Control for Robotics: From Optimal Control to Reinforcement Learning

Assignment 2 Model-Based Iterative Linear Quadratic Control

General Information

Due date:	You can find the due date in the syllabus handout. Your solution must be submitted before 23h59 on the due date.
Submission:	Please submit your solution and the requested Matlab scripts (highlighted in `blue`) as a single PDF document. Both typed and scanned handwritten solutions are accepted. It is your responsibility to provide enough detail such that we can follow your approach and judge your solution. Students may discuss assignments. However, each student must code up and write up their solutions independently. We will check for plagiarism. The points for each question are shown in the left margin.

Introduction

In this assignment, we will assume that we have complete knowledge about the system dynamics and design model-based iterative linear quadratic controllers (ILQC) for two systems. This assignment includes two marked problems and one unmarked problem. In the first problem, we will consider a simple mobile robot model and design an ILQC controller that drives the mobile robot along a straight line. In the second problem, we will build on the components developed in the first problem and design controllers that enable a quadrotor to perform agile maneuvers when fulfilling tasks such as flying to a given point and/or passing through predefined via-points. The last, unmarked problem is a simple example that illustrates the connection between LQR and Model Predictive Control (MPC) for solving an optimal control problem.

Problem 2.1 Mobile Robot

We consider a mobile robot modeled as a unicycle (Figure 1). The goal is to design a controller that moves the robot along a straight line. The robot has two inputs v and ω, where v is a *predefined*, desired forward speed, and ω is the turning rate. We assume that there are no constraints on the inputs v and ω.

The continuous-time dynamic equations describing the robot behavior are given by

$$\begin{aligned} \dot{y}(t) &= v(t)\ \sin\big(h(t)\big) \\ \dot{h}(t) &= \omega(t), \end{aligned} \tag{1}$$

where y and h are the lateral position and heading of the robot, respectively, and $v(t) = \bar{v}$ is the desired constant forward speed. By controlling h to zero and y to a desired y_{goal}, we automatically make progress in the x-direction.

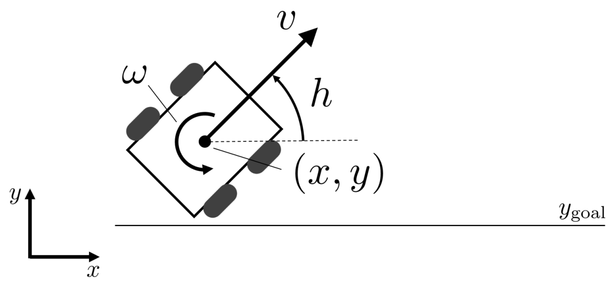

Figure 1: Setup of a unicycle-type robot moving along a straight line.

For the convenience of discussion, we denote $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ as the state and input of a system, respectively. The cost function to be minimized may be written as

$$\begin{aligned} J(x_0) = &\frac{1}{2} \left(x(T) - x_{\text{goal}}\right)^T Q_t \left(x(T) - x_{\text{goal}}\right) \\ &+ \int_0^T \frac{1}{2} \Big(\left(x(t) - x_{\text{goal}}\right)^T Q_s \left(x(t) - x_{\text{goal}}\right) + u(t)^T R_s u(t) \Big)\, dt, \end{aligned} \tag{2}$$

where $Q_t, Q_s \in \mathbb{R}^{n \times n}$ are symmetric positive semi-definite, $R_s \in \mathbb{R}^{m \times m}$ is symmetric positive definite, $x_{\text{goal}} \in \mathbb{R}^n$ is a predefined goal state, and T is the terminal time. For this problem, we have $x := [y, h]^T$, $u := \omega$, and $x_{\text{goal}} := [y_{\text{goal}}, 0]^T$, where y_{goal} is the desired lateral position of the robot.

LQR Controller Design

We first design an LQR controller. This controller will be used to initialize the ILQC algorithm.

5 (a) Find the equilibria of system (1) and linearize the system about an equilibrium $(x_{\text{eq}}, u_{\text{eq}})$. Express the linearized dynamics in terms of $\delta x = x - x_{\text{eq}}$ and $\delta u = u - u_{\text{eq}}$.

5 (b) Let $Q_t = Q_s = \text{diag}(1, 1)$ and $R_s = 20$ and assume infinite time horizon ($T \to \infty$). Use the Matlab function `lqr` to design an LQR controller that drives the robot to x_{goal}. Express the control law in the following form:

$$u(t) = \begin{bmatrix} \theta_{\text{ff}} & \theta_{\text{fb}} \end{bmatrix} \begin{bmatrix} 1 \\ x(t) \end{bmatrix} = \theta_{\text{ff}} + \theta_{\text{fb}}\, x(t), \tag{3}$$

where $\theta_{\text{ff}} \in \mathbb{R}^m$ and $\theta_{\text{fb}} \in \mathbb{R}^{m \times n}$ are controller parameters.

10 (c) Use the provided Matlab function `mobile_robot_sim(model, task, theta)` to simulate the response of system (1) driven by the designed LQR controller. **Note:** The function `mobile_robot_sim(model, task, theta)` simulates the system response using a time discretization of the system dynamics. The argument `theta` defines the controller parameter θ at each simulated time step. For this problem, `theta` should be a $(3 \times (N - 1))$ array with each column being $\theta = [\theta_{\text{ff}}, \theta_{\text{fb}}]^T$, where N is the number of simulated time steps. The argument `task` specifies the simulation setup, which includes the initial and goal states, the simulation start and end time, the discrete-time step size, as well as the cost function. Start with the default settings and make changes in the provided function `task_design()` when necessary. The argument `model` contains the basic model parameters used for simulation, and does not need to be modified. Please complete your implementation in `main_p1_lqr.m` and provide brief answers to the following questions:

(i) How does the result change as we vary Q_s and R_s? Support your answer with plots.

(ii) How does the controller performance vary when the initial state x_0 changes and why?

ILQC Controller Design

We now design an ILQC controller for the mobile robot line-following problem. We initialize the ILQC algorithm with the LQR control law obtained above and iteratively update the control law to minimize the prescribed cost function (2). For the ILQC design, we consider the time-discretized dynamics given below:

$$x_{k+1} = \begin{bmatrix} y_{k+1} \\ h_{k+1} \end{bmatrix} = \begin{bmatrix} y_k \\ h_k \end{bmatrix} + \delta t \begin{bmatrix} \bar{v}\sin(h_k) \\ \omega_k \end{bmatrix}, \tag{4}$$

where k is the discrete-time index, and δt is the discrete-time step size (in seconds).

Recall that, in each ILQC iteration, we first perform a forward pass and roll out the states and inputs based on the control law from the previous iteration. We denote the sequence of rollout states as $\{\bar{x}_k\}_{k=0}^{N}$ and the sequence of inputs as $\{\bar{u}_k\}_{k=0}^{N-1}$, where N is the number of simulation time steps. In the questions below, we will derive the key elements of the backward pass for updating the control law in each iteration.

2.5 (d) Linearize system (4) about an operating point $(\bar{x}_k, \bar{u}_k)$ and express the linearized system in terms of $\delta x_k = x_k - \bar{x}_k$ and $\delta u_k = u_k - \bar{u}_k$. Write a Matlab function that outputs the matrices of the linearized system for a given operating point $(\bar{x}_k, \bar{u}_k)$: `[`A_k, B_k`] = mobile_robot_lin(`$\bar{x}_k, \bar{u}_k, \delta t, \bar{v}$`)`.

2.5 (e) Time-discretize the cost function in (2) and express the cost function in the following form:

$$J(x_0) = g_N(x_N) + \delta t \sum_{k=0}^{N-1} g_k(x_k, u_k), \tag{5}$$

where $g_N(x_N)$ is the terminal cost and $g_k(x_k, u_k)$ is the stage cost. Derive second-order approximations of the terminal cost and the stage cost about $(\bar{x}_k, \bar{u}_k)$. The approximated costs should take the following forms:

$$g_N(x_N) \approx \bar{g}_N + q_N^T\,\delta x_N + \frac{1}{2}\delta x_N^T Q_N \delta x_N, \tag{6}$$

$$g_k(x_k, u_k) \approx \bar{g}_k + q_k^T\,\delta x_k + r_k^T\,\delta u_k + \frac{1}{2}\delta x_k^T Q_k \delta x_k + \frac{1}{2}\delta u_k^T R_k \delta u_k + \delta u_k^T P_k \delta x_k. \tag{7}$$

Write Matlab functions that compute the terminal and the stage costs in Eqns. (6) and (7): `[`$\bar{g}_N, q_N, Q_N$`] = terminal_cost_quad(`$Q_t, x_{\texttt{goal}}, \bar{x}_N$`)` and `[`$\bar{g}_k, q_k, Q_k, r_k, R_k, P_k$`] = stage_cost_quad(`$Q_s, R_s, x_{\texttt{goal}}, \delta t, \bar{x}_k, \bar{u}_k$`)`.

5 (f) The Bellman Equation (BE) for the value function at time step k can be written as

$$J_k^*(x_k) = \min_{u_k} \Big(g_k(x_k, u_k) + J_{k+1}^*(x_{k+1}) \Big). \tag{8}$$

Assuming the following approximation of the value function $J_{k+1}^*(x_{k+1})$,

$$J_{k+1}^*(x_{k+1}) \approx \bar{s}_{k+1} + \delta x_{k+1}^T s_{k+1} + \frac{1}{2}\delta x_{k+1}^T S_{k+1} \delta x_{k+1}, \tag{9}$$

and given the approximate stage cost $g_k(x_k, u_k)$ in Eqn. (7), derive an expression for the optimal δu_k^* at each time step. Express the updated control law $u_k = \bar{u}_k + \delta u_k^*$ in the following form:

$$u_k = \begin{bmatrix} \theta_{k,\text{ff}} & \theta_{k,\text{fb}} \end{bmatrix} \begin{bmatrix} 1 \\ x_k \end{bmatrix} = \theta_{k,\text{ff}} + \theta_{k,\text{fb}} x_k, \tag{10}$$

where $\theta_{k,\text{ff}} \in \mathbb{R}^m$ and $\theta_{k,\text{fb}} \in \mathbb{R}^{m\times n}$.

5 (g) Write down the equations for updating $\bar{s}_k, s_k, S_k$ from $\bar{s}_{k+1}, s_{k+1}, S_{k+1}$ at each time step. How should we initialize $\bar{s}_k, s_k, S_k$? **Hint:** Substitute the expression of δu_k^* back into the BE in Eqn. (8). The BE should hold for arbitrary δx_k when $J_k^*(x_k)$ is approximated by Eqn. (9).

5 (h) Based on parts (d)-(g), write a Matlab function that updates the control law at time step k: `[`$\theta_{k,\mathtt{ff}}, \theta_{k,\mathtt{fb}}, \bar{s}_k, s_k, S_k$`] = update_policy(`$A_k, B_k, \bar{g}_k, q_k, Q_k, r_k, R_k, P_k, \bar{s}_{k+1}, s_{k+1}, S_{k+1}, \bar{x}_k, \bar{u}_k$`)`.

5 (i) Write pseudocode that outlines the main steps of the ILQC algorithm. Make use of the Matlab functions defined above where appropriate.

5 (j) Complete the implementation of the ILQC algorithm for the mobile robot line-following problem in `main_q1_ilqc.m` and answer the following questions:

(i) Plot the state, input, and stage cost trajectories for two initial conditions $x_{0,1} = [0,\ 0]^T$ and $x_{0,2} = [0,\ \pi]^T$. Comment on the difference between the LQR controller and the ILQC controller.

(ii) How would the controllers' performance change when there is a mismatch between the model and the real system?

Problem 2.2 Quadrotor

In this problem, we will design controllers that allow a quadrotor to reliably reach a predefined goal state and/or pass through predefined via-points. The questions and scripts in this problem are adapted from [1].

The platform that inspired the numerical values in this problem is the AscTec Hummingbird [**?**]. The state x of this robot is defined by the position of its center of mass (p_x, p_y, p_z) and the body orientations (ϕ, θ, ψ) (roll, pitch, yaw), and their derivatives. The inputs u that control the system are the vertical thrust F_z and the moments (M_x, M_y, M_z) around each axis of a coordinate frame fixed to the quadrotor body. The continuous-time nonlinear system dynamics are given by

$$\dot{x} = f(x) + g(x)\,u, \tag{11}$$

where $x = [p_x, p_y, p_z, \phi, \theta, \psi, \dot{p}_x, \dot{p}_y, \dot{p}_z, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ and $u = [F_z, M_x, M_y, M_z]^T$. This system is underactuated with two degrees of underactuation—it has twelve states in total (or, six degrees of freedom) and four independent actuation inputs. In the quadrotor simulator provided for this problem, the actuator dynamics are neglected (i.e., the fact that a commanded force is not immediately achieved by the motors) due to the fast rotor dynamics, but the rotor saturations (i.e., the fact that each rotor's generated force has a lower and upper limit) are maintained. Aerodynamics such as turbulence and ground effects are not modeled.

For each question below, complete the implementation in the corresponding Matlab script and concisely summarize your observations. You are not expected to provide detailed derivations for this problem.

LQR Controller Design

Similar to Problem 2.1, we design an LQR controller to later initialize the ILQC algorithm.

5 (a) *LQR controller gains:* Compute the gains of an infinite-horizon LQR controller for the quadrotor system by filling in the corresponding part in the Matlab function `[LQR_Controller, Cost] = LQR_Design(Model,Task)`. The quadratic cost function is given by `Task.cost.Q_lqr` and `Task.cost.R_lqr`. **Note:** The continuous linearized model around an operating point is provided by `Model.Alin{1}(`x_{op}`, `u_{op}`, Model.param.syspar_vec)` and `Model.Blin{1}(`x_{op}`, `u_{op}`, Model.param.syspar_vec)`, where $(x_{\text{op}}, u_{\text{op}})$ are the state and input vectors defining the operating point, and `Model.param.syspar_vec` contains the system parameters that are given.

10 (b) *LQR controller design for reaching a goal state:* Design an LQR controller for reaching a defined goal state x_{goal} and complete the corresponding part in `[LQR_Controller, Cost] = LQR_Design(Model,Task)`. **Note:** The quadrotor controller structure includes feedforward inputs u_{ff} (not depended on the current state x) and feedback inputs u_{fb}. The controller requires these to be specified in the form $\theta_k \in \mathbb{R}^{(n+1)\times m}$, with the number of states $n = 12$ and the number of control inputs $m = 4$. For an infinite horizon formulation with one goal state, we have

$$
\begin{aligned}
u_k := \begin{bmatrix} F_{k,z} & M_{k,x} & M_{k,y} & M_{k,z} \end{bmatrix}^T &= u_{\text{ff}} + u_{k,\text{fb}} && (12)\\
&= u_{\text{ff}} + K(x_k - x_{\text{ref}}) && (13)\\
&= \begin{bmatrix} u_{\text{ff}} - K x_{\text{ref}} & K \end{bmatrix} \begin{bmatrix} 1 \\ x_k \end{bmatrix} && (14)\\
&= \theta_k^T x_{k,\text{aug}}. && (15)
\end{aligned}
$$

Each time step $k = 0, 1, ..., N-1$ requires such a $\theta_k \in \mathbb{R}^{(n+1)\times m}$ matrix. These matrices of each time step are layered to produce $\Theta \in \mathbb{R}^{(n+1)\times m \times (N-1)}$. For a time-varying controller where feedforward inputs and feedback gains change over time the individual θ_k change with k.

The goal of this task is to compute the matrix Θ to drive the system to the defined goal state `Task.goal_x`. Since, by definition, LQR regulates the system to zero, this goal state must be correctly included in the controller structure. When ready, run `main_p2_lqr.m` and observe the behavior.

(i) How does the controller behave for varying positions of `Task.goal_x`? Include plots for $x_{\text{goal}} = [10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$ and $t_{\text{goal}} = 10$.

(ii) What is the cause of the varying performance (e.g., model, cost function, etc.)?

10 (c) *LQR controller design with via-point:* After observing the behavior of the previous controller, it might be helpful to include an intermediate via-point `Task.vp1` that the controller should drive towards until `Task.vp_time` before aiming towards `Task.goal_x`. Redesign the matrix Θ to reflect this task, change `lqr_type = 'goal_state'` to `'via_point'` in `[LQR_Controller, Cost] = LQR_Design(Model,Task)` and fill in the missing part. Run `main_p2_lqr.m` to observe the behavior.

(i) How does a via-point change the behavior of the quadrotor with the LQR controller? Include plots for the case where $x_{\text{goal}} = [10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$, $t_{\text{goal}} = 10$, $x_{\text{vp}} = [5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$, and $t_{\text{vp}} = t_{\text{goal}}/3$. Compare the result with that in part (b).

(ii) What points are reachable now? Why is the system capable of reaching further away `Task.goal_x` states?

(iii) Defining via-points `Task.vp` and via-times `Task.vp_time` seem to have a positive impact on the system behavior and increase stability. Why and what are the disadvantages?

ILQC Controller Design

We now complete the implementation of an ILQC controller and experiment with the cost function to fulfill the task of passing through given via-points.

15 (d) *ILQC controller design for reaching a goal state:* Follow the steps derived in Problem 1 and fill in the missing code in `ILQC_Design.m`. Test the algorithm by running `main_p2_ilqc.m`. **Note:** The ILQC controller requires discretized, linearized system dynamics (A_k, B_k), whereas the quadrotor model supplies continuous, linearized system dynamics $(A_{\text{lin}}, B_{\text{lin}})$. A simple approximation between the

two is given as follows:

$$\delta\dot{x}(t) = A_{\text{lin}}\delta x(t) + B_{\text{lin}}\delta u(t) \tag{16}$$

$$\frac{\delta x_{k+1} - \delta x_k}{\delta t} = A_{\text{lin}}\delta x_k + B_{\text{lin}}\delta u_k \tag{17}$$

$$\delta x_{k+1} = (I + A_{\text{lin}}\delta t)\,\delta x_k + (B_{\text{lin}}\delta t)\,\delta u_k. \tag{18}$$

You may reuse code from Problem 2.1 where applicable.

(i) Simulate the system response with the ILQC controller and present the plots for the default setting. How do the trajectories of the LQR and ILQC controller differ? How do the costs compare?

(ii) Why does the ILQC perform better for distant and similarly well for close goal states?

10 (e) *ILQC controller design with via-point:* For your convenience, we provide the solution to the previous exercise in `Private/Design_functions/[ILQC/LQR]_Design_Solution.p`. To use these, append "_Solution" to the respective function calls in `main_p2_ilqc.m`.

The goal of this subproblem is to make the quadrotor pass through via-points as done previously for the LQR controller and compare the performance. This part of the exercise must be implemented in the `Cost_Design.m`. We are interested in passing through the via-point smoothly, allowing the ILQC algorithm to determine an optimal velocity on its own. Define a new final `Cost.h` and intermediate cost `Cost.l` including this via-point cost. Change `ilqc_type = 'goal_state'` to `'via_point'` and run `main_p2_ilqc.m`. Additionally, try ILQC for the more complex via-point `Task.vp2`. In contrast, observe how the LQR controller behaves with this via-point.

Hint: In contrast to the LQR controller, the ILQC controller includes via-points in the cost function in `Cost_Design.m`. The time-dependent via-point cost $g_{\text{vp}}(t)$ is added to the intermediate cost to drive the robot towards the via-point. It can be viewed as a ρ-steep, bell-shaped curve; it peaks at time t_{vp} when this via-point should be reached. For other times, the effect of this cost is negligible. The cost is implemented in `viapoint(.)` as follows:

$$g_{\text{vp}}(t) = (x - x_{\text{vp}})^T Q_{\text{vp}}(x - x_{\text{vp}})\sqrt{\frac{\rho}{2\pi}}\exp\left(-\frac{\rho}{2}(t - t_{\text{vp}})^2\right). \tag{19}$$

(i) How must the via-point weighting matrix `Q_vp` be chosen for the algorithm to determine the optimal via-point velocity on its own? Simulate the response of the system with the via-point added and submit the plots for the default task setting.

(ii) How can exact passing through the via-point be enforced and how can it be included only as a soft suggestion on top of the other performance criteria?

Additional Practice Problem (Not Marked)

Problem 2.3 Infinite- and Finite-Horizon Linear Quadratic Control

Consider the following first-order discrete-time system

$$x_{k+1} = 1.5x_k + u_k\,, \tag{20}$$

which is to be controlled using a linear quadratic controller that minimizes the quadratic objective function

$$J(x_0) = \sum_{k=0}^{\infty} x_k^2 + 10u_k^2\,. \tag{21}$$

(a) Determine the unconstrained optimal control law with respect to the infinite-horizon objective function.

(b) Consider the finite-horizon objective function

$$J(x_0) = \sum_{k=0}^{1}(x_k^2 + 10u_k^2) + p_N x_2^2\,. \tag{22}$$

Determine the value of p_N such that the optimal control policy for the infinite-horizon and finite-horizon objective function coincide.

(c) The finite-horizon objective function is minimized subject to the input constraints:

$$-0.5 \le u_k \le 1.0, \quad \forall\, k \in \mathbb{N}\,. \tag{23}$$

Assume that the constraints are satisfied for $k \in \{0, 1, 2\}$. Show that the constraints are satisfied for all future time steps $k > 0$ when applying the optimal control policy found in (a).

(d) Consider using this optimization problem in a receding horizon fashion as an MPC. Write down the MPC formulation given by the dynamics, finite-horizon cost, and input constraints. Is the problem convex? If so, what is the type of the optimization problem? Justify your answer.

References

[1] Jonas Buchli. *Course on Optimal and Learning Control for Autonomous Robots*, April 2015. Course Number 151-0607-00L, Swiss Federal Institute of Technology in Zurich (ETH Zurich). Accessed on: Mar. 07, 2020. [Online]. Available: http://www.adrlab.org/doku.php/adrl:education:lecture:fs2015.