1 Model predictive control unconstrained

1.1 UAV dynamics

For understanding the UAV's controller, it is important to understand the physical properties of UAV. It has 4 rotors,

1.2 MPC Introduction

MPC is an advanced regulator, using prediction of future states of the system for determining system input actions. Because of it's computational demanding, it is used mainly in processes with long time constants, such as control of chemical processes. Using MPC for controlling UAV is a big challenge, because it is hard to implement on embedded hardware. The MPC needs to know the system's state space model, initial condition and, unlike other controllers, a sequence of desired future states. Another great advantage of MPC is applying large variety of constraints. On the other hand, MPC is sensitive to the model inaccuracy and to errors in input. Output of the MPC is not only desired input actions in next time step, but also predicted action inputs in the whole prediction horizon for T following time steps.

1.3 Formulation of quadratic programming

MPC formulates optimization problem, that then has to be solved. The problem takes form of Linear Programming(LP), or in this case Quadratic Programming(QP).

$$\mathrm{V}\left(\underline{\mathbf{x}}, \underline{\mathbf{u}}\right) = \frac{1}{2} \sum_{i=0}^{M-2} \left(\mathbf{e}_{[i]}^T \mathbf{Q} \mathbf{e}_{[i]} + \mathbf{u}_{[i]}^T \mathbf{P} \mathbf{u}_{[i]} \right) + \frac{1}{2} \mathbf{e}_{[M-1]} \mathbf{S} \mathbf{e}_{[M-1]} \tag{1}$$

There are several ways to solve this problem, which will be discussed later.

1.4 Coordinate system

This whole problem will be solved in 2D only. The height of the UAV is controlled separately. There are several reasons to do that. The first one is, that most applications use constant height, for example building interiors. The desired trajectory is usually also given in 2D. The second reason is, as mentioned above, MPC is very demanding on computing time. To save computational capacity, standard PID controller can be used instead. The height PID controller has already been implemented and it is will not be part of this thesis. In these 2 dimensions, there are 2 axis as shown in figure.... There are two coordinate systems, which will be used. The first one is a standard world coordinate system W. The second one is a coordinate system U, which is a system with origin in the UAV. Coordinate system U is created only by translation of the system W for the vector $\vec{r} = (\Delta x, \Delta y)$. There is no rotation between the two coordinate

systems, so the axis of the both coordinate systems are parallel. Because the UAV can move easily along each axis, there is no need to introduce the UAV's rotation. If the UAV would rotate over time, the model would no more be linear and control of this system would be much more complex. These coordinate systems transformations work according the equation 2.

$$\begin{aligned} x^{(W)} &= x^{(U)} + \Delta x \\ y^{(W)} &= y^{(U)} + \Delta y \\ \dot{x}^{(W)} &= \dot{x}^{(U)} + \Delta \dot{x} \\ \dot{y}^{(W)} &= \dot{y}^{(U)} + \Delta \dot{y} \\ \ddot{x}^{(W)} &= \ddot{x}^{(U)} + \Delta \ddot{x} \\ \ddot{y}^{(W)} &= \ddot{y}^{(U)} + \Delta \ddot{y} \end{aligned} \tag{2}$$

In the following text, there will be used the world coordination system W unless stated otherwise.

1.5 One axis model

The UAV model has been already analyzed –cite Tomas–. Thanks to symmetrical body of the UAV, both axis have the same model. For one axis, the states take form of $\vec{x}_x = (x_x; \dot{x}_x; \ddot{x}_x)^T$ and $\vec{x}_y = (x_y; \dot{x}_y; \ddot{x}_y)^T$, where x_x is the aileron and x_y is the elevator position. The aileron and elevator models are mathematically identical. The system in discrete state space model takes form of system matrices $\mathbf{A}_s, \mathbf{B}_s$ according to equation 5

$$\vec{x}_{x,y,[t+1]} = \mathbf{A}_{x,y}\vec{x}_{x,y,[t]} + \mathbf{B}_{x,y}u_{x,y,[t]} \tag{3}$$

where $u_{x,y,[t]}$ is an input at time t, sampling with the frequency of $1/\Delta t = 70Hz$.

$$\mathbf{A}_{x,y} = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & \Delta t \\ 0 & 0 & p_1 \end{bmatrix}, \mathbf{B}_{x,y} = \begin{bmatrix} 0 \\ 0 \\ p_2 \end{bmatrix}, \tag{4}$$

where $p_1 = 0.9799$ and $p_2 = 5.0719 \cdot 10^{-5}$. The unconstrained MPC can be computed separately for elevator and aileron axis. Simple constraints, such as input saturation can be applied.

1.6 Linked/Extended model

For more complex constraints, such as position constraints, one axis position is a function of the other axis position. For these kinds of constraints more complicated system is needed. The state space system must be preserved, connecting both identical systems for each axes into one system extending equation 5 into

$$\mathbf{x}_{[t+1]} = \mathbf{A}\mathbf{x}_{[t]} + \mathbf{B}\mathbf{u}_{[t]} \tag{5}$$

where $\mathbf{u}_{[t]} = (u_{x,[t]}, u_{y,[t]})^T$ is input vector containing elevator and aileron system inputs at the time t. Extended state vector $\mathbf{x}_{[t]} = (x_{[t]}, \dot{x}_{[t]}, \ddot{x}_{[t]}, y_{[t]}, \dot{y}_{[t]}, \ddot{y}_{[t]})^T$ contains positions x, y and theirs derivatives at the time t.

By connecting these 2 systems, we get the following state space matrices of the whole system

$$\begin{bmatrix} \mathbf{A}_s & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_s \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_s & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_s \end{bmatrix}. \tag{6}$$

1.7 Kalman

As mentioned in introduction, MPC is very sensitive to noise and errors. Wrong initial condition can result in a bad prediction because of the double integration of acceleration into position. To work properly, the MPC has to have access to very accurate initial condition. These measurements are position, speed and acceleration in both axis. An Kalman estimator has been already implemented -cite Tomas- to estimate all states of the UAV. Above all of that, it estimates disturbances of the acceleration.

1.8 system constraints

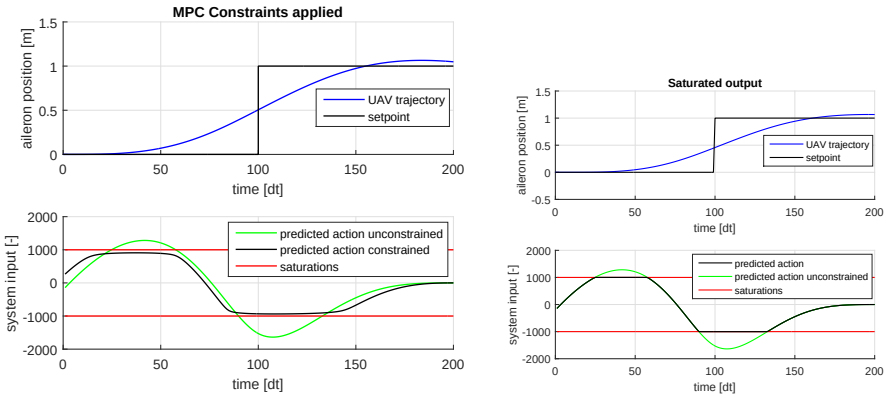
One of the greatest advantages of MPC is being able to apply large variety of constraints. As mentioned in 1, the MPC uses linearly constraint quadratic programming for finding the input action prediction. The constraints must defined with the matrices $\mathbf{A}_c$ and $\mathbf{B}_c$ according to

$$\mathbf{A}_c u < \mathbf{B}_c \tag{7}$$

$\mathbf{A}_c$ is a matrix of width 2T and height of the number of constraints applied. $\mathbf{B}_c$ is a column vector of the same height.

1.8.1 input constraints

In control, one of the biggest problems one must overcome is system saturation. This means, that the real system is linear only on a certain range of inputs and can't achieve desired output [**?**]. For example motor can spin in certain maximum speed despite input voltage. Also, if the controller sends too high input actions, the system can be destroyed. This can happen for example in PD control, when the difference between real and desired output changes rapidly in time, for example because of sensory noise. The standard solution for this kind of problem is simply saturate the output of the controller. This is a very simple solution, but it is not the best one. Because the controller doesn't know about this saturation, the system will not behave correctly. The great advantage of input constraints in MPC is, that the controller will consider these aspects of real system and find an input prediction, that will not violate the constraints and at the same time will achieve the desired output.



(a) Tricopter aircraft.

(b) The px4flow sensor.

Figure 1: Tricopter platform with px4flow optical flow sensor.

The condition

1.8.2

1.9 System prediction

The MPC algorithm is based on predicting future states based on initial condition and system input. Such a general equation must be found. Let's repeat the equation 5.

$$\mathbf{x}_{[t+1]} = \mathbf{A}\mathbf{x}_{[t]} + \mathbf{B}\mathbf{u}_{[t]}, \tag{8}$$

With a simple substitution we can get prediction of the states at the time $t = 2$.

$$\begin{aligned} \mathbf{x}_{[1]} &= \mathbf{A}\mathbf{x}_{[0]} + \mathbf{B}\mathbf{u}_{[0]}, \\ \mathbf{x}_{[2]} &= \mathbf{A}\mathbf{x}_{[1]} + \mathbf{B}\mathbf{u}_{[1]} \\ &= \mathbf{A} \cdot (\mathbf{A}\mathbf{x}_{[0]} + \mathbf{B}\mathbf{u}_{[0]}) + \mathbf{B}\mathbf{u}_{[1]} \\ &= \mathbf{A}^2\mathbf{x}_{[0]} + \mathbf{A}\mathbf{B}\mathbf{u}_{[0]} + \mathbf{B}\mathbf{u}_{[1]} \end{aligned} \tag{9}$$

The equation 9 can be rewritten in a more general way:

$$\mathbf{x}_{[t]} = \mathbf{A}^t\mathbf{x}_{[0]} + \sum_{i=1}^{t-1} \mathbf{A}^i\mathbf{B}\mathbf{u}_{[i-1]} + \mathbf{B}\mathbf{u}_{[t-1]} \tag{10}$$

Let's combine the sequence of predicted states into one vector $\underline{\mathbf{x}} = (\mathbf{x}_{[1]}^T, \mathbf{x}_{[2]}^T, ..., \mathbf{x}_{[T]}^T)^T$ and the sequence of inputs into $\underline{\mathbf{u}} = (\mathbf{u}_{x,[0]}, \mathbf{u}_{y,[0]}, \mathbf{u}_{x,[1]}, \mathbf{u}_{y,[1]}, ..., \mathbf{u}_{x,[T-1]}, \mathbf{u}_{y,[T-1]})^T$

With this notation, equation 10 can be represented as a simple matrix multiplication.

$$\underbrace{\begin{bmatrix}\mathbf{x}_{[1]}\\ \mathbf{x}_{[2]}\\ \vdots\\ \mathbf{x}_{[T]}\end{bmatrix}}_{\underline{\mathbf{x}}} = \underbrace{\begin{bmatrix}\mathbf{A}\\ \mathbf{A}^2\\ \vdots\\ \mathbf{A}^{(T-1)}\end{bmatrix}}_{\hat{\mathbf{A}}} \mathbf{x}_{[0]} + \underbrace{\begin{bmatrix}\mathbf{B} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{AB} & \mathbf{B} & \mathbf{0} & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{A}^{(T-1)}\mathbf{B} & \mathbf{A}^{(T-2)}\mathbf{B} & \dots & \mathbf{B}\end{bmatrix}}_{\hat{\mathbf{B}}} \cdot \underbrace{\begin{bmatrix}\mathbf{u}_{[0]}\\ \mathbf{u}_{[1]}\\ \vdots\\ \mathbf{u}_{[T-1]}\end{bmatrix}}_{\underline{\mathbf{u}}} \tag{11}$$

Using the new notation, this can be be rewritten in a simple form

$$\underline{\mathbf{x}} = \hat{\mathbf{A}}\mathbf{x}_{[0]} + \hat{\mathbf{B}}\underline{\mathbf{u}}. \tag{12}$$

2 Problem formulation

As mentioned in a section ???, MPC uses a quadratic optimization problem. Let's first solve the problem, where obstacles are not involved.

2.1 Trajectory

For every task, there is given a desired trajectory $\underline{\mathbf{x}}_d = (\mathbf{x}_{d,[1]}^T, \mathbf{x}_{d,[2]}^T, ..., \mathbf{x}_{d,[T]}^T)^T$, where the desired state at the time t is $\mathbf{x}_{d,[t]} = (x_{d,[t]}, \dot{x}_{d,[t]}, \ddot{x}_{d,[t]}, y_{d,[t]}, \dot{y}_{d,[t]}, \ddot{y}_{d,[t]})^T$. These desired states contain, besides aileron and elevator position, also velocity and acceleration. This gives the MPC a chance to enforce other properties outside position. However, The UAV desired velocity is already given by the desired positions at certain time as the distance $d = \sqrt{(x_{d,[t]} - x_{d,[t+1]})^2 + (y_{d,[t]} - y_{d,[t+1]})^2}$. The same can be applied for acceleration. Therefore, the velocity and acceleration is demanded in the sequence of desired positions, rather than the the states. The desired velocity and acceleration is than ignored, which will be discussed in a section ??? and it can hold any value, for example 0. The $\mathbf{x}_{d,[t]}$ than takes form of $\mathbf{x}_{d,[t]} = (x_{d,[t]}, 0, 0, y_{d,[t]}, 0, 0)^T$. When creating the desired trajectory, one should always keep in mind, that the desired positions hold also information about velocity and acceleration.

2.1.1 Objective function

The goal of unconstrained MPC is to follow the given trajectory as well as possible. This means, that we want to minimize the error between all the predicted positions and the desired positions, gaining error:

$$\begin{aligned} e_{x,t} &= x_{[t]} - x_{d,[t]}\\ e_{y,t} &= y_{[t]} - y_{d,[t]} \end{aligned} \tag{13}$$

Using this kind of error has many downsides. The first one is, that if we want to minimize it, we would have to take the absolute value. However this function

would not be differentiable. The second one is, that we don't get very good results if penalizing the error linearly. From the experience, it has come beneficial in many ways to use the the the square of the error. This preserves the condition of not prioritizing one direction error. The function is also easily differentiable. The next great advantage is penalizing big distances disproportionately more and ignoring very small errors. If a simple sum of $e_{x,t}^2$ and $e_{y,t}^2$ was applied, the system would behave very wildly, generating very high input actions to correct the error. However, this is not in the capabilities of the real system and could result in an unstable control. Therefore input actions must be penalized also. This combined together, we get the following objective function

$$\mathrm{V}\left(\underline{\mathbf{x}}, \underline{\mathbf{u}}\right) = \frac{1}{2} \sum_{i=1}^{T} \left(k_q \cdot (e_{x,[i]}^2 + e_{y,[i]}^2) + k_s \cdot (u_{x,[i-1]}^2 + u_{y,[i-1]}^2) \right) \tag{14}$$

where k_q and k_s are constants, whose ratio is the only parameter of the MPC and determines how wildly the system behaves. The error at the time $t = 0$ is determined only by the initial condition and doesn't depend on on the input action $\underline{\mathbf{u}}$. Because the function is to be optimized, this error can be left out. The equation 14 can be rewritten in a matrix form using penalizing matrices $\mathbf{Q}$ and $\mathbf{S}$

$$\mathrm{V}\left(\underline{\mathbf{x}}, \underline{\mathbf{u}}\right) = \frac{1}{2} \sum_{i=1}^{T} \left(\mathbf{e}_{[i]}^T \mathbf{Q} \mathbf{e}_{[i]} + \mathbf{u}_{[i-1]}^T \mathbf{P} \mathbf{u}_{[i-1]} \right) \tag{15}$$

where $\mathbf{e}_{[t]} = \mathbf{x}_{[t]} - \mathbf{x}_{d,[t]}$ is the error of all states at time t and matrices $\mathbf{Q}$ and $\mathbf{S}$ are

$$\mathbf{Q} = \begin{bmatrix} k_q & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_q & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{P} = \begin{bmatrix} k_p & 0 \\ 0 & k_p \end{bmatrix}. \tag{16}$$

This form of $\mathbf{Q}$ allows to penalize only position errors and ignore the velocity and acceleration errors. To ensure, that the function $\mathrm{V}\left(\underline{\mathbf{x}}, \underline{\mathbf{u}}\right)$ is strictly convex, the matrix $\mathbf{Q}$ must be positive semi-definite ($\mathbf{Q} \succeq 0$) and $\mathbf{P}$ must be positive definite ($\mathbf{P} \succ 0$) –cite Tom–. The matrix $\mathbf{Q}$ has it's eigenvalues 0 and $-k_q$, therefore $k_q \geq 0$. The eigenvalues of $\mathbf{P}$ are $-k_p$, so $k_p > 0$.

Equation 15 can be rewritten once more to

$$\mathrm{J}(\underline{\mathbf{u}}) = \frac{1}{2} \underline{\mathbf{u}}^T \underbrace{\left(\hat{\mathbf{B}}^T \hat{\mathbf{Q}} \hat{\mathbf{B}} + \hat{\mathbf{P}} \right)}_{\hat{\mathbf{H}}} \underline{\mathbf{u}} + \underline{\mathbf{u}}^T \underbrace{\left(\hat{\mathbf{Q}} \hat{\mathbf{B}} \right)^T \left(\hat{\mathbf{A}} \mathbf{x}_{[0]} - \underline{\mathbf{x}}_d \right)}_{\hat{\mathbf{c}}}, \tag{17}$$