . \tag{5}$$

This task requires the following hierarchy of tasks:

1. (TODO) Fulfill equations of motion
2. (TODO) Ensure feet remain stationary on the ground
3. (TODO) Move the body according to the reference trajectory
4. (TODO) Keep the end-effector at a fixed point in space
5. Maintain close proximity to a nominal configuration
6. Minimize joint torque
7. Minimize contact force at the feet

As noted by the TODO, only the four higher-level tasks need to be implemented and the lower-level ones are already provided. The relevant implementation file for this exercise is `controllers/floating_base_control.m`.

Hint: Task no. 2-4 can be formulated as task space motions with acceleration

$$\dot{\mathbf{w}}_i^* = \mathbf{J}_i \ddot{\mathbf{q}} + \dot{\mathbf{J}}_i \dot{\mathbf{q}} \tag{6}$$

for some task i, where $\mathbf{w}_i^*$ can be designed to have some form depending on the desired behavior.

Hybrid Force-Motion Control

Exercise 2.2

The objective of this exercise is to extend and adapt the formulation of the previous, but now we wish to also control the interaction force acting between the end-effector and the wall. Essentially, we want to use the end-effector to additionally push against a wall at a fixed spot. We therefore extend our vector of decision variables to $\mathbf{x} = [\ddot{\mathbf{q}}^\top, \mathbf{f}_{c,xz}^\top, \boldsymbol{\tau}^\top, \mathbf{f}_{\text{EE}}^\top]^\top$.

With the addition of end-effector forces at play, the EoM now become

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{S}^\top \boldsymbol{\tau} + \mathbf{J}_c(\mathbf{q})^\top \mathbf{f}_c + \mathbf{J}_{\text{EE}}(\mathbf{q})^\top \mathbf{f}_{\text{EE}} \ . \tag{7}$$

The optimization objectives must now comprise the following hierarchy of tasks:

1. (TODO) Fulfill equations of motion
2. (TODO) Ensure feet remain stationary on the ground
3. (TODO) Apply a force against the wall along the x direction
4. (TODO) Keep the end-effector at a fixed position on the wall in the z direction
5. (TODO) Move the body according to the reference trajectory
6. Maintain close proximity to a nominal configuration
7. Minimize joint torque
8. Minimize contact force at the feet

Again, only the tasks marked with TODO need to be implemented and the rest are provided in `controllers/hybrid_force_motion_control.m`.

Hint: Although we only want to control the end-effector force in the X-axis, we still need to include the Z-axis as well as the torque about Y-axis into the optimization task to ensure that they are as close to zero as possible. If we don't do this, then the optimizer could produce arbitrarily large values for $f_{EE,z}$ and $f_{EE,\theta}$.

Analysing the Optimization Scheme

We will now analyse the HO formulation of the previous exercises in order to understand how adding or removing terms may effect the final behavior of the robot. You can relax though, this exercise only requires thinking and writing, so no code needs to be implemented[2].

Exercise 2.3

What is the dimensionality of the vector of decision variables x in Exercise 2.1, and how many equations in total do the optimization objectives try to enforce?

Exercise 2.4

If we would observe slippage (i.e. sliding) occurring at the contacts between feet and terrain (i.e. non-zero $\mathbf{w}_c$), what kind of task (acting on motion or forces) could we add or change, to reduce this effect?

Exercise 2.5

So why is prioritization useful? Consider the same tracking problem, but we instead option to use a simpler constrained QP without prioritization (or HO with all task with the same priority). Name one case in which this would not work for such a problem.

Exercise 2.6

So why are hard constraints necessary? Consider again, the same tracking problem, but we instead option to use a very simple unconstrained QP, where constraints can be included as additional cost terms. Name one case in which this would not work well.

[2]We recommend modifying the tasks further to experiment with different terms.

3 Further Reading

For the interested reader, we highly studying some of our original works [1, 2] on the topic of whole-body control as applied to the quadrupedal robot ANYmal. Given the material covered in the lecture and in this exercise, it should become clear how applying HO to realize inverse-dynamics-based motion controllers can be used for such complex dynamic systems.

Moreover, we also provide the `models/leggedrobot.m` script, which can be used to re-generate the kinematic and dynamic model of the legged robot. It is implemented using the `proNeu` MATLAB toolbox for modelling rigid multi-body systems using both symbolic and numerical computations. The toolbox, provided in the `proneu/` directory, has a set of examples which can be used for modelling and control of different problems in robotics and control. The `proneu/scripts/test_models.m` script provides a demo of all provided examples.

References

[1] C. Dario Bellicoso, C. Gehring, J. Hwangbo, P. Fankhauser, and M. Hutter. Perception-less terrain adaptation through whole body control and hierarchical optimization. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 558–564, Nov 2016.

[2] C. Dario Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter. Dynamic locomotion and whole-body control for quadrupedal robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3359–3365, Sep. 2017.