Model Predictive Control
Prof. M. Zeilinger

Spring 2017

Programming Exercise
Quadrotor control

In this project you will implement an MPC controller for a quadrotor, which is an aircraft driven by four independently controlled rotors.

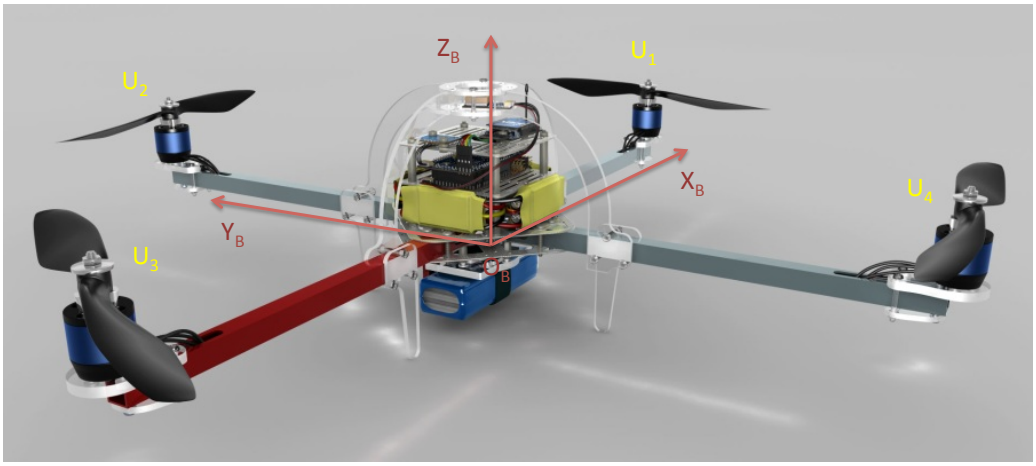

Figure 1: Quadrotor prototype, the body frame is shown in red and the controlled inputs in yellow.

Provided MATLAB files

Please go to Moodle, download and unpack the files `quadrotor.zip`. Once you will open the quadrotor folder you will find the template for the exercise `main.m`. Download the latest version of Yalmip from `https://yalmip.github.io/download/`. Add the unpacked folder `YALMIP-master` to the Matlab path.

To install FORCES Pro proceed as follows:

1. Register as a new user on `embotech.com/register`. Complete registration by verifying your email and clicking on the corresponding link.
2. Navigate to the *Downloads* page and download FORCES PRO v1.6.
3. Save the zip file in a convenient location.
4. Go to that directory in Matlab and unpack the archive by double clicking or by typing in the command window: `unzip FORCES_PRO_v1_6.zip`
5. Add FORCES PRO to the Matlab path, e.g. by right click to `FORCES` → Add to Path → Selected Folders and Subfolders.

And to install MPT follow the following instructions

1. Browse to `http://people.ee.ethz.ch/~mpt/3/` and click on "Installation & updating instructions".
2. *Downloads* the file `install_mpt3.m`.
3. Run on MATLAB the file `install_mpt3.m`.

Now run the function `main.m` from the folder `quadrotor`, if it finished successfully, you are ready to start. The set of Matlab files provided is

- `main.m`: Template for the programming exercise,
- `getOuterController.m`: Function that computes the outer controller,
- `plotQuad.m`: Function called by the simulink models to plot the quadrotor,
- `plotQuadSimResult.m`: Script called at the end of the simulink simulations for plotting the results,
- `simQuad.m`: Function that simulates the linearized model,
- `simulation1.mdl`: Simulink model that simulates the quadrotor without disturbances,
- `simulation2.mdl`: Simulink model that simulates the quadrotor with disturbances,
- `quadData.mat`: Data files containing the linearized model.

For the tasks given in the following, you only need to modify the files `main.m` and `simulation2.mdl`.

What you have to hand in

Up to three students are allowed to work together on the programming exercise. They will all receive the same grade. As a group, you must read and understand the ETH plagiarism policy here: `http://www.plagiarism.ethz.ch/` - each submitted work will be tested for plagiarism. You must download and fill out the Declaration of Orignality, available at the same link.

Hand in a single zip-file, where the filename contains the names of all team-members according to this template (note that there are no spaces in the filename):

`MPC17PE_Firstname1Surname1_Firstname2Surname2_Firstname3Surname3.zip`.

The zip-file contains only four files:

- The file `main.m` with the working MATLAB code,
- The file `simulation2.mdl`,
- A small pdf report with requested plots and answers to questions posed.
- A scan of the Declaration of Originality, signed by all team-members.

The zip-file should be uploaded in Moodle in the Programming Exercise assignment area by just one of the members of the group. The deadline for submission is **May 26** (midnight). Late submissions will not be considered.

Nonlinear model and linearization

In order to derive a nonlinear model consider two reference frames. The first one is the body frame (subscript B) with the origin O_B attached to the center of mass of the quadrotor as shown in Figure 1. The second one is the world frame which remains fixed. We are going to derive a 12-state description of the robot with the state vector

$$\mathbf{x} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \alpha & \beta & \gamma & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix},$$

where $(x, y, z) = O_B$ is the position of the center of mass and (α, β, γ) are the roll, pitch and yaw angles. These angles represent, in order, the rotation around the x_B axis, the rotation around the y_B axis, and the rotation around the z_B axis.

The input of the system

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$$

is the thrusts of the four rotors. These thrusts are proportional to the square of the rotor angular velocity and hence are nonnegative. Consequently, each rotor produces a force F_i and moment M_i according to

$$F_i = k_F u_i, \quad M_i = k_M u_i.$$

The net body force F and body moments $(M_\alpha, M_\beta, M_\gamma)$ can be expressed in terms of $\mathbf{u}$ as

$$\begin{bmatrix} F \\ M_\alpha \\ M_\beta \\ M_\gamma \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \mathbf{u}. \tag{1}$$

where L is the distance from the center of mass to the center of the rotor.

The acceleration of the center of mass is then given by

$$\ddot{O}_B = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + F\mathbf{z}_{\mathrm{B}}^{\mathrm{W}}. \tag{2}$$

where g is the acceleration due to gravity and $\mathbf{z}_{\mathrm{B}}^{\mathrm{W}}$ is the unit vector in the z direction of the body coordinate frame expressed in the world coordinates.

The angular dynamics are given by

$$\dot{\omega} = \mathcal{I}^{-1} \left(-\omega \times \mathcal{I}\omega + \begin{bmatrix} M_\alpha \\ M_\beta \\ M_\gamma \end{bmatrix} \right), \tag{3}$$

where $\omega = \dot{\alpha}\mathbf{x}_{\mathrm{B}}^{\mathrm{W}} + \dot{\beta}\mathbf{y}_{\mathrm{B}}^{\mathrm{W}} + \dot{\gamma}\mathbf{z}_{\mathrm{B}}^{\mathrm{W}}$ is the angular velocity of the body coordinate frame with respect to the world coordinate frame and $\mathcal{I}$ is the inertia matrix of the quadrotor.

Now take a look at Equation 1, which is the key to understanding of how a quadrotor works. From the first line we can see that the net body force F (in the direction $\mathbf{z}_{\mathrm{B}}^{\mathrm{W}}$) is simply the sum of the four rotor thrusts (times a constant k_F). From the second line we see that the moment around the $\mathbf{x}_{\mathrm{B}}^{\mathrm{W}}$ axis (which then gives rise to a change in the roll angle α) is given by the imbalance in the thrusts of the rotors 2 and 4. Similarly the pitch angle β is controlled by the imbalance in the thrusts of the rotors 1 and 3. Finally, from the fourth line we can see that the yaw angle γ is controlled by the imbalance in the net thrusts of the opposite rotor pairs (1,3) and (2,4). This is because the two pairs rotate in the opposite direction (and therefore generate no net moment around $\mathbf{z}_{\mathrm{B}}^{\mathrm{W}}$ when the net thrusts of the two pairs are equal; otherwise there is a moment imbalance and the quadrotor rotates).

From Equation 2 we see that in order to fly the quadrotor around we need to control the body direction $\mathbf{z}_{\mathrm{B}}^{\mathrm{W}}$ (which is a function of α and β) and simultaneously apply the net body force F in that direction, all of this while counteracting gravity.

Finally, Equation 3 says that the body angles (α, β, γ) are controlled by the moments $(M_\alpha, M_\beta, M_\gamma)$, which are directly related to the rotor thrusts through Equation 1.

Now have a look at the linearized model that we have prepared for you. The state-space description is given by the matrices A_c and B_c which has been loaded by running the function `main.m`. The model has been linearized in a hovering equilibrium $(\mathbf{x}_s, \mathbf{u}_s)$. This is an equilibrium given by zero angles (α, β, γ), some (x, y, z) position, and a steady state control input $\mathbf{u}_s$ that exactly counteracts the gravity.

Consider the structure of the matrices A_c and B_c. Does it make intuitive sense in relation to the physical model?

Deliverables

1. Interpretation of the structure of matrices A_c and B_c. Explain in particular the nonzero numbers in rows 4 and 5 of A_c and the nonzero rows of B_c in connection with the nonlinear dynamics described above. *2.5 %*

Control Structure

In this exercise we will use a cascaded controller, where the inner loop controller controls

$$\mathbf{x}^1 = \begin{bmatrix} \dot{z} & \alpha & \beta & \gamma & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix},$$

and the outer loop controller is in charge of

$$\mathbf{x}^2 = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \end{bmatrix}.$$

The inner-loop reference signal is

$$\mathbf{r}^1 = (r_{\dot{z}}, r_\alpha, r_\beta, r_\gamma),$$

where the first three components $(r_{\dot{z}}, r_\alpha, r_\beta)$ are governed by the outer loop controller, whereas r_γ (the yaw angle) comes from outside. The outer-loop reference signal $\mathbf{r}^2$ is the space position (r_x, r_y, r_z). The control structure is shown in Figure 2.

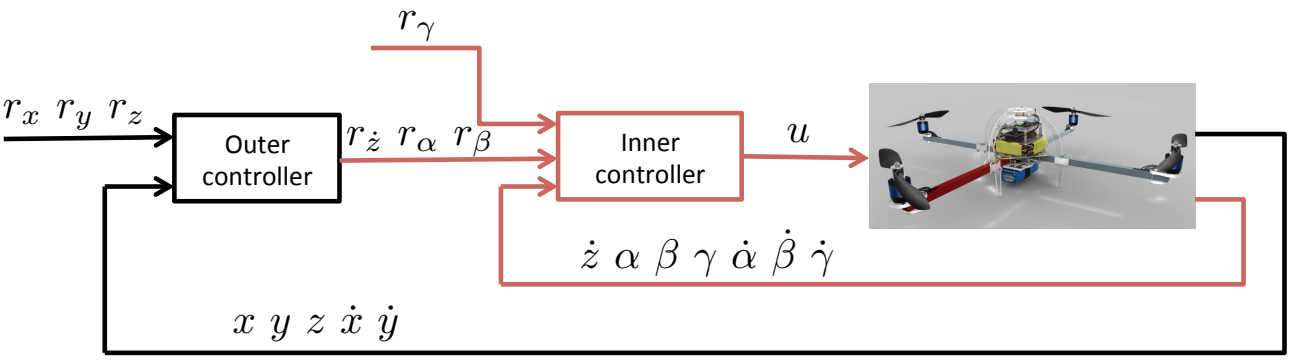

Figure 2: Casceded control loop used to control the system.

The outer-loop controller and the interconnection of the two controllers have already been designed for you. Your task is to design the inner-loop controller shown in red.

You are given an inner-loop model discretized with sampling period $T_s = 0.1\,\text{s}$. All you need to focus on from now on is the inner loop model with the measured state $\mathbf{x}^1$, the input $\mathbf{u}$ and the reference signal $\mathbf{r}^1$.

The discrete time model matrices A, B and the sampling period T_s are contained in the structure `sys` that is loaded when running `main.m`.

Constraint specification

The inner-loop quadrotor model is subject to constraints on the maximum angle, maximum angle velocity as well as maximum velocity in the z direction – these constraints mainly ensure validity of the linearized model and are specified as follows

- $|\dot{z}| \leq 1$ m/s,
- $|\alpha| \leq 10^\circ, \quad |\beta| \leq 10^\circ,$
- $|\dot{\alpha}| \leq 15^\circ/\text{s}, \quad |\dot{\beta}| \leq 15^\circ/\text{s},$
- $|\dot{\gamma}| \leq 60^\circ/\text{s}.$

The nonlinear model is also subject to the input constraints

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leq \mathbf{u} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Exploit Yalmip's modeling capabilities when specifying the constraints. For instance, the constraint on α and β along the prediction horizon N (excluding the initial state) can be written as

```
x1 = sdpvar(7, N);
constraints = [];
for i = 2:N
  constraints = constraints + set(-angleMax <= x1(2:3,i) <= angleMax);
end
```

where `angleMax` is a vector containing the constraints on α and β.

Warning: You will need to convert the degrees to radians and also bear in mind that the input constraint is on the *nonlinear* model whereas you are specifying constraints for the model linearized in the hovering equilibrium $(\mathbf{x}_s, \mathbf{u}_s)$ for which $\mathbf{u}_s \neq 0$.

Note that there are also constraints on the outer loop variables, that is, on the spatial coordinates and velocities of the quadrotor. You don't need to take them into account for the design of the inner loop controller, but they will constrain the movement of the quad during simulations on the full nonlinear model. These constraints are

- $|x| \leq 1$ m,
- $|y| \leq 1$ m,
- $-1\,\text{m} \leq z \leq 4$ m.

First MPC controller

The next task is to design and test an inner loop MPC controller. Design an MPC controller that will regulate the state from the nonzero initial condition

$$\mathbf{x}(0) = \begin{bmatrix} -1 & 0.1745 & -0.1745 & 0.8727 & 0 & 0 & 0 \end{bmatrix},$$

to the origin. The response you are aiming for has an error of less than 1 degree and 0.1m/s in about 2 seconds for roll and pitch angles α and β and vertical velocity $\dot{z}$ – these are signals provided by the outer controller and hence have to be controlled quickly. The yaw angle γ can be slower. Therefore the standard initial guess $Q = I$ and $R = I$ may not work for you in this case. You will also need to choose the prediction horizon length and design a polytopic terminal set $(A_{\mathcal{X}_f}, b_{\mathcal{X}_f})$ and the terminal cost.
Hint: To compute the terminal set you can use the MATLAB toolbox MPT. In particular have a look at the functions `LTISystem.m` and `InvariantSet`.

To simulate the linearized system you can use the function `simQuad.m`. It takes the Yalmip optimizer instance as its second argument. Take a look at the documentation of the function `simQuad.m`. The syntax you end up using should look like this

```
options = sdpsettings;
innerController = optimizer(constraint, objective, options, x(:,1), u(:,1));
simQuad( sys, innerController, bForces, x0, T);
```

where `x(:,1)` and `u(:,1)` are sdpvar instances representing the initial state x_0 and the first control input u_0, T is the simulation length in seconds, and `bForces` has to set to 1 if the solver is FORCES Pro, 0 otherwise. The remaining arguments of `simQuad.m` will be needed later on for reference tracking and disturbance rejection.

Deliverables

2. Choice of tuning parameters $(Q, R, P, A_{\mathcal{X}_f}, b_{\mathcal{X}_f})$ and motivation for them *5 %*
3. Plots of the response starting from the given initial condition $\mathbf{x}(0)$ *10 %*

Reference tracking

Next, you should add reference tracking. The reference signals to be tracked are $(\dot{z}, \alpha, \beta, \gamma)$ commands from the outer loop controller. For design purposes you can regard them as constant signals.

First you should define the set of equilibrium states and inputs $(\mathbf{x}_r, \mathbf{u}_r)$ corresponding to the reference signal

$$\mathbf{r}^1 = \begin{bmatrix} \dot{z} & \alpha & \beta & \gamma \end{bmatrix}^T,$$

given the output matrix $C = [I_4 \;\; 0_{4\times 3}]$.

After computing the steady state solution, implement the reference tracking MPC formulation from the lecture. When you are done, you can check that you can also track (perhaps with some error) slowly varying reference signals. Consider $\mathbf{x}(0) = 0$ as your initial condition and you do not need to implement the terminal set.

The parameter vector for your controller optimizer instance should be $[(\mathbf{x}^1)^T \;\; (\mathbf{r}^1)^T]^T$; then you can use the same function for simulations as before.

Deliverables

4. Define the steady state $(\mathbf{x}_r, \mathbf{u}_r)$ as a function of an arbitrary reference $\mathbf{r}^1$ *5 %*
5. Plots of the response for the constant reference signal $\mathbf{r}^1 = \begin{bmatrix} 1.0000 \\ 0.1745 \\ -0.1745 \\ 1.7453 \end{bmatrix}$ *10 %*
6. Plots of the response for the slowly varying reference signal $\mathbf{r}^1(k) = \begin{bmatrix} 1 \\ 0.1745\sin(T_s k) \\ -0.1745\sin(T_s k) \\ \pi/2 \end{bmatrix}$ *2.5 %*

First simulation of the nonlinear model

The next step is to test your controller on the nonlinear model. Open the file `simulation1.mdl`. Make sure that your optimizer instance is named `innerController` and the order of variables is $[(\mathbf{x}^1)^T \;\; (\mathbf{r}^1)^T]^T$, then you should be able to run the simulation with the simulink prespecified reference signal without any changes, just by pressing the play button.

Deliverables

7. Plots of a reference tracking response of the nonlinear model. *5 %*

Offset free MPC

The nonlinear model is subject to various disturbances, e.g. wind, mismatch in the model, etc. All these disturbances can be packed into a single disturbance (random) variable $\mathbf{d} \in \mathbb{R}^7$ with mean $\bar{\mathbf{d}}$ and zero-mean component $\tilde{\mathbf{d}}$. The discrete-dynamics can then be written as

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + B\mathbf{u}(k) + \bar{\mathbf{d}} + \tilde{\mathbf{d}}(k).$$

Your goal is to estimate the time-invariant mean $\bar{\mathbf{d}}$ while rejecting the unpredictable $\tilde{\mathbf{d}}(k)$. Therefore, design a disturbance estimator to estimate $\bar{\mathbf{d}}$. You can model the disturbance as the dynamical system

$$\bar{\mathbf{d}}(k+1) = \bar{\mathbf{d}}(k),$$

and design a full-state estimator for the augmented system

$$\begin{bmatrix} \mathbf{x}^1(k+1) \\ \bar{\mathbf{d}}(k+1) \end{bmatrix} = \underbrace{\begin{bmatrix} A_1 & I \\ 0 & I \end{bmatrix}}_{A_{aug}} \begin{bmatrix} \mathbf{x}^1(k) \\ \bar{\mathbf{d}}(k) \end{bmatrix} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{B_{aug}} \mathbf{u}(k). \quad \mathbf{y}(k) = \underbrace{\begin{bmatrix} I_{7\times 7} & I_{7\times 7} \end{bmatrix}}_{C_{aug}} \begin{bmatrix} \mathbf{x}^1(k) \\ \bar{\mathbf{d}}(k) \end{bmatrix} + \tilde{d}(k).$$

The estimator should be of the form

$$\begin{bmatrix} \hat{\mathbf{x}}^1(k+1) \\ \hat{\bar{d}}(k+1) \end{bmatrix} = A_f \begin{bmatrix} \hat{\mathbf{x}}^1(k) \\ \hat{\bar{d}}(k) \end{bmatrix} + B_f \begin{bmatrix} \mathbf{u}(k) \\ \mathbf{x}^1(k) \end{bmatrix},$$

where $\hat{\mathbf{x}}^1$ is the estimate of $\mathbf{x}^1$, which is not needed since it is measured, and $\hat{\bar{d}}$ is the estimate of $\bar{\mathbf{d}}$. The matrices A_f and B_f are of the form

$$A_f = A_{aug} - LC_{aug}, \quad B_f = \begin{bmatrix} B_{aug} & L \end{bmatrix},$$

where $L \in \mathbb{R}^{14\times 7}$ is the observer matrix that you need to design.

Again, you can use `simQuad.m` to perform your simulations with one of the arguments being a structure `filter` that has to contain the matrices (A_f, B_f) (as fields `filter.Af` and `filter.Bf`) of the state-space representation of the disturbance estimator. Consider your system starting at the initial condition $\mathbf{x}(0) = 0$.

Hints: If you have trouble (e.g., running into infeasibility) when tuning the estimator, you may consider dropping the constraints in your MPC problem during the tuning stage or otherwise separate the controller and estimator design.

Deliverables

8. Provide the matrix L and justify your choice. *5 %*

9. Plots of the response for the constant reference signal $\mathbf{r}^1 = \begin{bmatrix} 0.8 \\ 0.12 \\ -0.12 \\ \pi/2 \end{bmatrix}$ *10 %*

10. Plots of the response for the slowly varying reference signal $\mathbf{r}^1(k) = \begin{bmatrix} 0.8 \\ 0.12\sin(T_s k) \\ -0.12\sin(T_s k) \\ \pi/2 \end{bmatrix}$ *2.5 %*

Simulations on the nonlinear model

Try out your offset-free MPC controller on the nonlinear model. Open the Simulink model `Simulation2.mdl` and type in the Inner MPC block the name of your controller optimizer instance. The parameter vector for your controller optimizer instance should be $[(\mathbf{x}^1)^T\ (\mathbf{r}^1)^T\ \hat{\mathbf{d}}^T]^T$, where $\hat{\mathbf{d}}$ is the estimate of $\bar{\mathbf{d}}$.

Simulate first the model without disturbances (you can turn them on and off via a checkbox on the quadrotor block). If the response is satisfactory, simulate the system with disturbances.

Deliverables

All of the following need to be performed in the presence of disturbances.

11. Plots of the reference tracking of a step signal provided in the simulink model. *5 %*

12. Plots of the reference tracking of the hexagon signal provided in the simulink model. *2.5 %*

13. Plots of the reference tracking of an "eight" (i.e., a lemniscate or ∞ shape) – you need to create the signal on your own. *5 %*

Slew rate constraints

Add slew rate constraints on the control inputs, that is, constraints of the form $|u_{k+1} - u_k| \leq \Delta$. The function `simQuad.m` (with 1 as its last argument) can handle a controller optimizer instance which accepts the last control move applied to the system as its *third* input and the estimated disturbance as its *fourth* input (read the description of the function carefully). How small a Δ can you find without significant performance deterioration / feasibility problems starting form the initial condition $\mathbf{x}(0) = 0$ and tracking the constant signal $\mathbf{r}^1 = [0.8\ 0.12\ -0.12\ \pi/2]^T$?

Deliverables

14. Provide plots of all state variables and control inputs for the step response for the linearized model and the value of Δ. 5 %

Soft Constraints

To ensure feasibility of the MPC problem with slew rate constraints on the control input, next task is to implement soft constraints. Implement soft constraints only on the slew rate constraints of the inputs and design a suitable cost function to add to the MPC problem.

Deliverables

15. Provide the cost function for the MPC problem with soft constraints 5 %
16. Plots of the response for the constant reference signal $r^1 = \begin{bmatrix} 0.8 \\ 0.12 \\ -0.12 \\ \pi/2 \end{bmatrix}$ starting from the initial condition $\mathbf{x}(0)$ *10 %*

FORCES Pro

To deploy your controller to a real quadrotor you would need to achieve fast sampling times and generally to write the MPC problem using a low-level language like C or use a code generator. FORCES Pro is code generator that is compatible with any embedded platform having a C compiler. Now run again points 5 and 9 with FORCES Pro and compare the average solver running time (returned by `simQuad.m`).

Deliverables

17. Report the average solver running time for points 5 and 9 using FORCES Pro and compare it with the one you used before (if you have used Forces Pro before then compare with `quadprog`) *10 %*