1 Introduction

Unmanned aerial vehicles (UAV) have become very popular in the last few years. It is mainly thanks to the multicopters, especially quadcopters and their dropping prizes. UAVs are used in many areas, for instance they allow many amateur film makers to capture aerial shots and often substitute expensive helicopters in professional movie shooting. Many companies, such as DJI, have developed quadcopters for professional and amateur film making as shown in Fig. 12. The quadcopters are also used in military[**?**] mainly for search and rescue missions. Some quadcopters[**?**], carrying defibrillators, are used for medical assistance in cases of heart failures, for their much faster reaction time compared with ambulance vehicles. The future use of quadcopters is almost limitless, from monitoring cities, delivering packages, to collecting air condition data.

Quadcopter's body is simple compared with a classical helicopter. It has a rigid body and four propellers with fixed pitch angles. In this thesis, the term UAV will refer to a quadcopter.

Many quadcopters are equipped with a global position system (GPS). This allows a better outdoor control of the aircraft and additional functions, such as position hold. Quadcopters are usually directly controlled by people, but in many applications the ability of autonomous flight is useful. For example, in case of the signal loss, the UAV can autonomously return to the takeoff place and even land.

Another system Vicon has been developed for more precise indoor localization, using on the wall cameras. The UAV communicates wirelessly to the on the ground computer. The UAV's controller is usually also implemented in the on the ground computer, because of the UAV's limited payload. These constraints limit the UAV flight only to the laboratory conditions.

There has been a demand for an on board controller, operating outside laboratory conditions, with the controller depending only on the on board computer and sensors. This would allow a fully autonomous flight, where some high level unit computes trajectories for a single or multiple UAV's.

1.1 Problem statement

The task of this thesis is to design and implement a model predictive controller capable of precise trajectory tracking and obstacle avoidance. The trajectory will be given by a higher level planning unit and the UAV will track the trajectory, unless obstacles appear. In this case, the Model Predictive Control (MPC) automatically avoids the obstacles. This is useful for swarms of UAVs and other trajectory planning algorithms, which do not have to take into consideration either the actual control of the particular UAV or avoiding unexpected obstacles. The appropriate input actions will be found as a mathematical optimization of linearly constrained quadratic

Figure 1: DJI Phantom 3

function.

The UAV will be fully autonomous with the localization, obstacle detection and computing happening solely on board. This allows the UAV to operate outside laboratory conditions.

1.2 Previous work

Since 2011, the laboratory of Intelligent and Mobile Robotics (IMR) group at FEE CTU has been developing UAV controllers. At the early stages the group concentrated on simulations of swarms and formations. In the following years hardware platforms were developed.

The hardware platform, which is further described in Sec. 9.1, was developed in [**?**]. This platform was used for real flight experiments. An MPC, capable of tracking a trajectory was developed for this platform, but it was not designed to avoid obstacles. For obstacle avoidance different approaches have to be used mainly in the mathematical optimization part.

A system for sensor px4flow [**?**] allowes to measure speed and, combined with a previously implemented state observer[**?**], allows position measurements.

A system WhyCon[**?**][**?**] is used for obstacle detection. It allows to detect multiple circular markers, shown in Fig. 5, used in the experiments to represent obstacles.

1.3 Related work

Many laboratories have been developing different obstacle avoidance systems. Most of them develop precise trajectory tracking. If an obstacle appears, a higher level planning unit updates the desired trajectory.

Such a system can be used for on board 3D mapping and obstacle

avoidance[**?**] at the same time. This system is used mainly for mapping unknown areas.

Obstacles do not have to be detected by the cameras, ultrasonic sensors can be used instead. A system for indoor obstacle avoidance[**?**] has been developed to allow computer assisted maneuvering in static indoor environment.

Another obstacle avoidance system is based on the potential field principle, which is a popular model for swarm control[**?**]. In this model, the acceleration of the UAV is the sum of fictional forces, which repulse the UAV from obstacles and push it towards the desired position.

Not all obstacle avoidance systems are designed for multicopters. A flapping wing micro areal vehicle(MAV) system has been also equipped by an on board obstacle avoidance system [**?**], using trajectory updating.

2 UAV dynamics

For understanding the UAV's controller, it is important to understand the physical properties of the UAV. The UAV has 6 degrees of freedom which are the position x, y, z and the rotation pitch(θ), roll (ψ) and yaw(ϕ). The UAV is equipped with the KK2 board providing basic stabilization. The inputs, that are used in this thesis are roll and pitch rotation references. After linearisation at the equilibrium at the horizontal position, with rotations close to the equilibrium, it can be said, that $\psi \propto \ddot{x}$ and $\theta \propto -\ddot{y}$, which are the accelerations in each axis. The KK2's inputs are desired ψ and θ and output is the voltage on the individual motors. This controller provides a time-invariant linear system. The continuous transfer diagram is shown in the Fig. 2

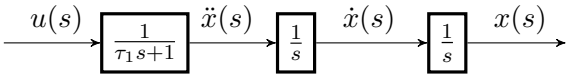


Figure 2: Diagram of the continuous system.

The input action $u(s)$, as a from the MPC, is given to the KK2 and transformed to the the UAV's tilt, which corresponds to the real acceleration $\ddot{x}(s)$, and then twice integrated to the position $x(s)$.

2.1 Coordinate systems

This whole thesis will consider 2D coordinate system only. In these 2 dimensions, there is an aileron axis x with the direction to the right and the elevator axis y with the direction forwards. These both axes are perpendicular. There are two coordinate systems which will be used. The first system

is a standard world coordinate system W with the origin usually at the starting point of the UAV. The second system is a coordinate system U, which is a system with the origin in the center of the mass of the UAV. This is for example the coordinate system of detecting obstacles, which are observed relatively to the UAV by on-board cameras. Coordinate system U is created only by the translation of the system W by the vector $\vec{r} = (\Delta x, \Delta y)$. There is no rotation between the two coordinate systems, so the axes 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 rotated over time, the model would no longer be linear and the control of this system would be much more complex, therefore slower. These coordinate systems transformations follow equations

$$\begin{aligned} x^{(W)} &= x^{(U)} + \Delta x \\ y^{(W)} &= y^{(U)} + \Delta y \\ \dot{x}^{(W)} &= \dot{x}^{(U)} + \Delta \dot{x} \\ \dot{y}^{(W)} &= \dot{y}^{(U)} + \Delta \dot{y} \\ \ddot{x}^{(W)} &= \ddot{x}^{(U)} + \Delta \ddot{x} \\ \ddot{y}^{(W)} &= \ddot{y}^{(U)} + \Delta \ddot{y}. \end{aligned} \tag{1}$$

In the following text the world coordination system W will be used unless stated otherwise. The UAV has its own proportions. However, it is complicated to take into consideration the whole UAV's body. It is easier to proximate the UAV's body with a single mass point in its center. The orientation of the UAV is constant and will not be taken into consideration, thus the position of the UAV can be described by two coordinates.

The altitude of the UAV is controlled separately. There are several reasons to do that. The first reason is, that many applications treat the altitude differently and enforce constant altitude for long periods of time. For example, in building interiors, the altitude is constant most of the time. The desired trajectory is also usually given in 2D and the altitude is described separately. The second reason is, as mentioned above, an MPC is very demanding on computing time. To save computational capacity, a standard PID controller can be used instead of an MPC for controlling the altitude. The altitude PID controller has already been implemented [**?**] and it is not a part of this thesis.

2.2 State observer

An MPC is very sensitive to a data noise and errors. Inaccurate initial condition can result in a bad prediction because of the double integration of acceleration into position. To work properly, the MPC requires very accurate

initial conditions. These are position, speed and acceleration in both axes. A Kalman estimator has been implemented[**?**] to estimate states of the UAV and disturbances of the acceleration. The inputs of the state observer are the system's input action and measured speed by a camera sensor.

2.3 Single axis model

The UAV model has been analyzed[**?**]. Thanks to the symmetrical body of the UAV, both axes share the same model. For aileron axis, the states take form of $\mathbf{x}_x = (x, \dot{x}, \ddot{x})^T$ and $\mathbf{x}_y = (y, \dot{y}, \ddot{y})^T$ for elevator axis, where x is the aileron and y is the elevator position. The aileron and elevator models are mathematically identical. The discrete state space model takes form of system matrices $\mathbf{A}_s, \mathbf{B}_s$ as

$$\mathbf{x}_{x,y,[t+1]} = \mathbf{A}_s \mathbf{x}_{x,y,[t]} + \mathbf{B}_s u_{x,y,[t]}, \tag{2}$$

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

$$\mathbf{A}_s = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & \Delta t \\ 0 & 0 & p_1 \end{bmatrix}, \mathbf{B}_s = \begin{bmatrix} 0 \\ 0 \\ p_2 \end{bmatrix}, \tag{3}$$

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

3 Model predictive control formulation

An MPC is an advanced regulator. It uses prediction of future states of a system to determine system input actions. This prediction runs constantly in a loop. Because of its computational demands, it is used mainly in processes with long time constants. A motivation for its development was to control various chemical processes, where the computational time was not limiting. Using an MPC for controlling a UAV is a challenge, because it is hard to implement on embedded hardware. Controlling a real time system, such as UAV, requires regulation in tens of Hz, giving the hardware very little time to compute such a complex problem.

An MPC requires the system's state space model, initial condition and, unlike other controllers, a sequence of desired future states. An advantage of an MPC is ability to apply large variety of constraints, which can be useful for example in the chemical processes control. On the other hand, an MPC is sensitive to the model's inaccuracy and to sensory noise.

Output of an MPC is not only the desired input action for the next time step, but also predicted input actions and predicted behavior of the system in

the whole prediction horizon for the T following time steps. Unlike standard controllers like a PID, an MPC can adjust the input action based on future demands.

The MPC controller for tracking desired trajectory was developed [**?**] in the past. The goal of the thesis is to design an MPC controller with capability to avoid obstacles. Obstacles are not previously known and can change position.

The MPC formulates optimization problem, that has to be solved. This problem takes usually form of Linear Programming (LP) or, in the case of this thesis, linearly constrained Quadratic Programming (QP). The only difference between trajectory and not detecting obstacles and collision avoidance is applying constraints.

3.1 Extended model

For complex MPC constraints, such as position constraints for obstacle avoidance, position of the UAV in one axis is a function of the position in the other axis. For these kinds of constraints axes can not be treated separately and more complicated system description is needed. The state space system must be preserved, connecting both identical systems for each axis into one system extending Eq. (4) into

$$\mathbf{x}_{[t+1]} = \mathbf{A}\mathbf{x}_{[t]} + \mathbf{B}\mathbf{u}_{[t]}, \tag{4}$$

where $\mathbf{u}_{[t]} = (u_{x,[t]}, u_{y,[t]})^T$ is an input vector containing elevator and aileron system inputs at the time t. Extended state vector

$$\mathbf{x}_{[t]} = (x_{[t]}, \dot{x}_{[t]}, \ddot{x}_{[t]}, y_{[t]}, \dot{y}_{[t]}, \ddot{y}_{[t]})^T \tag{5}$$

contains positions x, y and their derivatives at the time t.

By connecting these two single axis systems, we get the following state space matrices of the extended system

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

3.2 System prediction

The MPC algorithm is based on predicting future states based on initial condition and system input. Such a general equation can be derived from the Eq. (4). With a simple substitution it is possible to get the prediction of the states at the time $t = 2$:

$$
\begin{aligned}
\mathbf{x}_{[1]} &= \mathbf{A}\mathbf{x}_{[0]} + \mathbf{B}\mathbf{u}_{[0]}, \\
\mathbf{x}_{[2]} &= \mathbf{A}\mathbf{x}_{[1]} + \mathbf{B}\mathbf{u}_{[1]} \\
&= \mathbf{A} \cdot (\mathbf{A}\mathbf{x}_{[0]} + \mathbf{B}\mathbf{u}_{[0]}) + \mathbf{B}\mathbf{u}_{[1]} \\
&= \mathbf{A}^2\mathbf{x}_{[0]} + \mathbf{A}\mathbf{B}\mathbf{u}_{[0]} + \mathbf{B}\mathbf{u}_{[1]}.
\end{aligned} \tag{7}
$$

The Eq. (7) can be rewritten in a more general way:

$$
\mathbf{x}_{[t]} = \mathbf{A}^t\mathbf{x}_{[0]} + \sum_{i=1}^{t-1} \mathbf{A}^i\mathbf{B}\mathbf{u}_{[i-1]} + \mathbf{B}\mathbf{u}_{[0]}. \tag{8}
$$

Let's combine the sequence of the predicted states into a column vector

$$
\underline{\mathbf{x}} = (\mathbf{x}_{[1]}^T, \mathbf{x}_{[2]}^T, ..., \mathbf{x}_{[T]}^T)^T \tag{9}
$$

of the length $6T$ and the sequence of the inputs into a column vector

$$
\underline{\mathbf{u}} = (\mathbf{u}_{x,[0]}, \mathbf{u}_{y,[0]}, \mathbf{u}_{x,[1]}, \mathbf{u}_{y,[1]}, ..., \mathbf{u}_{x,[T-1]}, \mathbf{u}_{y,[T-1]})^T. \tag{10}
$$

of size $2T$. With this notation, Eq. (8) can be represented as a matrix multiplication equation

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

Using this new notation, Eq. 11 can be rewritten in a simpler form as

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

4 MPC implementation

As mentioned in the Sec. 3, an MPC formulates a quadratic optimization problem. This problem can be either constrained or unconstrained, depending on the fact if constraints are demanded. Let's first solve the unconstrained problem, where the obstacles are not involved.

4.1 Trajectory

For an MPC to compute input actions, there must be 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 besides position. However, the UAV desired velocity is already given by the desired positions at certain time as the distance

$$d = \sqrt{(x_{d,[t]} - x_{d,[t+1]})^2 + (y_{d,[t]} - y_{d,[t+1]})^2}. \tag{13}$$

The same method can be applied for acceleration. Therefore, the velocity and acceleration are demanded in the sequence of desired positions, rather than the desired system states. The desired velocity and acceleration are then ignored and it can hold any value, for example 0. The $\mathbf{x}_{d,[t]}$ then takes form of $\mathbf{x}_{d,[t]} = (x_{d,[t]}, 0, 0, y_{d,[t]}, 0, 0)^T$. When creating the desired trajectory, it must be always taken into consideration, that the desired positions hold also information about velocity and acceleration.

4.2 Objective function

As in many other areas of engineering, this problem can be split into two independent parts. The first part is to create an objective function and second is to minimize it. The objective function's (sometimes called a cost function) value describes, how good the particular solution is. The set of particular solutions is the domain of the objective function. In this case the solution takes form of a vector $\underline{\mathbf{u}}$, which can be directly transformed into predicted trajectory using the Eq. 12. The lower the value of the objective function is, the better is the particular solution. The solution with the minimal objective function's value is called an optimal solution $\underline{\mathbf{u}}^*$.

The goal of unconstrained MPC is to track a given trajectory as precisely as possible. This means, that we want to minimize the distance between all the predicted and the desired positions. This deviation in both axes can be computed as

$$\begin{aligned} e_{x,[t]} &= x_{[t]} - x_{d,[t]} \\ e_{y,[t]} &= y_{[t]} - y_{d,[t]} \end{aligned} \tag{14}$$

Using this kind of deviation has many downsides. The obvious one is, that it penalizes deviation in only one direction and favors the other. If the distance is to be used, the absolute value has to be taken. This function is not differentiable [**?**] and solving this task is complicated. Differentiability is a useful property. Capability to compute gradient of the final objective function results in fast task solving by applying gradient descend described

in Sec. 6.4. Besides that it is not possible to get good results if penalizing the deviation linearly.

Previous experiments have show, that using of the square of the deviation, is beneficial in many ways. This preserves the condition of not prioritizing one direction deviation. The function is also easily differentiable. Another advantage is penalizing big distances disproportionately more while ignoring minute deviations. This also describes our requirements, where the exact tracking of the trajectory is not as important as eliminating big deviations from the trajectory. If a simple sum of $e^2_{x,[t]}$ and $e^2_{y,[t]}$ is applied, the system will behave wildly, generating high input actions to correct the deviation.This is not in the capabilities of the real system and could result in an unstable control. Therefore input actions must also be penalized. When combined together, we get the following objective function

$$\mathrm{V}\left(\mathbf{e}_x, \mathbf{e}_y, \underline{\mathbf{u}}\right) = \frac{1}{2}\sum_{i=1}^{T}\left(k_q \cdot (e^2_{x,[i]} + e^2_{y,[i]}) + k_s \cdot (\mathbf{u}^2_{x,[i-1]} + \mathbf{u}^2_{y,[i-1]})\right), \quad (15)$$

where $\mathbf{e}_x = (e_{x,[1]}, e_{x,[2]}, ..., e_{x,[T]})^T$ and $\mathbf{e}_y = (e_{y,[1]}, e_{y,[2]}, ..., e_{y,[T]})^T$. The ratio of the constants k_q and k_s is the only parameter of the MPC and determines how wildly the system behaves. The deviation at the time $t = 0$ is determined only by the initial condition and does not depend on on the input action $\underline{\mathbf{u}}$. Because the function is to be optimized, this error can be left out. The Eq. (15) can be rewritten in a matrix form using penalizing matrices $\mathbf{Q}$ and $\mathbf{S}$

$$\mathrm{V}\left(\underline{\mathbf{e}}, \underline{\mathbf{u}}\right) = \frac{1}{2}\sum_{i=1}^{T}\left(\mathbf{e}^T_{[i]}\mathbf{Q}\mathbf{e}_{[i]} + \mathbf{u}^T_{[i-1]}\mathbf{P}\mathbf{u}_{[i-1]}\right), \quad (16)$$

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

$$\mathbf{Q} = \begin{bmatrix} k_q & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_q & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{P} = \begin{bmatrix} k_p & 0 \\ 0 & k_p \end{bmatrix}. \quad (17)$$

This form of $\mathbf{Q}$ allows to penalize only position deviations and ignore the velocity and acceleration deviations. For a fast optimization, the objective function has to be convex. To ensure 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 ($\mathbf{P} \succ 0$). 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 $k_p > 0$. In some MPC algorithms

the last deviation is penalized with higher weight, forcing the system to end up in the last state. It turned out, that this can be counterproductive in obstacle avoidance system for reasons that will be discussed later.

It is important to transform the Eq. (16) into a matrix multiplication form. Let's first introduce the matrices $\hat{\mathbf{Q}}$ and $\hat{\mathbf{P}}$ as

$$\hat{\mathbf{Q}} = \begin{bmatrix} \mathbf{Q} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} & \dots & \vdots \\ \mathbf{0} & \dots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{Q} \end{bmatrix}, \hat{\mathbf{P}} = \begin{bmatrix} \mathbf{P} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{P} & \dots & \vdots \\ \mathbf{0} & \dots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{P} \end{bmatrix}. \tag{18}$$

If the last error was to be penalized more, the last matrix $\mathbf{Q}$ on the diagonal of the matrix $\hat{\mathbf{Q}}$ would consist of higher constants k_q. Lets rewrite the Eq. (16) using the Eq. (12):

$$\begin{aligned} \mathrm{J}(\underline{\mathbf{u}}) &= \frac{1}{2}\left(\underline{\mathbf{e}}^T \hat{\mathbf{Q}} \underline{\mathbf{e}} + \underline{\mathbf{u}}^T \hat{\mathbf{P}} \underline{\mathbf{u}}\right) \\ &= \frac{1}{2}\left((\hat{\mathbf{A}}\mathbf{x}_{[0]} + \hat{\mathbf{B}}\underline{\mathbf{u}} - \underline{\tilde{\mathbf{x}}})^T \hat{\mathbf{Q}} (\hat{\mathbf{A}}\mathbf{x}_{[0]} + \hat{\mathbf{B}}\underline{\mathbf{u}} - \underline{\tilde{\mathbf{x}}}) + \underline{\mathbf{u}}^T \hat{\mathbf{P}} \underline{\mathbf{u}}\right) \\ &= \frac{1}{2}\left(2(\hat{\mathbf{Q}}\hat{\mathbf{B}})^T \hat{\mathbf{A}}\mathbf{x}_{[0]} \underline{\mathbf{u}} + \underline{\mathbf{u}}^T \hat{\mathbf{B}}^T \hat{\mathbf{Q}} \hat{\mathbf{B}} \underline{\mathbf{u}} - 2(\hat{\mathbf{Q}}\hat{\mathbf{B}})^T \underline{\tilde{\mathbf{x}}}\, \underline{\mathbf{u}} + \underline{\mathbf{u}}^T \hat{\mathbf{P}} \underline{\mathbf{u}} + const.\right) \end{aligned} \tag{19}$$

In the Eq. (19), all constant elements have been put into the *const* part. Because the goal is to minimize the objective function $\mathrm{J}(\underline{\mathbf{u}})$, the parts of the equation that do not depend on the $\underline{\mathbf{u}}$ can be left out. The Eq. (19) can be rewritten as

$$\mathrm{J}(\underline{\mathbf{u}}) = \frac{1}{2}\underline{\mathbf{u}}^T \underbrace{\left(\hat{\mathbf{B}}^T \hat{\mathbf{Q}} \hat{\mathbf{B}} + \hat{\mathbf{P}}\right)}_{\hat{\mathbf{H}}} \underline{\mathbf{u}} + \underbrace{\left(\hat{\mathbf{Q}}\hat{\mathbf{B}}\right)^T \left(\hat{\mathbf{A}}\mathbf{x}_{[0]} - \underline{\tilde{\mathbf{x}}}\right)}_{\hat{\mathbf{c}}} \underline{\mathbf{u}}. \tag{20}$$

The task can be formulated as a linearly constrained quadratic programming

$$\begin{aligned} \min_{\underline{\mathbf{u}} \in \mathrm{R}^{kM}} \quad & \mathrm{J}(\underline{\mathbf{u}}) = \frac{1}{2}\underline{\mathbf{u}}^T \hat{\mathbf{H}} \underline{\mathbf{u}} + \hat{\mathbf{c}} \underline{\mathbf{u}} \\ \text{s.t.} \quad & \mathbf{A}_c \underline{\mathbf{u}} \leq \mathbf{B}_c. \end{aligned} \tag{21}$$

The constraints will be further discussed in section 4.4. They can be ignored for unconstrained MPC.

4.3 Solving QP unconstrained

We want to solve the problem formulated in the Eq. 21 while ignoring the constraints. For this task to have a single optimal solution $\underline{\mathbf{u}}^\star$, the objective

function has to be convex. Therefore the matrix $\hat{\mathbf{H}}$ has to be positive semi definite. This is guaranteed, because the matrices $\mathbf{Q}$ and $\mathbf{S}$ were chosen to be positive semi-definite ($\mathbf{Q}, \mathbf{S} \succeq 0$) and $\mathbf{P}$ positive definite ($\mathbf{P} \succ 0$). For solving this task, we need the gradient of the objective function [**?**] as

$$\nabla \mathrm{J}(\underline{\mathbf{u}}) = \hat{\mathbf{H}}\underline{\mathbf{u}} + \hat{\mathbf{c}}. \tag{22}$$

The gradient $\nabla \mathrm{J}(\underline{\mathbf{u}})$ is a column vector of the same size as $\underline{\mathbf{u}}$ with the opposite direction to the global minimum, so the $-\nabla \mathrm{J}(\underline{\mathbf{u}})$ is a vector pointing the direction towards the global minimum $\underline{\mathbf{u}}^\star$. To solve this task, a gradient descent algorithm can be used, but also analytic solution can be found.

Because the $\hat{\mathbf{H}}$ is positive semidefinite, the quadratic form $\frac{1}{2}\underline{\mathbf{u}}^T\hat{\mathbf{H}}\underline{\mathbf{u}}$ is convex. The term $\hat{\mathbf{c}}\underline{\mathbf{u}}$ is a linear function and adding it to a convex function does not change the convexity of the task. Because of this, we know, that a local minimum is also a global minimum. The local minimum can be found as

$$\begin{aligned} \nabla \mathrm{J}(\underline{\mathbf{u}}^\star) = \hat{\mathbf{H}}\underline{\mathbf{u}}^\star + \hat{\mathbf{c}} &= 0 \\ \hat{\mathbf{H}}\underline{\mathbf{u}}^\star &= -\hat{\mathbf{c}} \\ \underline{\mathbf{u}}^\star &= -\hat{\mathbf{H}}^{-1}\hat{\mathbf{c}}. \end{aligned} \tag{23}$$

The inverse of $\hat{\mathbf{H}}$ can be computed, because $\hat{\mathbf{H}}$ is positive-definite, therefore regular. This is guaranteed by the definition of positive-definite property that its leading principal minors are all positive [**?**]. On-board computation is fast, because matrix $\hat{\mathbf{H}}^{-1}$ does not depend on the system state and is only the function of the constants k_q, k_p and the system model. It can be generated in advance and stored as a constant in the UAV's read-only memory. Finding the $\underline{\mathbf{u}}^\star$ is only matter of creating vector $\hat{\mathbf{c}}$ and computing $-\hat{\mathbf{H}}^{-1}\hat{\mathbf{c}}$.

4.4 System constraints

One of the greatest advantages of an MPC is ability to apply large variety of constraints. As mentioned in Sec. 3, the MPC uses linearly constraint quadratic programming for finding the input action prediction. There can be many different kinds of constraints and the MPC has to fulfill all of them. These constraints take form of linear inequalities

$$\vec{\omega}_i \cdot \underline{\mathbf{u}} \leq b_i, \quad i \in \{1, 2, ..., m\}, \tag{24}$$

where $\vec{\omega}_i$ is a vector of the length 2T and b_i is a scalar, both describing the constraint number i, m is the total number of constraints applied. Each constraint takes form of a half space in a space $\mathbf{R}^{2T}$ constraining it by hyperplane with a normal vector $\vec{\omega}_i$ shifted by b_i. To apply multiple constraints at the same time, the Eq. (24) can be rewritten in matrix form as

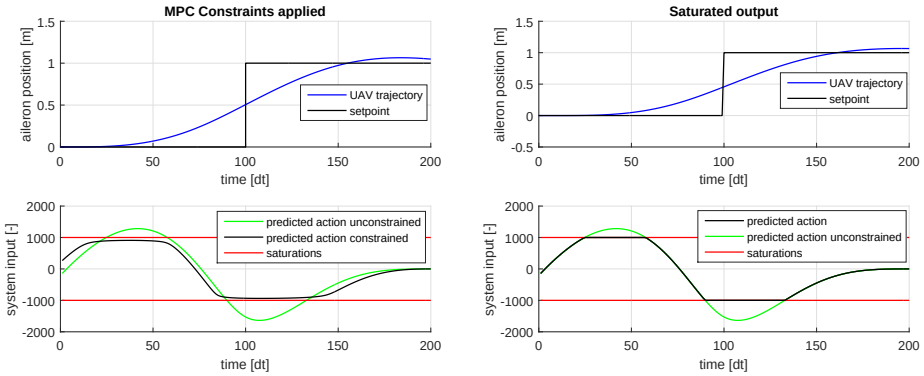
$$\mathbf{A}_c \underline{\mathbf{u}} \leq \mathbf{B}_c, \tag{25}$$

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

$$\mathbf{A}_c = \begin{bmatrix} \vec{\omega}_1 \\ \vec{\omega}_2 \\ \dots \\ \vec{\omega}_m \end{bmatrix}, \mathbf{B}_c = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{bmatrix}. \tag{26}$$

Each line of the matrix $\mathbf{A}_c$ together with a particular member of $\mathbf{B}_c$ represents one constraint.

4.5 Input constraints



(a) constrained MPC. **(b)** saturated MPC.

Figure 3: Step response of the system using two approaches of saturation.

In control engeneering a common problem that has to be delt with is a system saturation. This means that the real system is linear only on a certain range of inputs and has limited capapibilities [**?**]. For example a motor can spin in a certain maximum speed despite input voltage. If the system receives too high input actions from the controller, it can get damaged. This can happen for example in a proportional derivative (PD) control, when the difference between real and desired output changes rapidly in time, for example because of a sensory noise. A standard solution for this problem is a simple saturation of the output of the controller. This is a simple solution and for many tasks it is sufficient. However, because the controller does not take into consideration this saturation, the system can behave incorrectly. The advantage of an MPC is that the controller can 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. This is achieved by the MPC constraints. The system saturation thresholds are u_{min} and u_{max}. We can see in Fig. (3b) simply saturated input action with the computed trajectory and int Fig. (3a) the input action computed by the MPC using system constraints.

5 Collision Avoidance

Collision avoidance in the MPC is realized through constraining the position. This means, that predicted position has to lie in a certain allowed space and every violation of this space is considered to be a collision. Algorithm used in this section prohibits the UAV to get outside of this allowed space.

5.1 Obstacles

Through this thesis, obstacles are represented by circles in 2D only. They have 3 parameters: position x_{obs}, position y_{obs} and radius r_{obs}. Despite the fact that obstacles can have various shapes, we simplify them to allow simpler computations. Because the MPC runs constantly in a loop and reacts to the changing environment, it works also with moving objects. Until now the UAV body has been approximated with one mass point without considering its real size. Since it would be difficult to calculate, whether the UAV's body has collided, we use Minkovski sum.

The UAV body can be approximated with a circle wrapping the whole body with the same center (x, y) and radius R_{UAV}. If the distance between the position of the UAV and the edge of the allowed space is less or equal R_{UAV}, it can be stated, that the UAV will not collide.

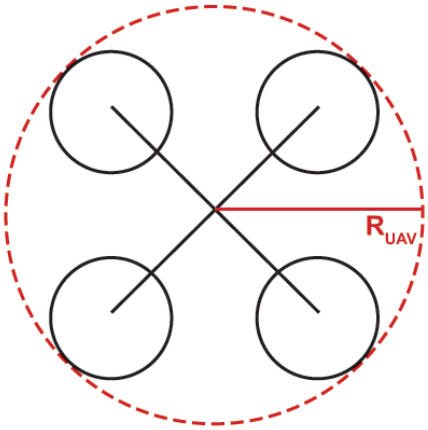


Figure 4: UAV dimensions

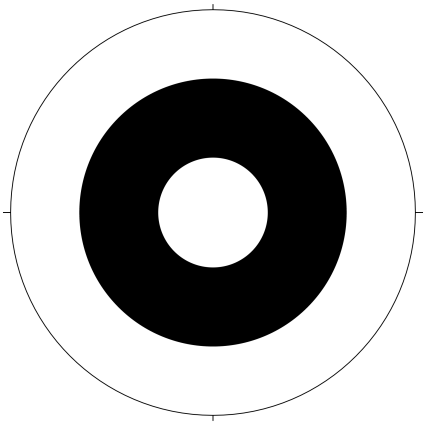


Figure 5: Recognizable marker

$$(x_{obs} - x)^2 + (y_{obs} - y)^2 \geq (R_{UAV} + r_{obs})^2. \tag{27}$$

From now on we will still compute with the UAV as a single mass point and every obstacle's radius r_{obs} extend to $r_{obs} + R_{UAV}$.

Let's make an assumption, that the UAV's trajectory is a line created by connecting all the predicted positions. This line has a zigzag shape. Because the UAV can not change vector of speed immediately, the real trajectory has to have a continuous first time derivative. Also because the second derivative(acceleration) of the UAV's position is a result of the UAV's pitch and roll, which also can not be changed immediately, the second derivative is also continuous. In short, the real trajectory intersects all the predicted positions, but there is a small deviation caused by the first and second derivatives of the real system. The real trajectory has to be smooth up to the second derivative. To be able to approximate the real trajectory by the simplified zigzag trajectory, we need to make sure, that the predicted positions are much closer to each other than the obstacles sizes. This is easy to achieve, because the time between the predicted positions is $\Delta t = 1/70s$, which with the maximum speed of 0.35 ms^{-1} is distance of 5 mm.

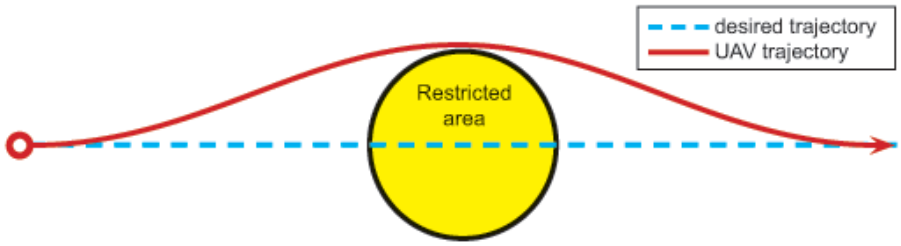


Figure 6: UAV avoidance in non-convex space.

The forward approach would be to say, that every trajectory is feasible, if any predicted position does not collide with any obstacle. This approach would search for any trajectory in the allowed space. However, if the obstacle is just a simple circle, the allowed space is non-convex as shown in Fig. 6. As mentioned in Sec. **??**, for keeping the quadratic programming convex, the allowed space for UAV has to be convex. This turned out to be the main disadvantage of using the MPC for obstacle avoidance.

This is also the reason, why higher penalization of the last state deviation is not desirable. The allowed space is not ideal and the prediction of the last position is not close to the real one.

5.2 Creating allowed space

A convex allowed space for trajectory planning has to be created to approximate the real world situation. However, there can not be enough good convex representation, that is robust at the same time. The approach for creating a convex allowed space in this thesis is restricting any position in

a half plane 'behind' any obstacle as shown in Fig. 7. When combining multiple obstacle constraints, this space becomes polytope defined by a set of half planes. They are updated by the relative positions of the obstacles in each MPC iteration. These positions continuously change due to the UAV's movement and the sensory noise. Let's first look at a situation with a single obstacle.

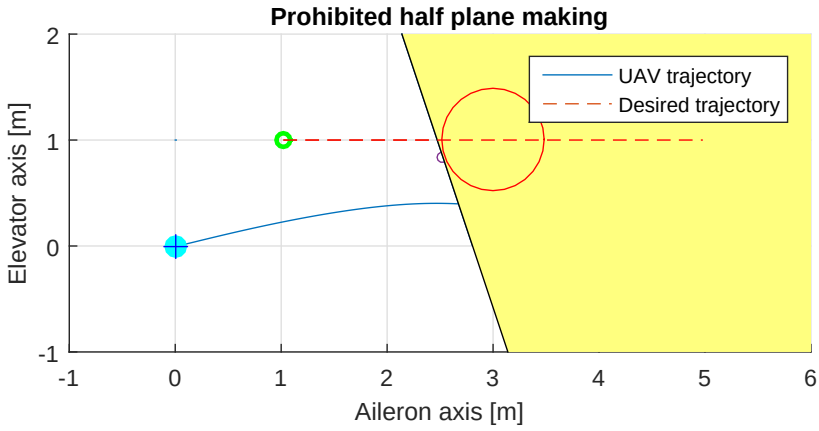


Figure 7: Creating a prohibited half plane

Half plane can be represented by the equation $y \leq kx + q$ where x and y are the allowed positions of the UAV and k is the slope of the line and q is the bias of the line. These equations can be rewritten as

$$s(y - kx - q) \leq 0, \tag{28}$$

where $s = \pm 1$. This is an inequality defining one obstacle by a single half plane. These constants k, q and s can be found as

$$\begin{aligned} k &= -\frac{y_{obs} - y_{UAV}}{x_{obs} - x_{UAV}} \\ b_x &= (x_{obs} - x_{UAV}) \cdot \big(1 - \frac{r_{obs} + R_{UAV}}{\sqrt{(x_{obs} - x)^2 + (y_{obs} - y)^2}}\big) \\ b_y &= (y_{obs} - y_{UAV}) \cdot \big(1 - \frac{r_{obs} + R_{UAV}}{\sqrt{(x_{obs} - x)^2 + (y_{obs} - y)^2}}\big) \\ q &= b_y - kb_x \\ s &= sign(y). \end{aligned} \tag{29}$$

The b_x and b_y is a position of the intersection of the constraining line and the obstacle circle with radius $R_{UAV} + r_{obs}$.

The constrains for position have been found. These constraints have to be further transformed into an input action constraints. This is an inverse transformation to the Eq. 12, where input actions have been transformed to position and its derivatives. For the purposes of MPC, matrices $\mathbf{A}_c$ and $\mathbf{B}_c$ have to be created to fit the condition 25. The vector $\underline{\mathbf{x}}$ from the Eq. 12 contains all predicted states for both axes. The predicted positions x and y have to be separated into the position column vectors $\mathbf{x}_p$ and $\mathbf{y}_p$ of size T as

$$\mathbf{x}_p = \begin{bmatrix} x_{[1]} \\ x_{[2]} \\ \dots \\ x_{[T]} \end{bmatrix}, \mathbf{y}_p = \begin{bmatrix} y_{[1]} \\ y_{[2]} \\ \dots \\ y_{[T]} \end{bmatrix}, \tag{30}$$

where $x_{[t]}$ is the predicted position x at the time t and $y_{[t]}$ is the predicted position y at the time t. These vectors are related to $\underline{\mathbf{u}}$ by the matrices $\hat{\mathbf{A}}_x$, $\hat{\mathbf{B}}_x$, $\hat{\mathbf{A}}_y$, $\hat{\mathbf{B}}_y$ as

$$\begin{aligned} \mathbf{x}_p &= \hat{\mathbf{A}}_x \mathbf{x}_{[0]} + \hat{\mathbf{B}}_x \underline{\mathbf{u}} \\ \mathbf{y}_p &= \hat{\mathbf{A}}_y \mathbf{x}_{[0]} + \hat{\mathbf{B}}_y \underline{\mathbf{u}}, \end{aligned} \tag{31}$$

where the matrices $\hat{\mathbf{A}}_x, \hat{\mathbf{A}}_y \in R^{T\times 6}$ and $\hat{\mathbf{B}}_x, \hat{\mathbf{B}}_y \in R^{T\times 2T}$ are submatrices of their corresponding matrices $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$, $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ as

$$\begin{aligned} \mathbf{B}_x &= \begin{bmatrix} \mathbf{B}_{1,1:2} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{1,1:6}\mathbf{B} & \mathbf{B}_{1,1:2} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{1,1:6}^{(T-1)}\mathbf{B} & \mathbf{A}_{1,1:6}^{(T-2)}\mathbf{B} & \dots & \mathbf{B}_{1,1:2} \end{bmatrix}, \mathbf{A}_x = \begin{bmatrix} \mathbf{A}_{1,1:6} \\ \mathbf{A}_{1,1:6}^2 \\ \vdots \\ \mathbf{A}_{1,1:6}^{(T-1)} \end{bmatrix}, \\ \mathbf{B}_y &= \begin{bmatrix} \mathbf{B}_{4,1:2} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{4,1:6}\mathbf{B} & \mathbf{B}_{4,1:2} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{4,1:6}^{(T-1)}\mathbf{B} & \mathbf{A}_{4,1:6}^{(T-2)}\mathbf{B} & \dots & \mathbf{B}_{4,1:2}. \end{bmatrix}, \mathbf{A}_y = \begin{bmatrix} \mathbf{A}_{4,1:6} \\ \mathbf{A}_{4,1:6}^2 \\ \vdots \\ \mathbf{A}_{4,1:6}^{(T-1)} \end{bmatrix}. \end{aligned} \tag{32}$$

$\mathbf{A}_{i_1,i_2:i_3}$ is a matlab-like notation describing a submatrix of the matrix $\mathbf{A}$ with the line i_1 and columns from i_2 to i_3. The Eq. (28) is defined only for one position. These matrices are constant, so they can be computed in advance and stored in the read-only memory. It is important, that the UAV does not collide in any predicted position in the prediction horizon. Then this equation is rewritten in a vector form

$$s(\mathbf{y}_p - k\mathbf{x}_p - \mathbf{q}) \leq 0, \tag{33}$$

where $\mathbf{q}$ is the column vector of size T and every member is q. After the substitution of Eq. (31) into Eq. (33) we get

$$s\big(\hat{\mathbf{A}}_y\mathbf{x}_{[0]} + \hat{\mathbf{B}}_y\underline{\mathbf{u}} - k(\hat{\mathbf{A}}_x\mathbf{x}_{[0]} + \hat{\mathbf{B}}_x\underline{\mathbf{u}}) - \mathbf{q}\big) \leq 0$$
$$\underbrace{s\big(\hat{\mathbf{B}}_y - k\hat{\mathbf{B}}_x\big)}_{\hat{\mathbf{A}}_c}\underline{\mathbf{u}} + \underbrace{s\big(\hat{\mathbf{A}}_y\mathbf{x}_{[0]} - k\hat{\mathbf{A}}_x\mathbf{x}_{[0]} - k\mathbf{q}\big)}_{\mathbf{B}_c} \leq 0. \tag{34}$$

Until now, all the half planes of the obstacles have been computed from the relative position of the initial position of the UAV. This means, that for example the last predicted position will be still constrained by the previous position. However, the last position is not known, because it is computed from the vector $\underline{\mathbf{u}}$ that is being searched for. If an assumption is made, that the predicted trajectory is similar to the trajectory in the previous step, the previous can be used. The MPC loop runs in tens of Hz and the UAV does not change its position much in one iteration. This improvement is done by computing the previous trajectory using Eq. (12). Then the constants k_i, q_i, s_i need to be found for each predicted position $i, i \in \{1, ..., T\}$ using the Eq. (29). Then the Eq. (33) can be rewritten in a matrix form as

$$\hat{\mathbf{s}}(\mathbf{y}_p - \hat{\mathbf{k}}\mathbf{x}_p - \mathbf{q}) \leq 0, \tag{35}$$

where $\hat{\mathbf{s}}$ and $\hat{\mathbf{k}}$ are square diagonal matrices of size T.

$$\hat{\mathbf{s}} = \begin{bmatrix} s_1 & 0 & \dots & 0 \\ 0 & s_2 & \dots & \vdots \\ 0 & \dots & \ddots & \vdots \\ 0 & \dots & \dots & s_T \end{bmatrix}, \hat{\mathbf{k}} = \begin{bmatrix} k_1 & 0 & \dots & 0 \\ 0 & k_2 & \dots & \vdots \\ 0 & \dots & \ddots & \vdots \\ 0 & \dots & \dots & k_T \end{bmatrix}, \mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_t \end{bmatrix}. \tag{36}$$

Because of high computational demands, this algorithm has not been implemented. The MPC loop must run as fast as possible. The standard solution uses only the initial condition for knowing position and computes the constants from Eq. (29) only once. This is a fast operation. If the constants for each future position were to be computed, it would be necessary to compute the predicted trajectory from the previous prediction using Eq. (12) and compute the constraining constants T times.

Let's now consider a case with multiple obstacles. As described in the section 4.4, the lines of constraining matrices $\mathbf{A}_c$ and $\mathbf{B}_c$ enforce independent conditions. The computation can be done independently for N obstacles using the same procedure, marking the matrices $\mathbf{A}_{c,i}$ and $\mathbf{B}_{c,i}$, where $i \in 1, 2, ..., N$. The final constraining matrices would take form of

$$\mathbf{A}_c = \begin{bmatrix} \mathbf{A}_{c,1} \\ \mathbf{A}_{c,2} \\ \dots \\ \mathbf{A}_{c,N} \end{bmatrix}, \mathbf{B}_c = \begin{bmatrix} \mathbf{B}_{c,1} \\ \mathbf{B}_{c,2} \\ \dots \\ \mathbf{B}_{c,N} \end{bmatrix}. \tag{37}$$

Now the task is fully defined and can be given to an LCQP solver to find the optimal input action $\underline{\mathbf{u}}$.

6 Solving Linearly Constrained QP

Now the objective function J($\underline{\mathbf{u}}$) with the matrices $\hat{\mathbf{H}}$, $\hat{\mathbf{c}}$ with the constraining matrices $\mathbf{A}_c$ and $\mathbf{B}_c$ defined in the Eq. (21) has been created. Solving the unconstrained problem using the inverse $\hat{\mathbf{H}}^{-1}$ matrix has already been discussed. The constrained problem can not be solved analytically, and iterative methods are commonly used. Consequently, the solution is usually not optimal, but close to optimal.

6.1 Safe margin

in this particular case of obstacle avoidance, one special property of the solver is desirable. Instead of finding a strictly feasible solution (an $\underline{\mathbf{u}}$, that lies on at least one constraining hyper plane), a solution with some margin is more suitable. Strictly feasible solution leads to touch of the obstacle. The MPC finds the closest avoidance trajectory, where at least one state prediction is equal to the position constraint. If a safe distance from the obstacle is required, there are two different solutions.

The first one is to make R_{UAV} bigger than the real UAV's body and to take the strictly feasible solution. However, this solution has a disadvantage. The mathematical model will still touch the obstacle, even that the UAV body is actually smaller and will fly in a longer distance from the obstacle. Because of the sensory noise, the measured relative position of the obstacle is continuously changing. At the time when the UAV model touches the obstacle, the measured obstacle position can change and move little closer to the UAV. The UAV mathematically moves inside the obstacle and the initial condition does not satisfy the constraints any more. The constraints are not defined for the initial condition but for the first time step. The optimization algorithm would try to find an input action to escape the constraint in the first time step. The position is a second integration of the input and each integration takes one time step. This means, that the input action first influences the position in two time steps ahead.

Even a small amount of noise can result in a task formulation that has no solution. Choosing the strictly feasible solution is not applicable. Making the final solution $\underline{\mathbf{u}}$ further from the constraining hyper planes solves this problem well. The UAV will fly within some safe distance from the obstacle and if the obstacle changes its position as a result of the sensory noise, the initial condition will still satisfy the position constraints. This is the solution, that will be used in this thesis. To find this kind of solution, a special optimization method has to be used.

6.2 LCQP solvers

Because of the convexity of the function and the convexity of the constraints, the solution is either a global minimum of the function $\mathrm{J}(\underline{\mathbf{u}})$, or lies on the constraining border. There are several methods of solving the constraint optimization problem.

One method is called Active set method [**?**]. The constraints are solved almost exactly. This method supposes that the minimum lies on the constraint border. It searches only strictly feasible solutions. Some constraints play no role for the final result and some parts of constraints are prohibited by other constraints, which brings up a high complexity. This method uses analytic approach, and so it works well on noise-free data and when the exact minimum is desired. However, the UAV's optical registration of obstacles and optical localization brings a high noise. Active set method is not a good fit for this problem.

Another method is a method of Lagrange multipliers. This is a very widely used strategy in many different tasks. It also assumes that the minimum lies on the constraining border. This is enforced by the constraints defined as $g_i(\underline{\mathbf{u}}) = 0, i \in \{1, 2, ..., M\}$ for M constraint. This method requires that the objective function and the constraints are differentiable. It uses a function called Lagrangian $\mathcal{L}$

$$\mathcal{L}(\underline{\mathbf{u}}, \lambda_1, \lambda_2, ..., \lambda_M) = \mathrm{J}(\underline{\mathbf{u}}) - \sum_{i=1}^{M} \lambda_i g_i(\underline{\mathbf{u}}) \tag{38}$$

Then a solution must be found for $\nabla\mathcal{L} = 0$. This would mean a necessity to solve a set of $T + M$ equations

$$\frac{\partial \mathrm{J}(\underline{\mathbf{u}})}{\partial u_j} = \sum_{i=1}^{M} \frac{\partial \lambda_i g_i(\underline{\mathbf{u}})}{\partial u_j} \quad and \quad \frac{\partial \mathrm{J}(\underline{\mathbf{u}})}{\partial \lambda_i} = \sum_{i=1}^{M} \frac{\partial \lambda_i g_i(\underline{\mathbf{u}})}{\partial \lambda_i} \tag{39}$$
$$i \in \{1, 2, ..., M\}, \quad j \in \{1, 2, ..., 2T\}$$

This is a simple idea that the gradient of the objective function $J(\underline{\mathbf{u}})$ is perpendicular to all constraints. If not, it means that there is a better solution. However, in our case the perpendicularity is not always true. The solution can lie in an intersection of two half planes. The problem is also in the defining the constraints, which are not in the form of inequality but equality and it is required to satisfy all of them. There has to exist an intersection of all the constraints. This is not very likely, because our constraints are defined by hyper planes. If even two hyper planes are parallel, this algorithm does not find any solution. Such hyper planes have been described in the section 4.5, where a maximum thrust forwards and backwards is constrained. If defined as an equality, this would mean that the thrust must

be maximum forwards and maximum backwards at the same time, which is logically impossible. The method of Lagrange multipliers has to be also rejected.

Another method is a Penalty method. It is used for example in optimizing support vector machine (SVM) algorithm. It penalizes crossings of the constraints. It modifies the objective function by adding a penalty function, if a constraint is violated. The task is modified as

$$min \;\; \mathrm{J}(\underline{\mathbf{u}}) + \sigma \sum_{i=0}^{N} g(c_i(\underline{\mathbf{u}})), \tag{40}$$

where $g(c_i(\underline{\mathbf{u}})) = min(0, c_i(\underline{\mathbf{u}})^2)$ is a positive number if the constraint c_i has been violated, and zero otherwise. This function is not easily differentiable, but after some adjustments, gradient descent can be used. The major disadvantage is that the final solution likely violates the constraints. Violating these constraints in our case means either giving higher input actions or a collision. It is obvious, that this algorithm can not be used either.

6.3 Barrier method

The final method is called a Barrier method has been implemented in this thesis. It modifies the objective function by adding a barrier function $g(\underline{\mathbf{u}})$, which penalizes points close to the constraining border. Then a gradient descent algorithm is used to find the local minimum of the function $f(\underline{\mathbf{u}}) = \mathrm{J}(\underline{\mathbf{u}}) + g(\underline{\mathbf{u}})$. The function $g(\underline{\mathbf{u}})$ has to possess several properties.

- It has to be defined for all feasible $\underline{\mathbf{u}}$, but It does not have to be defined elsewhere.
- It should be convex to preserve convexity of the sum of the function $J(\underline{\mathbf{u}})$ and $g(\underline{\mathbf{u}})$.
- To make sure the constraints will be satisfied, it has to equal to infinity when approaching the border.
- It has to be differentiable over all the domain to have the possibility to use the gradient descent method.
- From these assumptions it can be said that without the loss of generality that the barrier function should be a function of the distance from the constraining hyper plane.

The second requirement supposes that an addition of two convex functions is a convex function. Let's choose some convex function $f_1(x), f_2(x)$

and some variable x. Then it has to be proven, that $f_3(x) = f_1(x) + f_2(x)$ is also a convex function. This is proven by proving the definition of convexity

$$f_3(tx_1 + (1-t)x_2) \leq tf_3(x_1) + (1-t)f_3(x_2), \quad t \in < 0, 1 >, \tag{41}$$

where x_1 and x_2 are any points form the domain. Then because of the convexity of f_1 and f_2,

$$\begin{aligned} f_1(tx_1 + (1-t)x_2) &\leq tf_1(x_1) + (1-t)f_1(x_2) \\ f_2(tx_1 + (1-t)x_2) &\leq tf_2(x_1) + (1-t)f_2(x_2). \end{aligned} \tag{42}$$

The inequality 41 is actually a sum of the two lines of the inequalities 42. Adding these 2 lines preserves the inequality. It can be said, that the inequality 41 is valid and that the result of summing convex functions is a convex function.

The distance in a space of any dimension of a point from a hyper plane is

$$d = abs\left(\frac{\omega_i \cdot \underline{\mathbf{u}} + b_i}{\sqrt{\vec{\omega_i}^T \cdot \vec{\omega_i}}}\right) \tag{43}$$

The notation$|\vec{\omega}_i|$ represents the Euclidean norm as $\sqrt{\vec{\omega}_i \cdot \vec{\omega}_i^T}$ The function f can be then defined for $\underline{\mathbf{u}}$ or d depending on the requirements. After trying various forms of functions $g(\underline{\mathbf{u}})$, the function

$$g(\underline{\mathbf{u}}) = k_g \sum_{i=1}^{M} \frac{1}{d_i} \tag{44}$$

has been chosen, where k_g is a constant. It has been experimentally chosen to be 10^4. This function possesses all the required properties. If the point is close to any of the constraining hyper planes, the barrier function grows rapidly. It could look unusual that this function is defined for all points, not only for the feasible set. And it does not make sense in the not feasible area. Again, it penalizes less the points further from the plane then the closer points. This is not a difficulty, because the function will be only used in the feasible set.

The Fig. 8 shows, how the barrier method works in one dimension. It moves the location of the strictly feasible local minimum away from the prohibited space to a a new local minimum. This is a desired property, because the strictly feasible solution can result in an inappropriate behavior as mentioned in the beginning of this section.

The Fig. 9 shows the function $\mathrm{J}(\underline{\mathbf{u}}) + g(\underline{\mathbf{u}})$ with the 2 dimensional domain. The real optimizing function is in very high dimension and can not be graphed. Principal component analysis (PCA), which is an algorithm for reducing space dimensions by projecting objects to a created basis, can not be used, because it highly deforms the space.

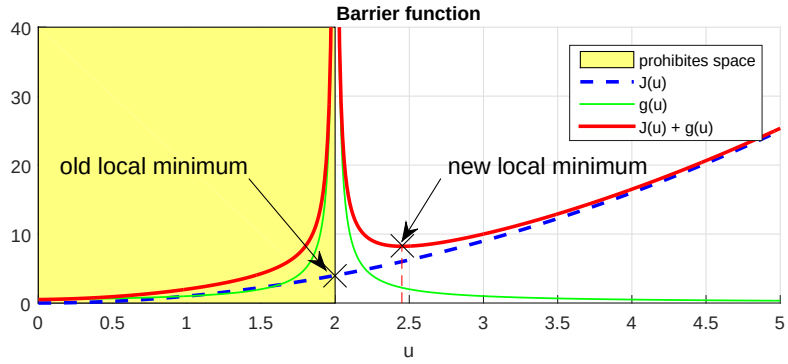


Figure 8: Barrier function with one variable

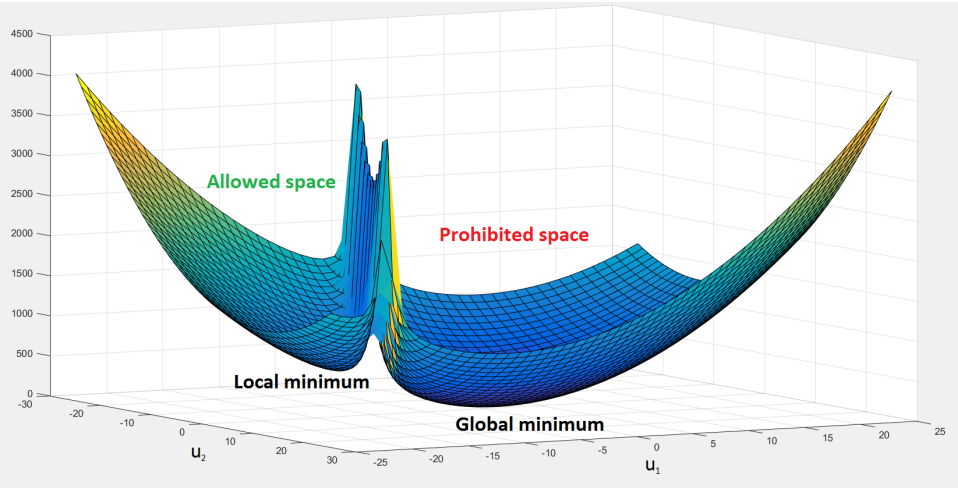


Figure 9: The final optimization function with two variables

6.4 Gradient descent

The optimization problem is solved with slightly modified gradient descent. This is an often used algorithm for optimizing functions. It has two big requirements. The first one is, that the function must be differentiable. This has been satisfied by choosing quadratic function as the objective function and hyperbola as the barrier function. The second requirement is, that the function must be convex. Instead of the function $\mathrm{J}(\underline{\mathbf{u}})$, the gradient descent optimizes the function $f(\underline{\mathbf{u}}) = \mathrm{J}(\underline{\mathbf{u}}) + g(\underline{\mathbf{u}})$ where $g(\underline{\mathbf{u}})$ is the barrier function from Eq. (44). It is an iterative method, where in each iteration a point $\underline{\mathbf{u}}$ is moved in the negative direction of the gradient of the function $f(\underline{\mathbf{u}})$. This means, that

$$\underline{\mathbf{u}}_{[t+1]} = \underline{\mathbf{u}}_{[t]} - k_d \nabla f(\underline{\mathbf{u}}_{[t]}), \tag{45}$$

where $\underline{\mathbf{u}}_{[t]}$ is the $\underline{\mathbf{u}}$ at the t-th iteration. The size of the gradient step at i-th iteration is $k_d \nabla f(\underline{\mathbf{u}})$. This algorithm runs for a certain number of iterations until the final $\underline{\mathbf{u}}$ is used as the solution. For using this algorithm, the gradient $\nabla f(\underline{\mathbf{u}})$ is needed to be computed as

$$\begin{aligned} \nabla f(\underline{\mathbf{u}}) &= \nabla \mathrm{J}(\underline{\mathbf{u}}) + \nabla g(\underline{\mathbf{u}}) \\ \nabla \mathrm{J}(\underline{\mathbf{u}}) &= \hat{\mathbf{H}}\underline{\mathbf{u}} + \hat{\mathbf{c}} \\ \nabla g(\underline{\mathbf{u}}) &= k_g \sum_{i=1}^{N} \frac{\vec{\omega}_i.^2 sig(\vec{\omega}_i \cdot \underline{\mathbf{u}} + b_i)}{|\vec{\omega}_i| \cdot (\vec{\omega}_i \cdot \underline{\mathbf{u}} + b_i)}, \end{aligned} \tag{46}$$

where $\vec{\omega}_i$ is the i-th line of the matrix $\mathbf{A}_c$ and b_i is the i-th member of the column matrix $\mathbf{B}_c$. The notation $\vec{\omega}_i.^2$ represents a line vector from the matrix $\vec{\omega}_i^T \cdot \vec{\omega}_i$ diagonal.

6.5 Feasibility algorithm

Now, that we have the $\nabla f(\underline{\mathbf{u}})$, we have to find the initial search point $\underline{\mathbf{u}}_{[0]}$. Because the function $f(\underline{\mathbf{u}})$ is convex only on the feasible domain, the $\underline{\mathbf{u}}_{[0]}$ has to lie there. To find a feasible $\underline{\mathbf{u}}$ is not as simple task, as it may look. Especially if the UAV flies among many obstacle with higher velocity, the feasible polytope can be small. A zero vector means flying in the initial direction and back thrust is not robust and could not work. An algorithm for finding a feasible $\underline{\mathbf{u}}$ has been developed, that has been named a feasibility algorithm. While designing this algorithm, a useful feature was required. This algorithm takes a $\underline{\mathbf{u}}_{[0]}$, that may or may not be feasible. It then finds a feasible alternative of the point, that is very close to the given one. It will be very useful for the speed and accuracy of the whole MPC, that will be discussed later. The iterative algorithm inputs are matrices $\mathbf{A}_c$, $\mathbf{B}_c$ and a $\underline{\mathbf{u}}_{[0]}$. The algorithm can be formulated in the following steps:

1. Find the first constraint $\vec{\omega}_i \cdot \underline{\mathbf{u}}_{[t]} \leq b_i$, that is violated. If such a constraint does not exist, successfully terminate the algorithm and return the $\underline{\mathbf{u}}_{[t]}$.

2. Count the distance d from the constraint i and compute the normalized gradient $\mathbf{dg}_n = \frac{\nabla g(\underline{\mathbf{u}})}{|\nabla g(\underline{\mathbf{u}})|}$. This is a vector in the direction towards the constraint.

3. Move the search point along the gradient such as $\underline{\mathbf{u}}_{[t+1]} = \underline{\mathbf{u}}_{[t]} + k_f \cdot d \cdot \mathbf{dg}_n$ and continue to step 1.

The constant k_f should be between 1 and 2. If the k_f is 1, the $u_{[t+1]}$ ends exactly on the constraint. This is not good, because the hyperbola $g(\underline{\mathbf{u}})$ is not defined there. If the constant k_f is higher than 2, the algorithm

could oscillate, especially between 2 parallel constraints. From experiments, a constant $k_f = 1.5$ has been chosen and works well.

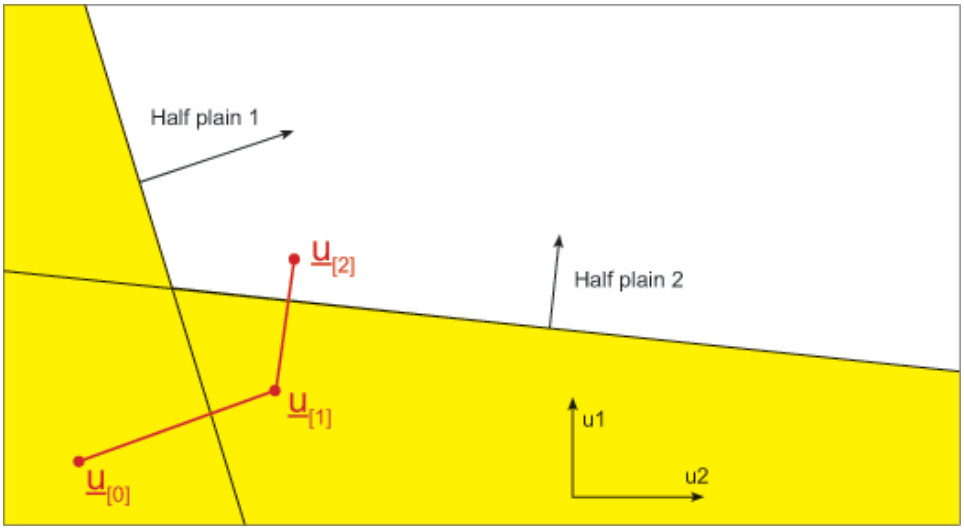


Figure 10: Feasibility algorithm.

The Fig. 15 shows, how the algorithm can correct a $\underline{\mathbf{u}}$, that is not feasible into a feasible one with a position very close to the initial $\underline{\mathbf{u}}_{[0]}$.

When the algorithm for tracking trajectory starts, the UAV has no velocity and the gradient descent algorithm can use the initial search point $\underline{\mathbf{u}}_{[0]}$ as a zero vector. It means, that it will stay at one place, so there is no collision. The MPC loop runs fast and if every time a new gradient descend is initialized with a random $\underline{\mathbf{u}}_{[0]}$, it would take a long time to come anywhere close to the local minimum. The knowledge, that the UAV moves relatively slow compared with the MPC loop can be very useful, because the function $f(\underline{\mathbf{u}})$ is very similar in the next step. A final prediction from previous MPC step can be used as an initial search point $\underline{\mathbf{u}}_{[0]}$. This incredibly increases the speed and accuracy of the gradient descend algorithm. This could not have been done without the feasibility algorithm. A final $\underline{\mathbf{u}}$ from previous MPC step can be feasible no more with a different UAV position and velocity. This is, where the required feature of the feasibility algorithm comes handy. It moves the previous $\underline{\mathbf{u}}$ to a new location, that is still close to the previous $\underline{\mathbf{u}}$ and as a result it is close to the local minimum.

There can be a situation, where a feasible solution does not exist. In this case, the loop would run infinitely. This can be achieved especially, if the sensory noise is too high that a distance to an obstacle is evaluated closer, then the $R_{UAV} + r_obs$. With this initial condition there is usually no solution as described in the section 6.1. To prevent the UAV to freeze, a counter needs to be added to terminate the feasibility algorithm after running too long.

6.6 Implementation of the gradient descend

The gradient descend is implemented by the equation 45. The only difference is, that in every iteration, every $\underline{\mathbf{u}}_{[t]}$ is checked for feasibility by applying the feasibility algorithm. The reason to do this is that the gradient step can be long and after moving the search point it can step over the barrier function and end up outside the feasible domain. If the gradient step constant k_d was set too small, it would take longer time to converge to the local minimum. The $k_d = 150$ based on simulations and experiments was chosen.

A simulation has been run for no barrier function using only the $\nabla \mathrm{J}(\underline{\mathbf{u}})$ with the feasibility algorithm and still providing fairly good results. This approach has not been used in later simulations or experiments.

Choosing the total number of iterations of the gradient algorithm generally depends on the requirements of accuracy vs control frequency. Experimentally has been observed, that a small number of total iterations can be helpful by providing a low pass filter in situations with high sensory noise. Experiments have shown, that 100 iterations is a good compromise between the accuracy and the control frequency.

7 Move Blocking

The whole MPC algorithm is very demanding on computational time and memory. This can result in a very slow frequency of the MPC controller. A matrix multiplication of a n by m matrix and a column vector of the length m has the computational complexity $O(nm)$. For example the matrix $\hat{\mathbf{B}}$ is the size of $6T$ by $2T$ and the vector $\underline{\mathbf{u}}$ is the length of $2T$. This makes the computational complexity $O(12T^2)$. When considering, that the system model is created for $dt = 1/70s$, for prediction of 2 seconds for a simple matrix multiplication, processor would have to do at least $12 \cdot 140^2 = 235\ 200$ operations. With the size of 4 bytes for one float, the matrix $\hat{\mathbf{B}}$ would also take almost 1 MB of memory. This is not that much for PC simulation, but it is a lot for the custom board flying on the UAV with only 192 kB RAM, 2 MB of ROM and 168 MHz processor with one instruction for float multiplication.

7.1 Move blocking implementation

The computation complexity can be greatly reduced by reducing the T, which is the total number of future predictions. The discretization of the prediction is very soft, meaning, that the Δt is small. In the maximum speed 0.35 ms^{-1} this is a distance 0.5 mm between predicted positions. There is no reason to predict position 70 times a second. A solution would be to change the system's Δt, but there is a better method. Move blocking algorithm allows us to choose exactly which time steps will be used. This allows only

small number of variables to cover long prediction horizon. Because the input actions tend to take form of a continuous function, the algorithm will generate constant input action between those predicted as shown in Fig. 16. Let's introduce a matrix $\mathbf{U}$ of the height $2T$ and width $2T_n$, where T_n is the new number of predicted states. This matrix determines, which time steps will be predicted.

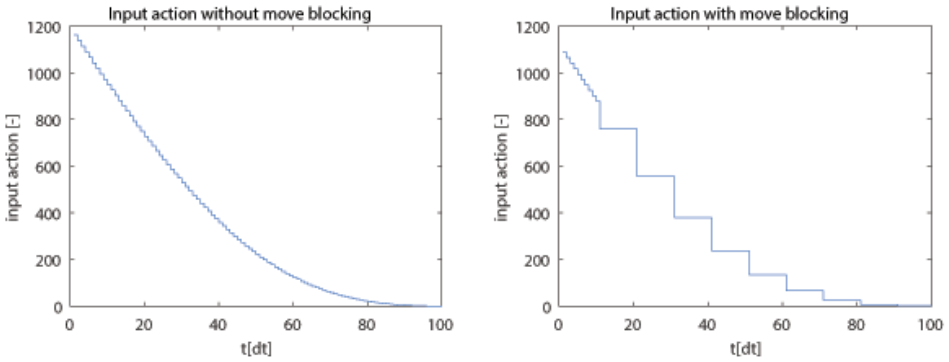


Figure 11: Move blocking algorithm in one axis

$$\underline{\mathbf{u}} = \underbrace{\begin{bmatrix} \begin{bmatrix}1\\\vdots\\1\end{bmatrix} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \begin{bmatrix}1\\\vdots\\1\end{bmatrix} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \begin{bmatrix}1\\\vdots\\1\end{bmatrix} \end{bmatrix}}_{\mathbf{U}} \underline{\mathbf{u}}_r, \quad \underline{\mathbf{u}}_r = \begin{bmatrix} \mathbf{u}_{[t_1]} \\ \mathbf{u}_{[t_2]} \\ \vdots \\ \mathbf{u}_{[T_r]} \end{bmatrix}, \tag{47}$$

$\underline{\mathbf{u}}_r$ is the reduced column vector of the length $2T_r$. Its members are time predictions at certain times $t_1, t_2, ..., T_r$. All matrices introduced in MPC must be reduced. The process is quite complicated and will not be described step by step.

In the Fig. 16 have been used prediction times $1, 2, ..., 9, 10, 20, 30, ..., 100$. The number of the prediction times is T_r. Using this particular vector, the T_r is 19 instead of T as 100. The acceleration of matrix multiplication can be computed as $(T/T_r)^2 = 27.7$. This means 27.7 times faster algorithm. The move blocking also creates the same memory savings with some matrices. This is a crucial improvement of the MPC algorithm, allowing to be

implemented on an embedded hardware. It can cover a longer prediction horizon without storing big matrices.

The penalization matrices $\mathbf{Q}$ and $\mathbf{P}$ need to be edited differently. Without move blocking algorithm, all predicted positions and inputs are penalized evenly. If now the predicted inputs represents inputs of different length, the penalization must be set accordingly. The main diagonal has to have its members individually multiplied by the corresponding constant representing the number of time steps.

8 Simulation

Developing the collision avoidance system on a paper and then programming it directly to the UAV's on-board computer is a naive approach that likely would not work. To minimize the probability of an expensive crash happening, the algorithm must be tested as much as possible.

During the development of the MPC, all steps were being tested by Matlab simulations. This program is good for these purposes, because it is optimized for matrix multiplication, which is the core of the MPC computations. The implemented matlab simulation consists of two separate modules: the Environment module and the UAV module.

8.1 The Environment module

The Environment module simulates the flight. This simulates the physical world and the sensors of the UAV. It is initialized by the obstacles positions, the UAV's initial condition and desired trajectory. It takes care of the UAV's dynamics and simulates the flight. The main program runs in a loop. It receives the computed input actions from the UAV module and computes the UAV's movement. Then it updates the relative obstacles positions, the absolute UAV position and the velocity and the desired trajectory. It then gives this information as an input to the UAV module. The environment module also sets the parameters for the UAV module, such as penalization of the position errors k_q and the input k_p, the indexes of the move blocking time predictions and gradient descend iterations and step size. This makes it easier to tune the overall constants.

It has also the ability to visualize the UAV, obstacles, the desired trajectory, the predicted trajectory and the convex feasible space. This graph updates with the loop and simulates the whole flight.

8.2 The UAV module

The second module simulates the the MPC controller on UAV's on-board computer. It has the ability to compute the input actions based on the

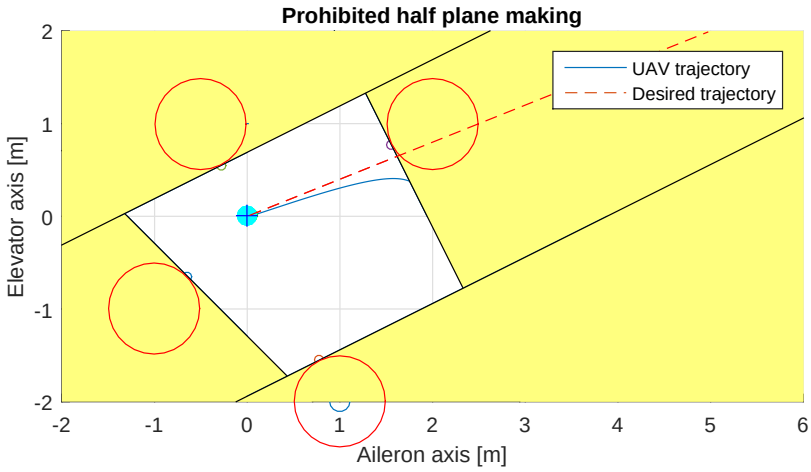


Figure 12: Simulation visualization.

simulated sensory data given by the Environment module. It consists of two parts.

The first part is the MPC problem definition, which creates the constraining matrices $\mathbf{A}_c$ and $\mathbf{B}_c$ from the information about the initial condition and obstacles positions. Only the objective function matrix $\hat{\mathbf{c}}$ is created, because the $\hat{\mathbf{H}}$ is a constant and can be stored in memory.

The second part is the LCQP solver. This solver has been implemented by the method described in the Sec. 6.6. Its inputs are the gradient step size, the total number of iterations and the problem defining matrices, which are identical to the function quadprog from the optimization toolbox. It returns slightly different results, because of the safe margin condition.

9 Hardware

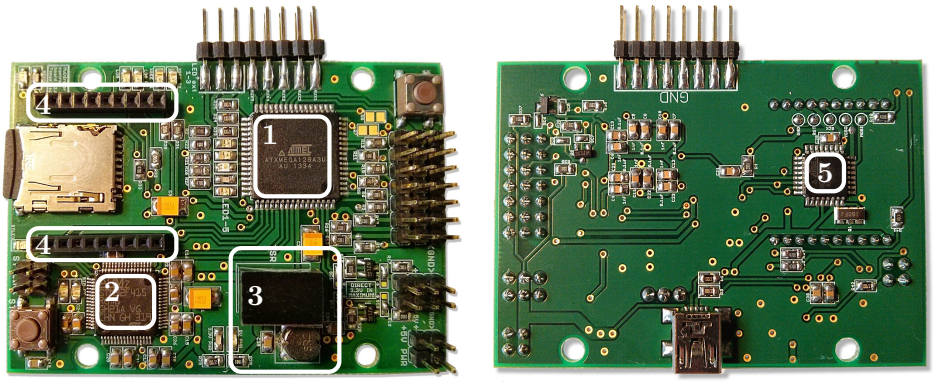
In Fig. 13 is shown a quadcopter used for experiments. It can be controlled remotely by an operator that can fly it manually or turn on the MPC. It consists of the rigid body, propellers with motors, KK2 stabilization board, the custom board and obstacle detection system.

9.1 UAV custom board

The UAV custom board [**?**], shown in the Fig. 14, is a computer with limited computational power compared to standard PC. This board has been

Figure 13: UAV body

designed specially for MPC control. It is programmable in the language C. It has two computational units: xMega and STM. For debugging and data logging is a socket for XBee and micro SD card slot. This board sticks to the philosophy of distributed computing.



(a) board's top **(b)** board's bottom

Figure 14: Custom control board v.2, key components are placed at follows: 1 – xMega, 2 – STM, 3 – switching power supply, 4 – socket for XBee, 5 – data logging MCU.

The xMega is a slow computational unit. It has 8-bit architecture with 32 MHz clock and 8 kB SRAM. It is mainly handling the communication between sensors and STM. The communication between peripherals goes through 7 UART ports, where sensors or a computer can be connected. With the help of the program Putty, a simple messages can be exchanged between the computer and the board. Most of the protocols have been already designed, but minor adjustments had to be done to fit the exact requirements of collision avoidance. For example the communication between the xMega and STM had to be extended to transfer information about ob-

stacle positions. The xMega also stores the desired trajectory or the location of the setpoint and it sends this data to the STM.

The STM is a powerful 32-bit unit with 168 MHz clock and 192 kB of RAM and 2 MB ROM. It supports basic operations of 4 byte float units with one instruction. The STM runs three tasks, as shown in Fig. (15).

- CommTask is responsible for communicating between the xMega and STM.
- KalmanTask is the state estimator task responsible for estimating positions, velocity, acceleration and input error based on information from the px4flow sensor and the inputs. When it computes the momentary system state, it sends it to the MPCTask.
- MPCTask is the task, that computes the MPC controller. The MPCTask is triggered by the KalmanTask's finished state computation, but only if the previous iteration has finished. This allows the MPC to run slower, than 70 Hz. In this case the input actions are held, until they are replaced by later computed $\underline{\mathbf{u}}$. This mechanism allows tuning between the speed and accuracy of the MPC, because there is no need to regulate the UAV with 70 Hz.

Each task uses a third of the CPU time and 16 kB RAM. Because of limited computational power, the MPCtask has to be programmed very efficiently.

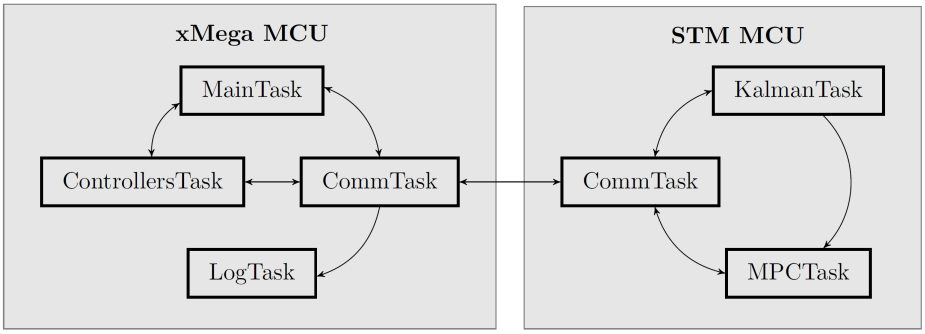


Figure 15: Block diagram of information ow between tasks of xMega and STM MCUs.

9.2 Obstacles detection

A system WhyCon[**?**][**?**] is used for obstacle detection. It uses 3 Mobius Actioncam cameras, two pointing to the sides and one forwards. This allows obstacle detection at 270°. Each camera has the capability of recording

in the resolution 1920x1080, but for faster performance the video is shoot in 1280x720 and processed in 640x480. The system is capable of precise detecting of multiple circular markers shown in Fig. 5. The 3-axis positions of these markers are then sent by UART to the custom board, where the xMega receives the information. The image processing runs in a loop on a computational unit NVIDIA Jetson TK1.

9.3 MPC hardware implementation

The exact algorithm, as written in matlab, has been rewritten to the C code. This was the most time demanding part of this thesis. The programmed code uses a CMatrixLib library, that allows simple matrix and vector operations. This library has been extended by more functions for the purposes of this thesis. Because the RAM is limited, the matrices must be divided into constant matrices, that are stored in ROM, and the changing matrices, which have allocated memory in RAM.

The constant matrices depend only on the system model and move blocking algorithm. These matrices are: $\hat{\mathbf{A}}_x$, $\hat{\mathbf{A}}_y$, $\hat{\mathbf{B}}_x$, $\hat{\mathbf{B}}_y$, $(\hat{\mathbf{Q}}\hat{\mathbf{B}})^T$, $\hat{\mathbf{A}}$ and $\hat{\mathbf{H}}$. They had been reduced by the move blocking algorithm, generated in matlab and stored in the ROM memory of the STM unit.

The UAV's max speed has been set to 0.35 ms^{-1}. This is enforced by editing the desired trajectory. If the trajectory doesn't violate the maximum speed, it is used unchanged. In the other case, the trajectory positions are moved closer to the UAV. If given only setpoint, the desired trajectory is created as a straight line with the maximum speed.

The input action can not be infinite and some method of constraint has to be applied. Applying the constraints in the MPC, as described in the Sec. 4.5 slows the algorithm. A saturation of the output was used instead. Experiments have shown, that this method works well.

9.4 Hardware in the loop

This is a method of testing embedded hardware. The custom board is connected to a computer instead of the UAV's body. The board's inputs are provided by the computer and outputs are sent back. The board is in the same situation, as if it would be while flying. This has been simulated in two different ways.

The first, was connecting the board to the computer using the putty terminal. The inputs have been sent through the terminal and outputs were received. These outputs were compared with the matlab simulations, to check if they match. The UAV has been tested in various situations, such as different initial conditions, desired trajectories and obstacle positions. This allowed to test and debug various segments of the code, as well as the while MPC algorithm. This method also allowed to test xMega as well as STM.

After successfully finishing this testing, it was sure, that the implemented algorithm behaves exactly in the same way as the simulation.

The second experiment was connecting the board to the UAV, without the ability of controlling the motors. It received inputs from the UAV's sensors, such as velocity and obstacle positions. The board was connected to matlab using the wireless Xbee. The Xbee communication protocol was extended for the important information to be transferred, such as the predicted input actions. A matlab visualization allowed to see the predicted trajectory in real time without the need to be connected to the UAV with a cable. This allowed testing the algorithm in real situations without worrying of damaging the UAV. It also allowed the final testing of the communication between the custom board and the on-board sensors. The frequency of the MPC loop has been measured close to 20 Hz. This is an sufficient regulator frequency.

After these extensive testings in various situations and behaving correctly, it was time to execute a real flight experiment.

10 Experiments

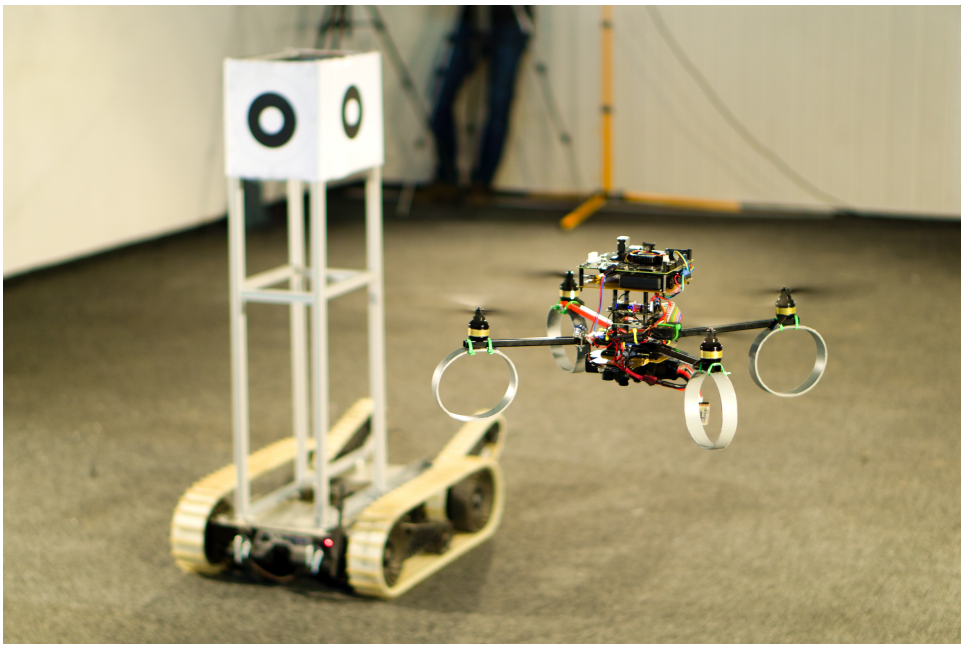


Figure 16: Photo of the obstacle avoidance experiment.

The implemented MPC controller has been verified by a real world experiments. Simulations, that have been done, differ from the real world experiments in the influences of the environment, system nonlinearity and

the sensory noise. Because the UAV has 4 propellers on sides, collisions are dangerous not only for the UAV, but for the operator as well. That is the reason why the operator has to have the access to take over the control of the UAV any time during these experiments. During the described experiments this possibility has not been used and the presented data are solely while the MPC was in control.

10.1 Stabilization

The first experiment was to test the stability and settle time of the MPC. The setpoint was set to the origin of the world coordinate system. The task for the UAV was to track this setpoint while the UAV was physically pushed by the operator. This tested reactions of the controller to an outside disturbances.

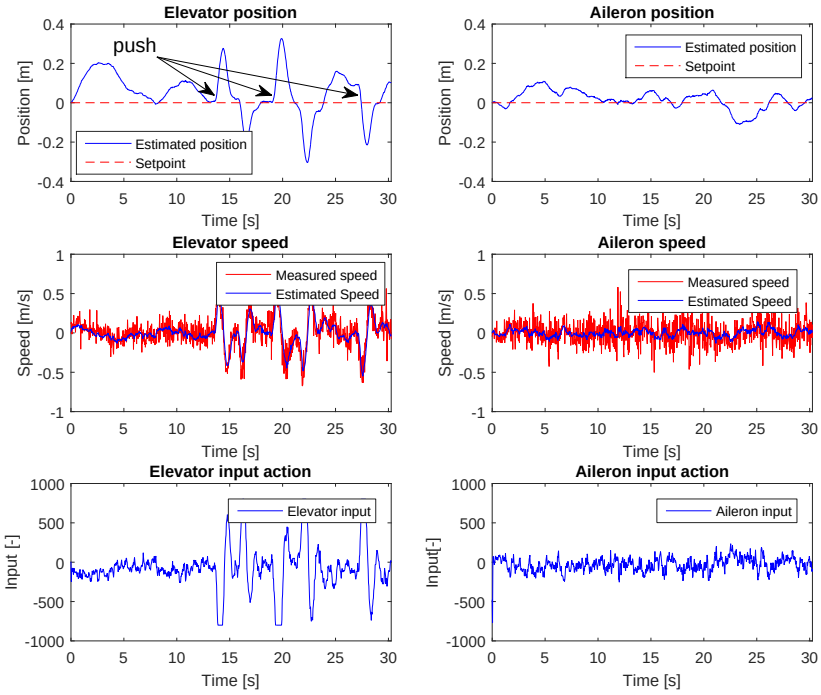


Figure 17: Data from the stabilization experiment

The Fig. 17 shows the position, speed and input action during the stabilization experiment. The UAV was pushed three times in the positive direction of the elevator axis. The UAV returned then to the original position. The aileron position stayed away from the setpoint within 11 cm, which is a good performance considering the size of the UAV's body. At the

elevator axis it is 20 cm if not considering the disturbances. The settle time was about 3 s.

10.2 Obstacle avoidance

The obstacle avoidance system has been also tested. The obstacle was represented by the blob. This was especially risky experiment, because the UAV had to get close to the obstacle. The maximum speed was set to 0.2 ms^{-1} for the operator to have enough time to take control of the UAV in case incorrect behavior. The initial arrangement is show in Fig. 18. The UAV was given a setpoint 4 m ahead on the elevator axis. The obstacle was in the middle between the UAV's initial position and the setpoint and did not move. The obstacle was given an extended predefined radius of 0.5 m. The trajectory was being created on-board, so this algorithm was tested as well. The task was to avoid the obstacle and arrive to the setpoint position.

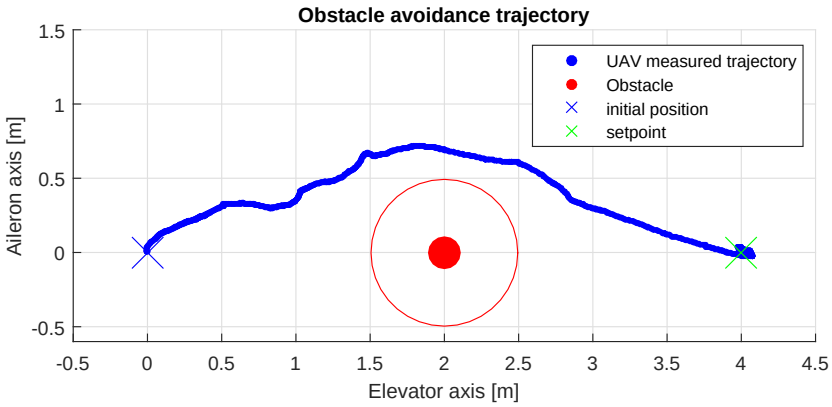


Figure 18: Obstacle avoidance experiment

The Fig. 18 shows the UAV's obstacle avoidance trajectory. Around the obstacle is a red circle of 0.5 m, which is the minimal distance allowed from the obstacle. The figure also shows, that there is a safe distance from the obstacle, which was enforced by the barrier function. The UAV also finishes on the setpoint and stabilizes there. The collision experiment was successful.

11 Conclusion

In this thesis a Model predictive controller for UAV was developed. It is capable of trajectory or setpoint tracking and obstacle avoidance. The MPC algorithm and linearly constrained quadratic programming solver has been designed and tested by matlab simulations. It was then implemented on

embedded hardware and tested again. Real flight experiments have been successfully conducted. The UAV was capable of trajectory tracking while detecting and avoiding an obstacle. The entire assignment of this thesis has been successfully fulfilled.

11.1 Future work

Although simulations and experiments have been successfully executed, there is still a room for future improvements. The greatest limitation of the introduced algorithm is keeping the convexity of the mathematical optimization. If this condition was lost, the optimization problem would become harder to solve, but the designed allowed prediction space would allow a better description of the surrounding area. This would allow the UAV to fly faster and safer.