Robotic Systems II: Homework I

Instructor: Konstantinos Chatzilygeroudis (costashatz@upatras.gr)

October 20 2024

Introduction

The homework concerns the modeling and control of a simplified robot and is split into three parts/sub-questions: 1) modeling and discretization of the system, 2) implementation of a generic Quadratic Programming (QP) solver using the Alternating Direction Method of Multipliers (ADMM), and 3) controlling the system via the QP solver.

1 Modeling (20%)

For this homework we are going to use a simplified one mass system that "emulates" the standing phase of a simplified hopper robot. The system can be seen in Fig. 1. The system "lives" in the 2D plane, and has two control inputs: a force, f, applied between the body and the ground, and an another force, τ, applied to the body perpendicular to the first force.
The state of the system is as follows:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} \in \mathbb{R}^4$$

And the control inputs:

$$\boldsymbol{u} = \begin{bmatrix} f \\ \tau \end{bmatrix} \in \mathbb{R}^2$$

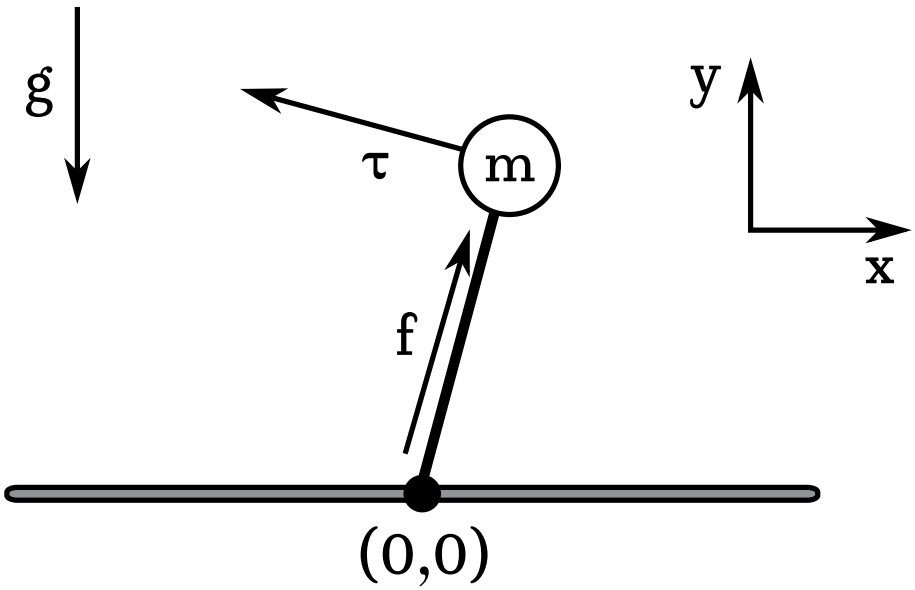

Figure 1: Visualization of the described system

The continuous dynamics of the system are given by the following equations:

$$f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \frac{r_x f - r_y \tau}{m} \\ \frac{r_y f + r_x \tau}{m} - g \end{bmatrix} \in \mathbb{R}^4$$

where $g = 9.81$ is the gravity, $m = 5\,kg$ is the body mass and $\boldsymbol{r} = \begin{bmatrix} r_x \\ r_y \end{bmatrix} = \frac{\begin{bmatrix} x & y \end{bmatrix}^T}{\sqrt{x^2+y^2}}$.

In this part, you are asked to:

1. Write down a function in Python called `dynamics()` that takes as input the state $\boldsymbol{x} \in \mathbb{R}^{4\times 1}$ and controls $\boldsymbol{u} \in \mathbb{R}^{2\times 1}$ and returns the $\dot{\boldsymbol{x}} \in \mathbb{R}^{4\times 1}$.
2. Create a simple visualization of the system (e.g. by using the `matplotlib` library); you are free to use any library for the visualization
3. Discretize the system using:
   (a) Euler integration
   (b) Semi-Implicit Euler integration
   (c) Runge-Kutta 4th Order integration
   (d) Midpoint integration:

$$\begin{aligned} \boldsymbol{x}_m &= \boldsymbol{x}_k + f(\boldsymbol{x}_k, \boldsymbol{u}_k)\frac{dt}{2} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + f(\boldsymbol{x}_m, \boldsymbol{u}_k)dt \end{aligned}$$

4. Create a simulation loop that works with any of the above integrators; **the initial state of the system should be** $\boldsymbol{x}_0 = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$
5. Write a simple PD-controller for the position of the body (e.g. to stay in place) and use it with the simulation loop
6. Compare the different integrators

For all the above, You are allowed to use only the native Python and the `numpy` library.

2 Quadratic Programming via the Alternating Direction Method of Multipliers (40%)

2.1 Quadratic Programming

Let's first remind us the Quadratic Programming (QP) problem.

$$\begin{aligned} \min_{\boldsymbol{x}} f(\boldsymbol{x}) &= \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \\ \text{s.t. } \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{C}\boldsymbol{x} - \boldsymbol{d} &\leq \boldsymbol{0} \end{aligned}$$

where $\boldsymbol{x}, \boldsymbol{q} \in \mathbb{R}^N$, $\boldsymbol{Q} \succ 0 \in \mathbb{R}^{N\times N}$, $\boldsymbol{A} \in \mathbb{R}^{M\times N}$, $\boldsymbol{b} \in \mathbb{R}^M$, $\boldsymbol{C} \in \mathbb{R}^{P\times N}$ and $\boldsymbol{d} \in \mathbb{R}^P$. We have N optimization variables, M equality constraints and P inequality constraints. Let's first write down the Lagrangian:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \boldsymbol{\mu}^T(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d})$$

And the **KKT Conditions:**

$$
\begin{aligned}
&1)\ \boldsymbol{Qx} + \boldsymbol{q} + \boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{C}^T\boldsymbol{\mu} = 0 \\
&2)\ \boldsymbol{Ax} - \boldsymbol{b} = \mathbf{0} \\
&3)\ \boldsymbol{Cx} - \boldsymbol{d} \leq \mathbf{0} \\
&4)\ \boldsymbol{\mu} \geq \mathbf{0} \\
&5)\ \boldsymbol{\mu}^T(\boldsymbol{Cx} - \boldsymbol{d}) = \mathbf{0}
\end{aligned}
$$

2.2 Task

Your task is to read the paper entitled "OSQP: An Operator Splitting Solver for Quadratic Programs" by *Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd* (Sections 1 up to and including 3), and implement the generic QP optimizer, called OSQP, in Python. In essence, you should implement Algorithm 1 from the paper (you can find the paper here: `https://arxiv.org/pdf/1711.08013`).

You are allowed to use only the native Python and the `numpy` library.

3 Control via QP (40%)

In this part, we are going to use our QP solver to control the system in Sec. 1. We are going to make this little "guy" dance! But first let's talk about how can we control this system with a QP. The main idea is to create a QP problem that optimizes for one step of the simulation. In other words, we will use the model of the system to predict what will happen one time-step in the future and take the best action according to our desired target.

If we are to control this system, we need to put its dynamics as constraints in the QP! **But**, QP optimizers accept constraints that are linear with respect to the optimization variables. The dynamics of our system are not linear. What can we do?

A popular and effective "trick" is to replace the position/velocity parts of the state with the acceleration parts and define the optimization variables for the QP as $\boldsymbol{z} = \begin{bmatrix}\ddot{x} & \ddot{y} & f & \tau\end{bmatrix}^T$. The intuition is that we give acceleration profiles as target to the QP optimizer and it gives us the control inputs that we need to put into the system such that we can follow them in the system. And what we gain is that now the dynamics of the system are linear with respect to the optimization variables. *Note that for the QP solver, $x, y, \dot{x}, \dot{y}$ are fixed values*!

The task for this subproblem is to use your QP solver (developed in Sec. 2) to make the body of the system follow **sinusoidal trajectories** in both x and y. You can define any sinusoidal trajectory: be creative and create a nice little dance routine. **The initial state of the system should be** $\boldsymbol{x}_0 = \begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}^T$. You are allowed to use only the native Python and the `numpy` library.

Note: if your QP solver is not correctly implemented, you are allowed to use a QP library (e.g., `ProxSuite`) and still get 3/4 of the points, if everything else is correct, for this exercise.

4 Deliverables

- Detailed report. In your report, you should also explain the OSQP algorithm that you implemented.
- 3 python files with commented code (one for each subquestion)[1]

Bonus Points

There is a 5% bonus if one derives the equations of the system (exactly as given), $f(\boldsymbol{x}, \boldsymbol{u})$, analytically.

[1] Jupyter notebooks are accepted as well.