Model Predictive Control - EL2700

A Linear Model Predictive Controller for a Quadrotor

2020

Automatic Control
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan

Introduction

And finally, here we are. Up to now you have learned about system discretization and how the poles of the system influence its behavior; you have learned about continuous time optimization and how to create Linear Programming and Quadratic Programming optimization problems; you learned about infinite horizon LQR controllers and how terminal weights influence the system behavior; you learned about MPC and which strategies exist to eliminate steady-state error. Even with all the head-scratching around Python and CasADi, you learned how to formalize these methods in a general form that you could implement with open-source tools, and hopefully be useful for you in the future!

So now, we want to show you how powerful these tools are. In this project we propose you to implement a Linear MPC for a Quadrotor. From your submissions, we will choose one to implement on a real quadrotor that you will have the chance to see live in the lab session, the 2nd of October, 2020 (more information on time slots will come later).

Throughout this project we want you to re-use the code given during the assignments and appropriately adjust it for the system at hand. You should have now all the tools you need to implement an MPC in a real system.

Quadrotor Dynamics

We first have to model the system we have at hand. To this end, let there be an inertial frame and a body frame B, respectively associated with the workspace where the quadrotor navigates and to the vehicle pose. See Figure 1 for an a representation of the body frame B.

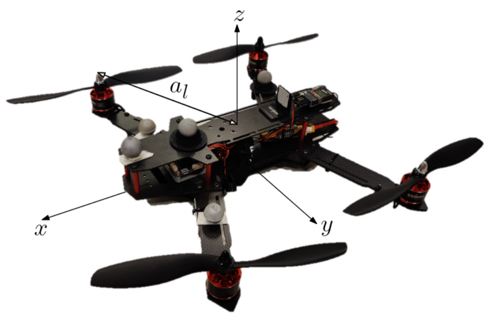

Figure 1: UAV body frame representation.

Given this representation, take the nonlinear dynamics of a quadrotor

$$\dot{\mathbf{p}} = \mathbf{v} \tag{1a}$$

$$\dot{\mathbf{v}} = \mathbf{R}_B(\boldsymbol{\alpha})\frac{\mathbf{f_t}}{m} + \mathbf{g}, \tag{1b}$$

$$\dot{\boldsymbol{\alpha}} = \mathbf{T}(\boldsymbol{\alpha})\boldsymbol{\omega}, \text{ and} \tag{1c}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{M}\boldsymbol{\omega}), \tag{1d}$$

where:

- $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T \in \mathbb{R}^3$ is the position of the quadrotor in the inertial frame;
- $\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T \in \mathbb{R}^3$ is the velocity of the quadrotor in the inertial frame;
- $\boldsymbol{\alpha} = \begin{bmatrix} \theta & \phi & \psi \end{bmatrix}^T \in \mathbb{R}^3$ is the vector containing the roll θ, pitch ϕ and yaw ψ angles of the quadrotor with respect to the inertial frame;
- $\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T \in \mathbb{R}^3$ the body angular velocity;

- $\mathbf{R}_B(\boldsymbol{\alpha}) \in \mathbb{SO}(3)$ the rotation matrix representing the attitude of the quadrotor with respect to the inertial frame, given by

$$\mathbf{R}_B(\boldsymbol{\alpha}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

;
- $\mathbf{f_t} = \begin{bmatrix} 0 & 0 & \mathrm{f}_z \end{bmatrix}^T, \mathrm{f}_z \in \mathbb{R}$ is the force vector with thrust along the Z axis of the vehicle;
- m is the mass of the vehicle;
- $\mathbf{g} = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T, g = 9.81$ is the gravity vector containing the gravity constant;
- $\mathbf{T}(\boldsymbol{\alpha})$ is the jacobian of the angular rates with respect to the inertial frame, given by

$$\mathbf{T}(\boldsymbol{\alpha}) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \tag{3}$$

- $\mathbf{M} = \mathrm{diag}(\begin{bmatrix} M_x & M_y & M_z \end{bmatrix}) \in \mathbb{R}^{3\times3}$ is the inertia matrix of the quadrotor;
- $\boldsymbol{\tau} = \begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix}^T \in \mathbb{R}^3$ is the torque on the body frame of the quadrotor;
- $a \times b$ represents a cross product between two vectors a and b.

With this model, we are ready to define our state vector $\mathbf{x} \in \mathbb{R}^{12}$ and control input vector $\mathbf{u} \in \mathbb{R}^4$, defined as

- $\mathbf{x} = \begin{bmatrix} \mathbf{p} & \mathbf{v} & \boldsymbol{\alpha} & \boldsymbol{\omega} \end{bmatrix}^T$;
- $\mathbf{u} = \begin{bmatrix} \mathrm{f}_z & \boldsymbol{\tau}^T \end{bmatrix}^T$.

At this point, we are ready to linearize our dynamics in (1).

Q1: Linearize the system in (1) around the hovering equilibrium point, that is, $\theta \simeq 0, \phi \simeq 0, \mathrm{f}_z = mg$. The yaw angle you can choose either to linearize at $\psi = 0$ or as a parameter that you can change depending on the state. Choose one and motivate it.

The next step is to implement our linearized model in CasADi.

Q2: For this, use the code given for the assignments in `model.py`, and use CasADi to discretize it. Note that the discretization can be done with the `ca.integrator`, as we have used it in the assignments. Take a look at the `set_integrators` and `set_discrete_time_system` methods.

Right now you should have a model with both discrete time and continuous time dynamics in it. Therefore, we should first check if our model makes sense. Set the inertial parameters of the system to

$$m = 1.4, \tag{4}$$
$$\mathbf{M} = \mathrm{diag}(\begin{bmatrix} 0.001 & 0.001 & 0.005 \end{bmatrix}), \tag{5}$$

and start the system at the initial state of $\mathbf{x} = \begin{bmatrix} \mathbf{0}^T \end{bmatrix}$. Then perform the following simple tests:

1. Set $\mathbf{u} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$ and observe that the system stays at the initial state;
2. Set $\mathbf{u} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \end{bmatrix}^T$ and observe that the quadrotor goes up;
3. Set $\mathbf{u} = \begin{bmatrix} 0.1 & -0.001 & 0 & 0 \end{bmatrix}^T$ and observe that the quadrotor moves to the positive y axis;
4. Set $\mathbf{u} = \begin{bmatrix} 0.1 & 0 & 0.001 & 0 \end{bmatrix}^T$ and observe that the quadrotor moves to the positive x axis.

If everything is good, then we are ready for the next step.

MPC for a Quadrotor

With a working model we are ready to design the MPC controller for our system. Take as basis the MPC controller from Assignment 4, that you should understand by now.

Q3: Make sure that the running cost and terminal cost functions, given a setpoint, calculate the cost given the following error

$$\mathbf{e} = \begin{bmatrix}\mathbf{p} - \bar{\mathbf{p}}; & \mathbf{v} - \bar{\mathbf{v}}; & \boldsymbol{\alpha} - \bar{\boldsymbol{\alpha}}; & \boldsymbol{\omega} - \bar{\boldsymbol{\omega}}\end{bmatrix} \tag{6}$$

and such the cost is given by

$$J_{stage} = \sum_{t=1}^{N-1} \mathbf{e}_t^T Q \mathbf{e}_t + \mathbf{u}_t^T R \mathbf{u}_t, \tag{7}$$

$$J_{togo} = \mathbf{e}_N^T P \mathbf{e}_N. \tag{8}$$

Now, dear students, we are ready to test our MPC controller! To start with, set input and state constraints taking into account:

- The linearization was done around θ and ϕ (and possibly ψ) as 0;
- The quadrotor as a maximum thrust of 4 times its weight;
- The quadrotor motors are displaced in a circle of 17.5$[cm]$ around the center of the UAV;
- ... and for simplicity, assume that the maximum $\tau_z = 0.01$.

Before simulating our system, modify appropriately the file `simulation.py` to cope with the new state dimensions and plotting required. Start by testing the MPC with the linearized model dynamics. Choose appropriate Q, R and P weighing matrices and test the following desired setpoints in simulation:

1. Move 0.5$[m]$ in the Z direction;
2. Move 0.5$[m]$ in the Y and Z direction;
3. Move 0.5$[m]$ in the X, Y and Z direction.

Finally, simulate the system using the nonlinear dynamics, by feeding the nonlinear dynamics to a `ca.integrator` function. Look at how the nonlinear dynamics of the pendulum on cart example are simulated using this function and adjust it to the new dynamics. Which problems do you find?

Save the system responses.

Note: as we linearized the system around an equilibrium thrust $\bar{\mathbf{u}}$, you need to add this thrust on top the result from the MPC controller for the nonlinear dynamics.

Hints:

1. To debug the MPC, make sure you plot/look at the prediction horizon and control inputs. Make sure they make sense given your objectives;
2. Adjust the weights in groups: position, velocity, attitude, angular velocity, thrust, torques...

Integrator Dynamics

The last piece of our puzzle. What happens to our system if the mass we design our controller with is not exactly the mass of the vehicle? Or what if the thrust we need is not exactly the thrust we get? For these situations, integrator dynamics can provide a useful tool to fix our implementation.

Q3: Based on the lecture notes, implement the integrator dynamics that allow our system to cope with the these differences. Then adjust the simulated model (nonlinear dynamics) to simulate the difference in mass with respect to the controller and simulate the system.

Finally, to cope with center of mass displacement in our vehicle, integrator dynamics for the lateral movement are useful.

Extra: Implement the integrator dynamics for the lateral motion of the vehicle. Simulate your result by adjusting the simulated inertia matrix to an off-diagonal and comment on the results.

As always, refer to the Slack workspace `el2700workspace.slack.com` for questions.

Good Luck!