OLCAR - Exercise 1 - ILQC
Handout: 17.03.2015
Due: 15.04.2013

Introduction

The goal of exercise 1 is to design an ILQC controller that is able to control a quadrotor for performing agile maneuvers such as flying to a defined point and/or passing through some predefined via-points. To fulfill these tasks, suitable cost functions are minimized based on the principle of optimality and the assumption of complete system model knowledge. The designed controllers in this exercise will be time-varying, linear feedback plus feedforward controllers.

In this exercise set we will assume complete knowledge about the system model, i.e. without any uncertainty about the system parameters. Having this model, we are able to design a model-based controller for stabilizing and controlling the robot. Later, in exercise 3, we will put aside the assumption of complete model knowledge. This means that the true system's parameters are unknown and we have just a biased estimation of them. However, we assume that we can still draw samples from the true system.

This in fact is a real scenario when we are dealing with an actual robot. Although often we have a model of the robot dynamics, the model is based on some assumptions and simplifications. Therefore the designed controller based on this model performs noticeably different on the real robot. In this case, we implement an adaption mechanism which adapts the model-based controller on fly.

Exercise 1 and 3 are trying to emulate such a scenario. In exercise 1 we are going to design a model-based controller based on a given model. Later, in exercise 3, we will assume that the given model is biased. In the case of the ILQC controller this means that the linearized model and the provided simulator will not comply with each other even at the first order approximation of the dynamics. In exercise 3, we are going to implement a learning algorithm which tries to adapt the biased model-derived controller to the unknown model of the simulator.

On the course website `http://www.adrl.ethz.ch/doku.php/adrl:education:lecture:fs2015` a skeleton of the matlab software can be downloaded. Please complete the missing parts and upload your `{*_Design.m}` files through the appropriate form on the website. Additionally upload a .pdf with the answers to the posed questions (no more than 3 sentences per question, max. 1 A4 page in total). All this material must be uploaded by **15.04.2013, 11:59 pm**.

Robotics platform

The main platform for the exercise 1 and 3 is a quadrotor as shown in Fig. 1. The state $\mathbf{x}$ of this robot is defined by its center of the mass positions x, y, z, the body orientations ϕ, θ, ψ (roll, pitch, yaw) and their derivatives. The inputs $\mathbf{u}$ that control the system are the vertical thrust F_z and the moments M_x, M_y, M_z

Figure 1: AscTec Hummingbird [1].

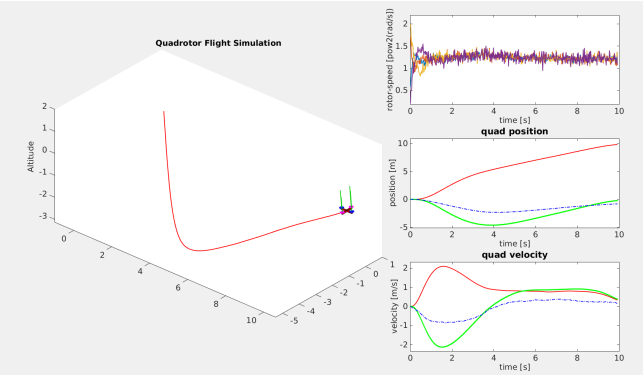

Figure 2: Simulated trajectory with via-point.

around each axis. The continuous-time nonlinear system dynamics are thus given by

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{u}) = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} \tag{1}$$

where $\mathbf{x} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ and $\mathbf{u} = [F_z, M_x, M_y, M_z]^T$.

As equation (1) indicates, this system is underactuated with two degree of underactuation. It has in total twelve states i.e. six degrees of freedom and its actuation degree is four. In the simulated quadrotor in these exercises, the actuator dynamics are neglected although the rotor saturations are still maintained. Also no aerodynamics such as turbulence effect or ground effect are modeled.

Software Structure

The main structure of the code is defined in `main_ex1.m`. The parts to implement are located in separate functions and highlighted by cell '%%' comments. A brief overview of the steps executed in the main function is given below:

- `Task = Task_Design()` defines an initial Task including start and goal state, time constraint and other requirements on a high level.
- `load('Quadcopter_Model.mat','Model')` defines the dynamics of the quadrotor (equation 1) including mass and thrust coefficients etc. It also provides functions such as `Model.Alin(x,u,...)` which returns the linearized-system-dynamics state-transition matrix $\mathbf{A}$ around point $(\mathbf{x}, \mathbf{u})$.
- `Task.cost = Cost_Design(Model.param.mQ,Task)` transforms the high level requirements of the specified `Task` into a cost function that reflects these intentions.

- `LQR_Design(Model,Task)` designs an LQR controller for the given `Task`.

- `ILQC_Design(Model,Task,Initial_Controller,@Quad_Simulator)` designs an ILQC controller for the given task and model. An initial controller for the first rollout and a simulator to generate the rollouts are specified.

- `sim_out = Quad_Simulator(Model,Task,Controller)` generates a trajectory of states and control inputs by applying the designed controller to the system. This trajectory can be visualized `Visualize(sim_out,...)` and/or plotted `Plot_Result(sim_out,...}` by commenting in the appropriate lines.

Problem 1

Before designing an ILQC controller we will try to accomplish this task with a standard LQR controller [2]. Please implement the solutions in `LQR_Design.m`.

1. **Calculate LQR feedback gains:** Design an infinite-horizon LQR controller. Since such a controller requires linearized system dynamics the linearization point (operating point) must be correctly chosen. The quadratic cost function is already given in `Task.cost`.

   Hint: The *continuous* linearized model of the system around the nominal state and input trajectories is provided by `Model.Alin{1}(x,u,Model.param.syspar_vec)` as a function of state and control input vectors

   $$\delta\dot{\mathbf{x}} = \mathbf{A}_{lin}\delta\mathbf{x} + \mathbf{B}_{lin}\delta\mathbf{u} \tag{2}$$

   where $\mathbf{A}_{lin} = \frac{\partial \mathbf{F}}{\partial x}$, $\mathbf{B}_{lin} = \frac{\partial \mathbf{F}}{\partial u}$. In order to design the infinite-horizon LQR, you can either directly use the continuous-time algebraic Riccati equation (`help lqr`) or you can time-discretize the linearized model and then use the discrete-time algebraic Riccati equation (`help dlqr`). For sufficiently fine time-discretization both approaches will generate comparable results. If you choose the discretization approach use the sampling time defined in `Task.dt` which is sufficiently small for this task.

2. **Design LQR controller with one goal state:** The quadrotor controller structure allows to implement feedforward inputs $\mathbf{u}^{ff}$ (not depended on current state $\mathbf{x}$) and feedback inputs $\mathbf{u}^{fb}$. The controller requires these to be specified in form of $\boldsymbol{\theta}_n \in \mathbb{R}^{(1+n_x)\times n_u}$, with the number of states

$n_x = 12$ and the number of control inputs $n_u = 4$, defined as:

$$\begin{aligned} \mathbf{u}_n = (F_z, M_x, M_y, M_z)^T &= \mathbf{u}^{ff} + \mathbf{u}^{fb} \\ &= \mathbf{u}^{ff} + K(\mathbf{x} - \mathbf{x}_{ref}) \\ &= \begin{bmatrix} \mathbf{u}^{ff} - K\mathbf{x}_{ref} & K \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \\ &= \boldsymbol{\theta}_n^T \mathbf{x}_{aug} \end{aligned} \tag{3}$$

Each time step $n = 0, ...N$ requires such a $\boldsymbol{\theta}_n$ matrix. These matrices are layered for every time step n to produce $\boldsymbol{\theta} \in \mathbb{R}^{(1+n_x)\times n_u \times N}$. For a time-varying controller where feedforward inputs and feedback gains change over time (e.g. with a via-point) the individual $\boldsymbol{\theta}_n$ change over the dimension n.

The goal of this task is to correctly fill the matrix $\boldsymbol{\theta}$ 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_ex1.m` and observe the visualization.

Question 1: How does the controller behave with varying positions of the `Task.goal_x`?
Question 2: What is the cause of the varying performance (model, cost function,...)?
Question 3: How do changes in `Task.cost.Q_lqr` and `Task.cost.R_lqr` affect the behavior?

3. **Design LQR controller with a 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 $\boldsymbol{\theta}$ to reflect this task, change `lqr_type = 'goal_state'` to `'via_point'` and run `main_ex1.m` to observe the behavior.

   Question 4: How does a via-point change behavior of the LQR controller?
   Question 5: Why is the system capable of reaching further away `Task.goal_x` states?
   Question 6: Defining via points `Task.vp` and time `Task.vp_time` seem to have a positive influence on the system behavior and increase stability. What are the disadvantages?

Problem 2

In this problem you will design the actual ILCQ controller [4] as introduced in the script and compare it with the two previously designed LQR controllers for the same tasks. The ILQC controller must be initialized with a state and input trajectory, which we generate by the previously designed LQR controller `Initial_Controller`.

1. **Design the ILQC controller:** The implementation of this problem is done in `ILQC_Design.m`. Follow script section 1.6 to fill in the missing code. Test the algorithm by removing `return;` in `main_ILQC.m` and run it.

   Hint: The ILQC controller requires *discretized* linearized system dynamics $\mathbf{A}_n, \mathbf{B}_n$, whereas the quadrotor model supplies *continuous* linearized system dynamics $\mathbf{A}_{lin}, \mathbf{B}_{lin}$. A simple approximation between the two is given as follows:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}_{lin}\mathbf{x} + \mathbf{B}_{lin}\mathbf{u} \\ \Leftrightarrow \frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{\delta t} &= \mathbf{A}_{lin}\mathbf{x}_n + \mathbf{B}_{lin}\mathbf{u}_n \\ \Leftrightarrow \quad \mathbf{x}_{n+1} &= (\mathbf{I} + \mathbf{A}_{lin}\delta t)\mathbf{x}_n + (\mathbf{B}_{lin}\delta t)\mathbf{u}_n \end{aligned} \tag{4}$$

   Question 7: How do the trajectories of the LQR and ILQC controller differ? How do the costs compare?
   Question 8: Why does the ILQC perform better for distant and similarly well for close goal states?

2. **Include via-points for the ILQC controller to pass through:** 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_ex1.m`.

   The goal of this sub problem is to make the quadrotor pass through via-points as in the LQR controller and compare the performance. This part of the exercise must be implemented in done in the `Cost_Design.m`. We are interested in passing through the via-point in a smooth motion, 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_ex1.m`. Additionally, try ILQC with 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 $L_{vp}(t)$ is added to the intermediate cost to drive the robot towards the via-point. It can be seen as a ρ-steep bell shaped curve with it's peak at the time t_{vp} this via point should be reached. For other times the effect of this cost is negligible. For more information see [3]. The cost is implemented in `viapoint(.)` as follows:

$$L_{vp}(t) = \quad (\mathbf{x} - \mathbf{x}_{vp})^\top \mathbf{Q}_{vp} (\mathbf{x} - \mathbf{x}_{vp}) \cdot \sqrt{\frac{\rho}{2\pi}} e^{\left(-\frac{\rho}{2}(t - t_{vp})^2\right)} \tag{5}$$

Question 9: How must the via-point weighting matrix `Q_vp` be chosen for the algorithm to determine the optimal via-point velocity on its own?
Question 10: How can exact passing through the via point be enforced and how can it be included only as a soft suggestions on top of the other performance criteria?

References

[1] Quadcopter image source. `http://www.asctec.de/en/uav-uas-drone-products/asctec-hummingbird/#pane-0-0`. Accessed: 2015-02-30.

[2] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[3] Cedric de Crousaz, Farbod Farshidian, and Jonas Buchli. Aggressive optimal control for agile flight with a slung load. In *IROS 2014 Workshop on Machine Learning in Planning and Control of Robot Motion*, 2014.

[4] Cedric de Crousaz, Farbod Farshidian, and Michael Neunert Jonas Buchli. Unified motion control for dynamic quadrotor maneuvers demonstrated on slung load and rotor failure tasks. In *IEEE International Conference on Robotics and Automation*, 2015.