Robotic Systems II: Homework I

Instructor: Konstantinos Chatzilygeroudis (`costashatz@upatras.gr`)

October 11 2025

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:

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

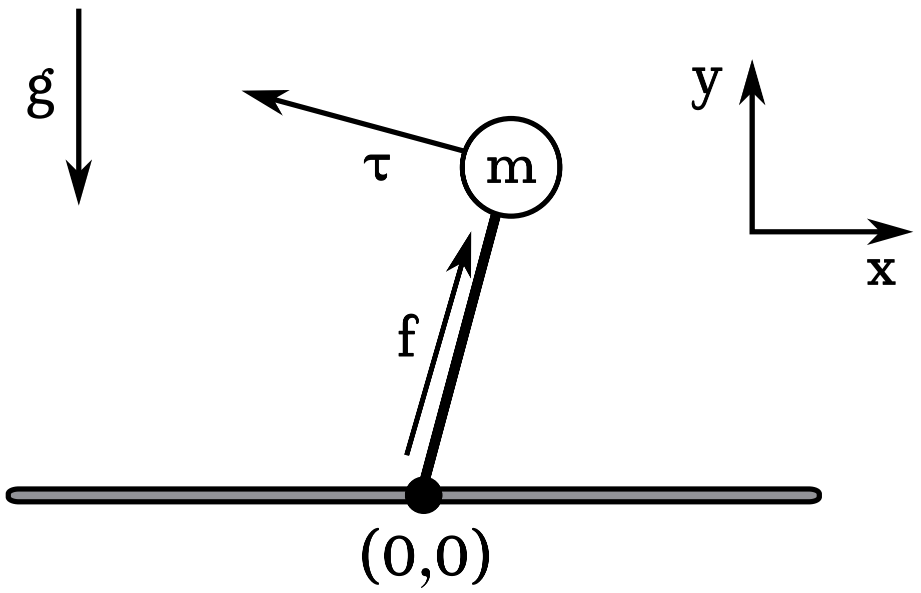

Figure 1: Visualization of the described system

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:

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

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 a Primal-Dual Augmented Lagrangian Method (40%)

2.1 Quadratic Programming

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

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

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.

2.2 Primal-Dual Augmented Lagrangian Method with Slack Variables

To deal with inequality constraints in a differentiable way, we introduce a nonnegative slack variable $\boldsymbol{s} \in \mathbb{R}^P_{\geq \mathbf{0}}$ such that

$$\boldsymbol{C}\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{d}, \qquad \boldsymbol{s} \geq \mathbf{0}.$$

This transforms the inequality-constrained QP into an equality-constrained one with simple bounds:

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

Let $\boldsymbol{\lambda} \in \mathbb{R}^M$ and $\boldsymbol{\mu} \in \mathbb{R}^P_{\geq \mathbf{0}}$ denote the Lagrange multipliers for the equality and inequality constraints, respectively. For a penalty parameter $\rho > 0$, we define the **augmented Lagrangian**

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{s}; \boldsymbol{\lambda}, \boldsymbol{\mu}) = \tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \tfrac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \boldsymbol{\mu}^T(\boldsymbol{C}\boldsymbol{x} + \boldsymbol{s} - \boldsymbol{d}) + \tfrac{\rho}{2}\|\boldsymbol{C}\boldsymbol{x} + \boldsymbol{s} - \boldsymbol{d}\|_2^2. \quad (1)$$

Primal updates. Given multipliers $(\boldsymbol{\lambda}^k, \boldsymbol{\mu}^k)$ and ρ, we minimize $\mathcal{L}_\rho$ with respect to $(\boldsymbol{x}, \boldsymbol{s})$ (subject to $\boldsymbol{s} \geq \mathbf{0}$). The first-order optimality conditions are

$$\mathbf{0} = \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q} + \boldsymbol{A}^T\big(\boldsymbol{\lambda}^k + \rho(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\big) + \boldsymbol{C}^T\big(\boldsymbol{\mu}^k + \rho(\boldsymbol{C}\boldsymbol{x} + \boldsymbol{s} - \boldsymbol{d})\big), \quad (2)$$

$$\mathbf{0} = \boldsymbol{\mu}^k + \rho(\boldsymbol{C}\boldsymbol{x} + \boldsymbol{s} - \boldsymbol{d}), \qquad \boldsymbol{s} \geq \mathbf{0}. \quad (3)$$

From the second equation, the slack update is explicit:

$$\boxed{\boldsymbol{s}^{k+1} = \Pi_{\geq \mathbf{0}}\big(\boldsymbol{d} - \boldsymbol{C}\boldsymbol{x}^k - \tfrac{1}{\rho}\boldsymbol{\mu}^k\big),} \quad (4)$$

where $\Pi_{\geq \mathbf{0}}(\cdot)$ denotes projection onto the nonnegative orthant.

Substituting this relation into (2) gives the linear system:

$$\boxed{(\boldsymbol{Q} + \rho\boldsymbol{A}^T\boldsymbol{A} + \rho\boldsymbol{C}^T\boldsymbol{C})\,\boldsymbol{x}^{k+1} = -\,\boldsymbol{q} - \boldsymbol{A}^T(\boldsymbol{\lambda}^k - \rho\,\boldsymbol{b}) - \boldsymbol{C}^T\big(\boldsymbol{\mu}^k - \rho(\boldsymbol{d} - \boldsymbol{s}^{k+1})\big).} \quad (5)$$

Dual updates. After updating $(\boldsymbol{x}^{k+1}, \boldsymbol{s}^{k+1})$, the multipliers are updated as:

$$\boxed{\begin{aligned} \boldsymbol{\lambda}^{k+1} &= \boldsymbol{\lambda}^k + \rho(\boldsymbol{A}\boldsymbol{x}^{k+1} - \boldsymbol{b}), \\ \boldsymbol{\mu}^{k+1} &= \Pi_{\geq \mathbf{0}}\big(\boldsymbol{\mu}^k + \rho(\boldsymbol{C}\boldsymbol{x}^{k+1} + \boldsymbol{s}^{k+1} - \boldsymbol{d})\big). \end{aligned}} \quad (6)$$

Algorithm.

Algorithm 1: Primal–Dual Augmented Lagrangian Method (with Slack Variables)

```
   Input: $Q, q, A, b, C, d$, penalty $\rho > 0$, tolerances $\varepsilon_{\text{feas}}, \varepsilon_{\text{stat}}$
   Output: Approximate solution $(x, s, \lambda, \mu)$
1  Initialize $x^0$, $s^0 = \Pi_{\geq 0}(d - Cx^0)$, $\lambda^0 = 0$, $\mu^0 = 0$;
2  while not converged do
3      $s^{k+1} \leftarrow \Pi_{\geq 0}(d - Cx^k - \frac{1}{\rho}\mu^k)$;
4      Solve $(Q + \rho A^T A + \rho C^T C)x^{k+1} = -q - A^T(\lambda^k - \rho b) - C^T(\mu^k - \rho(d - s^{k+1}))$;
5      $\lambda^{k+1} \leftarrow \lambda^k + \rho(Ax^{k+1} - b)$;
6      $\mu^{k+1} \leftarrow \Pi_{\geq 0}(\mu^k + \rho(Cx^{k+1} + s^{k+1} - d))$;
7      Check KKT conditions: $\|Ax^{k+1} - b\|_\infty$, $\|Cx^{k+1} + s^{k+1} - d\|_\infty$, and stationarity
         $\|Qx^{k+1} + q + A^T\lambda^{k+1} + C^T\mu^{k+1}\|_\infty$;
8      if residuals $< \varepsilon_{\text{feas}}, \varepsilon_{\text{stat}}$ then
9          break
10     end
11     if feasibility stagnates then
12         $\rho \leftarrow \tau\rho$ ;                    // Penalty increase, $\tau \in [5, 10]$
13     end
14 end
```

Why this method is *primal–dual.* At each iteration, the algorithm simultaneously updates:

- **Primal variables** $(\boldsymbol{x}, \boldsymbol{s})$ by minimizing the augmented Lagrangian $\mathcal{L}_\rho$ for fixed $(\boldsymbol{\lambda}, \boldsymbol{\mu})$, thus improving stationarity and primal feasibility;
- **Dual variables** $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ by performing gradient ascent on $\mathcal{L}_\rho$, enforcing feasibility and complementarity.

The method therefore searches for a *saddle point* of the augmented Lagrangian:

$$(\boldsymbol{x}^\star, \boldsymbol{s}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star) = \arg \min_{\boldsymbol{x},\, \boldsymbol{s} \geq \mathbf{0}} \arg \max_{\boldsymbol{\lambda},\, \boldsymbol{\mu} \geq \mathbf{0}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{s}; \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

This coupling between primal minimization and dual maximization gives the algorithm its **primal–dual character**: both sides of the optimization problem evolve in concert toward KKT optimality.

2.3 Task

Your task is to implement the Primal-Dual ALM as described above (you are not obliged to follow Algo. 1 exactly; it is merely given as a guideline). 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 (preferably written in LaTeX). In your report, you should also explain (with your own words) the Primal-Dual ALM 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. There is a 5% bonus if you analytically derive all equations for the Primal-Dual ALM.

[1] Jupyter notebooks are accepted as well.