Model Predictive Control - EL2700

Assignment 4 : Model Predictive Control

2022

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

Erik Berglund and Pedro Roque

Introduction

Dear student, it is with a warm heart that we finalize present you the control method you have all been waiting for - it's time for some Model Predictive Control (MPC)!

In this assignment, we will implement a linear MPC to navigate Honey to a desired setpoint. Typically, such setpoints would be provided by some planning entity, but for now we will keep things simple and just provide said setpoints directly. We want to minimize the energy consumption, while staying within a safe set of states and control bounds - all desired properties for any space system.

In Part 1, we will compute backwards reachable sets to visualize the feasible initial states of the problem, and design an MPC to stabilize an Astrobee with force and torque inputs. In Part 2, we will analyze an MPC implementation in CasADi and get comfortable with the code-base that we will rely on. Lastly, in Part III, we will evaluate the performance of the MPC controller for navigating a set of septoints.

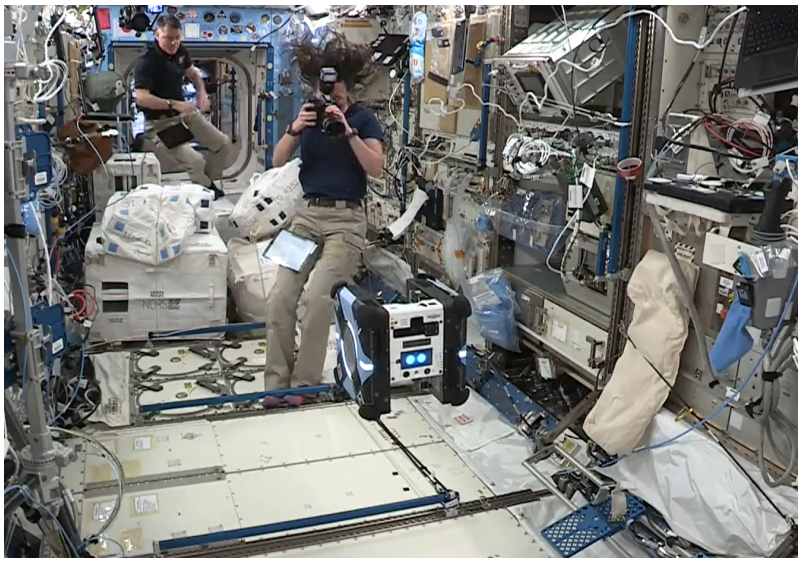

Figure 1: An Astrobee moving across the JEM module of the International Space Station (ISS) using Model Predictive Control. MPC is currently a control methodology of interest due to the possibility to handle actuation constraints, minimize the energy used for setpoint navigation, as well as ensuring operation on a safe set.

As in the LQR case, we will use the linearized version of the dynamics in (1) of Assignment 3, given by

$$\dot{\mathbf{x}} = A_c\mathbf{x} + B_c\mathbf{u}. \tag{1}$$

As before, our state $\mathbf{x}$ concatenates $\mathbf{x} = \begin{bmatrix}\mathbf{p}^T & \mathbf{v}^T & \boldsymbol{\theta}^T & \boldsymbol{\omega}^T\end{bmatrix}^T$, while the system input $\mathbf{u}$ concatenates $\mathbf{u} = \begin{bmatrix}\mathbf{f}^T & \mathbf{t}^T\end{bmatrix}^T$. Lastly, $\mathbf{p} \in \mathbb{R}^3$ is the position, $\mathbf{v} \in \mathbb{R}^3$ the linear velocity, $\boldsymbol{\theta} \in \mathbb{R}^3$ the Euler angles, $\boldsymbol{\omega} \in \mathbb{R}^3$ the angular velocity in the body-frame, $\mathbf{f}$ is the force-input to the system in body coordinates, and $\mathbf{t}$ the torque input, also in body coordinates.

Software Preparation

Before proceeding, let's make sure that you have all the software needed for this assignment. We recommend you to use Ubuntu 20.04 or above, but these instructions can also be completed on Windows and Mac as long as you have a working version of Python 3.8.10 or above, and below 3.10. Before proceeding, install the dependencies by running the script `install_deps.py` script with Python.

To complete the task, carefully look into the Python files that are part of the assignment and the classes they implement. The code entry point is `task4.py`.

Part I - The MPC Setup

MPC formulation We will now design a linear MPC for the astrobee. The discrete-time linearized dynamics of the Astrobee subjected to state and control constraints can be represented by

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t, \tag{2}$$

$$\mathbf{u}_t \in \mathcal{U}, \quad \mathbf{x}_t \in \mathcal{X}, \quad t \geq 0. \tag{3}$$

The sets $\mathcal{U}, \mathcal{X}$ are polyhedra described by the following linear inequalities

$$\mathcal{U} = \{\mathbf{u} : H_u\mathbf{u} \leq h_u\}, \quad \mathcal{X} = \{\mathbf{x} : H_x\mathbf{x} \leq h_x\} \tag{4}$$

In this design task, we will consider the following state and control constraints

$$\mathbf{x}_{min} \leq \mathbf{x}_t \leq \mathbf{x}_{max}, \quad \mathbf{u}_{min} \leq \mathbf{u}_t \leq \mathbf{u}_{max} \tag{5}$$

with $\mathbf{p}_{max} = -\mathbf{p}_{min} = \begin{bmatrix}1.2, 0.1, 0.1\end{bmatrix}^T$, $\mathbf{v}_{max} = -\mathbf{v}_{min} = \begin{bmatrix}0.5, 0.5, 0.5\end{bmatrix}^T$, $\boldsymbol{\theta}_{max} = -\boldsymbol{\theta}_{min} = \begin{bmatrix}0.2, 0.2, 0.2\end{bmatrix}^T$, $\boldsymbol{\omega}_{max} = -\boldsymbol{\omega}_{min} = \begin{bmatrix}0.1, 0.1, 0.1\end{bmatrix}^T$, $\mathbf{f}_{max} = -\mathbf{f}_{min} = \begin{bmatrix}0.85, 0.41, 0.41\end{bmatrix}^T$ and $\mathbf{t}_{max} = -\mathbf{t}_{min} = \begin{bmatrix}0.085, 0.041, 0.041\end{bmatrix}^T$.

The MPC finite-horizon optimal control (FHOC) problem is formally defined as

$$\min_{u_{t+\cdot|t}} \sum_{k=0}^{N-1} \mathbf{x}_{t+k|t}^T Q \mathbf{x}_{t+k|t} + \mathbf{u}_{t+k|t}^T R \mathbf{u}_{t+k|t} + \mathbf{x}_{t+N|t}^T P \mathbf{x}_{t+N|t} \tag{6a}$$

$$\text{subject to: } \mathbf{x}_{t+k+1|t} = A\mathbf{x}_{t+k|t} + B\mathbf{u}_{t+k|t} \tag{6b}$$

$$\mathbf{x}_{t+k|t} \in \mathcal{X} \tag{6c}$$

$$\mathbf{u}_{t+k|t} \in \mathcal{U} \tag{6d}$$

$$\mathbf{x}_{t+N|t} \in \mathcal{X}_f \tag{6e}$$

$$\mathbf{x}_{t|t} = \mathbf{x}_t \tag{6f}$$

with $P \succeq 0$, and $\mathcal{X}_f$ are the terminal cost and terminal constraint set respectively. If the terminal constraint set $\mathcal{X}_f$ is control invariant, then the MPC problem (6) is recursively feasible. In this design task, we will consider the following two control invariant sets as the terminal constraint set:

- the zero terminal constraint set $\mathcal{X}_f = \{0\}$;
- the invariant set for the closed-loop dynamics under the infinite-horizon LQR control $\mathcal{X}_f^{LQR}$.

For the MPC problem (2) to be feasible, the controller must be able to steer the system state $\mathbf{x}_t$ to the terminal set $\mathcal{X}_f$ in N steps, while satisfying the state and control constraints. To this end, we will first compute the N-step controllable set $\mathcal{K}_N(\mathcal{S})$.

We provide you the python class script `SetOperations.py` to compute $\mathcal{K}_N(\mathcal{S})$. Since the system is linearized around $\mathbf{0}$, the position-velocity subsystem becomes decoupled from the attitude-angular velocity subsystem - you can inspect matrices A and B and reason on why this is the case. Therefore, we can perform computations with them separately for time efficiency.

We will start by selecting `CASE_SELECTION = "translation"` and `SET_TYPE = "LQR"`. Your tasks are as follows:

Q1: Study the influence of the control horizon Compute the control invariant set $\mathcal{K}_N(\mathcal{X}_N)$ for varying control horizons $N = 5, 10$, and 20. Keep the state and control constraints as given in (1). Comment on the influence of N on $\mathcal{K}_N(\mathcal{X}_N)$

Q2: Influence of control constraints Compute the control invariant set $\mathcal{K}_N(\mathcal{X}_N)$ for varying the original control constraints and for $3 * \mathbf{u}_{max} = -3 * \mathbf{u}_{min}$. Keep the state constraints as given in (5) and use $N = 5$. Comment on the influence of the control constraint on $\mathcal{K}_N(\mathcal{X}_N)$.

Note: The same analysis can be taken for `CASE_SELECTION = "attitude"` and for `SET_TYPE = "zero"`, to which you should arrive at the same conclusions.

Part II - The MPC Implementation

Before simulating our Astrobee, we will inspect the MPC class that implements the Model Predictive Controller defined in (6).

Q3: Examine the MPC controller implementation under the file `mpc.py`. Explain using your own words (you can and should take code snippets to help with the explanation):

1. How the state constraints are set;
2. How the object function is formulated;
3. How the terminal constraint is set;
4. What the variable `param_s` holds;
5. Why we only select the first element of the control prediction horizon in the `mpc_controller` method.

Part III - MPC for Stabilization

You will now simulate an Astrobee with MPC by running the the script `task4.py` and setting `CASE_SELECTION = "simulate"` and `SET_TYPE = "LQR"`.. The initial state for the simulation is set to $\mathbf{x}_0 = \begin{bmatrix}0.2 & \mathbf{0}^5 & 0.08 & \mathbf{0}^5\end{bmatrix}^T$ and we will use a receding horizon of length $N = 10$.

Q4: Simulate the system with the current setup, and compare its performance against $u_{max} = -u_{min} = 3 * u_{lim}$ (that is, after multiplicating the control bounds by 3). Comment on the energy used, as well as the position and attitude integral errors (printed on your terminal at the end of the simulation).

Q5: Put the control bounds back to the original values. Set as terminal set the sets: (i) $\mathcal{X}_f = \{\mathbf{0}\}$, and (ii) $\mathcal{X}_f^{LQR}$. What do you notice? Compare your result with what you observe for Q1, when selecting the `SET_TYPE = "zero"`.

Q6: Change the controller horizon to $N = 50$ keeping $\mathcal{X}_f = \{\mathbf{0}\}$. Your problem should now be feasible. Comment on the computational time by inspecting the time to solve the MPC problem.

We will now focus on the case with $\mathcal{X}_f^{LQR}$ and the initial control limits given by $\mathbf{u}_{max} = -\mathbf{u}_{min} = \begin{bmatrix}0.85 & 0.41 & 0.41 & 0.085 & 0.041 & 0.041\end{bmatrix}$.

Q7: We will now investigate what effect the tuning parameters have on the performance of the controller. Starting with `Q = np.eye(12)` and `R = np.eye(6)`, make the following changes to the matrices, one at a time:

- Multiply R by 10
- Add 100 to the velocity components of the diagonal of Q, `Q[3:6]` and `Q[9:]`
- Revert the velocity components to their initial value of 1 and add 100 to the position and attitude components of the diagonal of Q, `Q[0:3]` and `Q[6:9]`
- Increase all elements of Q by 100.

For each case, briefly describe and explain what happens. Relate this behavior with what you observed for the LQR controller.

To complete this design project, you should upload a zip file containing the provided code, properly completed. The task grading is based on a successful run of `task4.py`, which should provide all the results. Post all your questions on the Slack workspace `el2700mpc-ht22.slack.com`.

Good Luck!