POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in

Ingegneria Meccanica

Online Trajectory Generation with Obstacle Avoidance for Quadcopters using Model Predictive Control

Relatore: Prof. Francesco BRAGHIN

Tesi di laurea di:

Federico BORGIA

Matr. 769964

Anno Accademico 2013 - 2014

Contents

List of Figures

Abstract

The goal of this thesis is to design a model predictive control strategy for quadcopters, capable of generating trajectories in real-time in order to reach given targets and, at the same time, avoid fixed and moving obstacles. The trajectory is generated in terms of the jerk by solving a convex quadratic optimization problem. The obstacle avoidance task should be provided exploiting linear constraints, due to the great advantages that these entail in terms of computational load. First, a method that approximates the feasible space through convex polyhedra is implemented. After detecting several problems associated with this technique a different strategy is developed. The latter relies on the assumption of moving with constant longitudinal speed. Thus, the bounds on the positions can be imposed directly in the time domain. Furthermore, thanks to the holonomic nature of quadcopters, the optimization problem can be decoupled in the three axes. The lateral and vertical controls provide translations in order to avoid collisions, while the longitudinal control only decelerates if the solved trajectories do not avoid the obstacles properly. The two collision avoidance strategies are compared to each other. The method developed in this work shows great improvements concerning the reliability of the provided trajectories and the computational load needed to solve the optimization problems. Furthermore, the strategy is capable of managing higher speeds. The performance of the overall control is assessed through simulations done on a highly accurate model of the quadcopter. Real-time feasibility is also tested for low CPU frequencies, thus providing the requirements for a future implementation of the controller directly onboard the vehicle.

Sommario

Questa tesi si propone di sviluppare un controllo per quadcopter basato sul Model Predictive Control in grado di generare traiettorie in tempo reale in modo da raggiungere determinati obiettivi evitando ostacoli fissi e mobili. La traiettoria è generata in termini di jerk, ovvero la derivata prima dell'accelerazione, risolvendo un problema di ottimizzazione quadratica. L'aggiramento degli ostacoli deve essere svolto per mezzo di vincoli lineari imposti sulla posizione del veicolo, in quanto questi ultimi comportano notevoli vantaggi in termini di costo computazionale. Il primo metodo studiato consiste nell'approssimare lo spazio praticabile attraverso poliedri convessi. Vari problemi associati a questa tecnica spingono a sviluppare un approccio diverso. Tale nuova strategia si basa sull'ipotesi di percorrere la traiettoria a velocità longitudinale costante. In tal modo, potendo esprimere la coordinata longitudinale in funzione del tempo, è possibile imporre dei vincoli sulle coordinate laterali e verticali per ogni istante di tempo. Inoltre, grazie alla natura olonoma del quadcopter, il problema di ottimizzazione può essere scomposto nei tre assi, riducendo così di molto il costo computazionale. La funzione di superamento ostacoli è quindi svolta dai controlli sugli assi laterale e verticale, mentre il controllo sull'asse orizzontale garantisce una velocità costante riducendola solamente in caso le traiettorie generate non siano in grado di evitare correttamente gli ostacoli. Le due strategie sono infine confrontate tra loro. Il metodo sviluppato in questa tesi risulta portare notevoli miglioramenti in termini di affidabilità delle traiettorie e di costo computazionale. Il controllo è quindi implementato in un modello altamente dettagliato del quadcopter in modo da valutarne le performance. Sono state inoltre svolte simulazioni a varie velocità di clock del processore in modo da verificare la fattibilità in tempo reale.

1 Introduction

1.1 Motivation

In the last few years there has been much interest in the use of small unmanned aerial vehicles (UAVs) for security, surveillance, search and rescue as well as film recording applications. Research has focused especially on multirotor helicopters because of their great agility and ability to move in tough environments. Multirotors are highly maneuverable and are characterized by the ability to take-off and land vertically. A special class is the quadrotor configuration with fixed-pitch propellers. This category has been particularly investigated due to its very simple mechanical structure. Other advantages of quadcopters are the great load capacity as well as the low manufacturing costs. Nevertheless, the highly nonlinear and coupled dynamics along with complex aerodynamic effects set demanding challenges in the development of reliable controls. Furthermore, the need to give more and more autonomy to this sort of vehicles leads to challenging issues regarding the development of real-time trajectory planning strategies. Various approaches are proposed in the literature both for stabilization and trajectory tracking of quadcopters. In [1] a nonlinear control for geometric tracking of prescribed trajectories is described. Controls for aggressive manuevers within tight indoor environments, where the approximation of small angles of the frame can not be justified, are developed in [2] and [3]. Few works present model predictive control strategies for both path tracking and trajectory generation. Many of them use model predictive control as part of two-layer control structures ([4], [5], [6], [7]). In this case the double-layer control may consist of two distinct MPCs, such as in [6], [8] and [7], or may have an MPC for the high-level and a different control type for the lower level ([4], [5]). Model predictive control is well suited also for state interception maneuvers, thanks to the possibility to directly impose constraints on the motor thrusts as well as on positions to be reached ([9], [10]). One of the most challenging tasks that trajectory planning controls have to cope with, is the avoidance of collisions with obstacles that may occur on the way. For this aim, many attempts were made in order to develop a control capable of solving valid trajectories online, and thus with hard restrictions imposed on computational load due to real-time feasibility. In [12] a strategy is presented, that computes waypoints and according to them solves polynomial trajectories. Also MPC was already used for obstacle avoidance. [1] exploits a nonlinear model predictive tracking control (NMPTC) in order to avoid collisions with other vehicles. In [8] the obstacle avoidance task is instead carried out by a linear time varying model predictive control (LTV-MPC) that approximates the flyable area through linear contraints. As will be seen, linear contraints bring great benefits in terms of computational load, however the approximations associated to them lead to great difficulties regarding the speed at which the manuevers can be performed.

The aim of this work is to develop a model predictive control for both stabilization and trajectory planning including obstacle avoidance, capable of controlling a quadcopter at higher speeds, while satisfying the strict requirements for real-time feasibility.

1.2 Content and structure of the work

The remainder of this work is organized as follows. In chapter 2 the system dynamics of the quadcopter are described together with the proposed overall control structure. The last section of chapter 2 is dedicated to the dynamic model considered by model predictive control in order to make its prediction. This relies on the assumptions made in [9]. Moreover, feasibility due to physical limitations is discussed and accordingly constraints on the control actions are imposed.

Chapter 3 first presents the basics of model predictive control. Then, after choosing a special class of MPC, it describes the optimization problem that underlies the control strategy. After this, simple trajectories are generated for short and longer distances in order to intercept given target states.

Chapter 4 relies on the results achieved in the previous chapters and adds the collision avoidance feature. For this task the strategy described in [6] is first implemented.

Various problems associated with this technique are detected. This gives the cue to develop a completely different approach for obstacle avoidance. Furthermore, the new method is tested in three-dimensional environments as well as for moving obstacles. At the end of the chapter a comparison between the strategy described in [6] and the method developed in this work is made. Trajectory planning performance as well as computational load are compared and discussed.

In chapter 5 the overall control and trajectory planning strategy is implemented into a highly accurate simulation model of the quadcopter. After discussing the results, real-time feasibility trials are done in order to verify the applicability of the control structure.

2 Quadcopter Model

2.1 Overview

The quadcopter consists of a cross-shaped frame with four independent motors, each placed on a vertex of the frame. Each motor is directly attached to a fixed-pitch propeller and by controlling its rotational speed it can generate a force directed along the axis of the rotor. Through variation of the four rotational speeds one can generate a total thrust, directed along the vertical axis of the body-fixed frame, and three torques, giving the quadcopter the ability to move in space. The system has six degrees of freedom, three translations and three rotations of the frame, but only four indipendent inputs (propeller speeds), hence it is strongly underactuated. Only four outputs can be controlled, namely the position coordinates x_1, x_2, x_3 and the yaw angle. This leads the rotations about the horizontal axes being heavily coupled with the translations of the vehicle. Furthermore the quadcopter is a highly nonlinear instable system, which leads to the necessity of an electronic control, even for a mere hovering. In fact, unlike ground vehicles, it has very small friction, hence a damping action has to be carried out by the controller itself.

It is important to understand the functioning principle that underlies the quadcopter's behavior. Hence a simple description of how the quadcopter's rotational and translational movements are related to the control action provided by the four motors is given in this section. Each rotor produces a thrust and torque about its center of rotation, as well as a drag force opposite to the vehicle's direction of flight, which for our studies can be neglected. As shown in figure 2.1 the first and the third propeller rotate counterclockwise, and the second and the fourth propeller rotate clockwise. Therefore, if the four propellers all generate the same thrust, the moments will cancel out leading to a null total momentum about the vertical axis in the body-fixed frame.

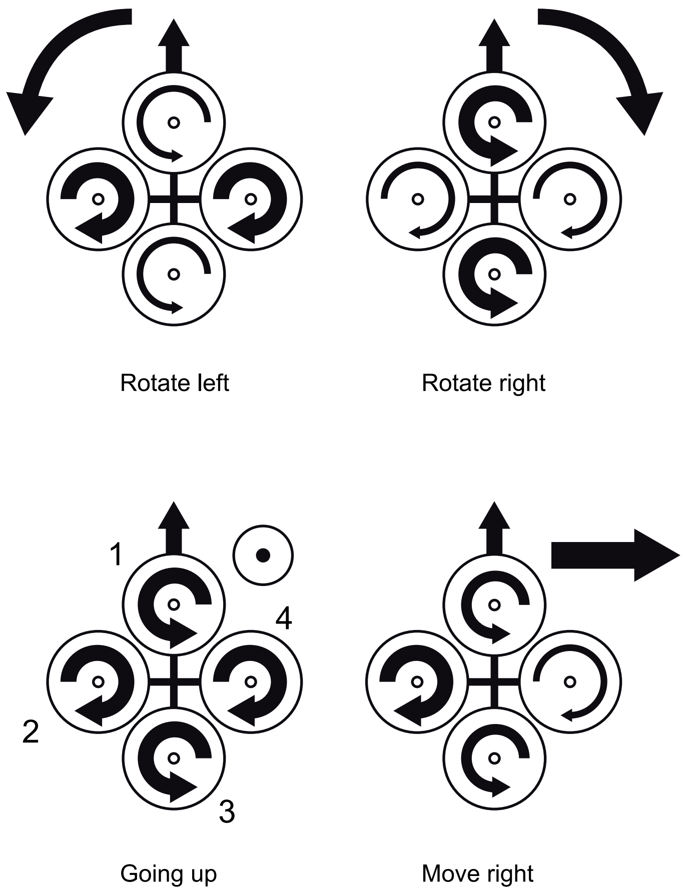

Figure 2.1: Rotors

A yaw-acceleration is induced by simply speeding up or down a pair of propellers rotating in the same direction. For example, speeding up the first and third propellers results in a clockwise rotation of the vehicle about its vertical axis. For simple hovering, the propellers will have to produce all the same power, so that the total thrust compensates the gravitational force. To move up, it is sufficient to speed up the propellers all about the same quantity, whereas to move the quadcopter down the propellers have to slow down. Lateral movements are achieved by unbalancing the rotational speed of two opposed motors. For example speeding up the second motor and slowing down the fourth will result in a torque about the body-fixed x_1-axis. This will lead to a rotation of the frame about the x_1-axis that will bring the direction of the total thrust, which is always directed normally to the quadcopters plane, to have a horizontal component that pulls the vehicle laterally. It is important to remark that rotations and translations are strongly coupled. Figure 2.1 gives a simple graphical explanation.

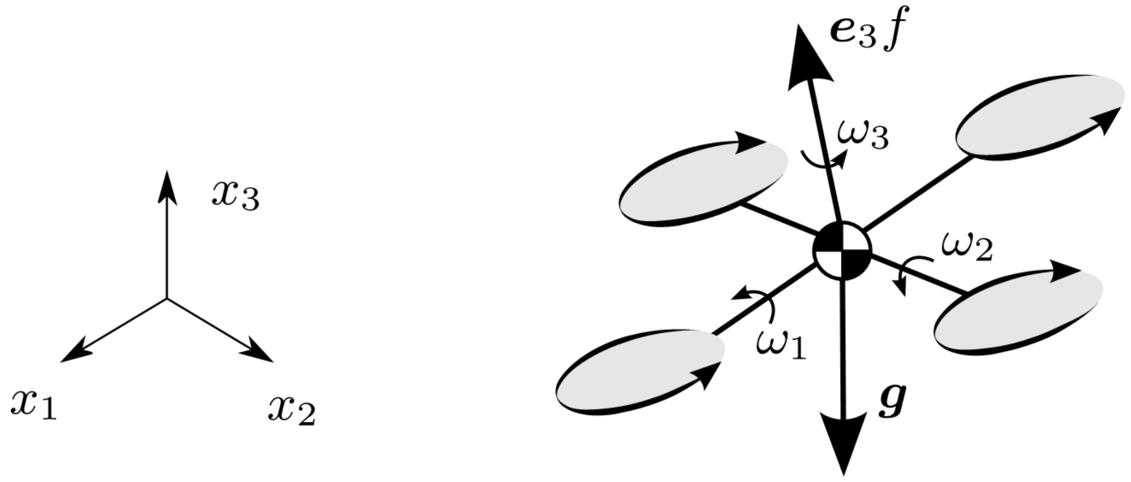

Figure 2.2: Quadcopter dynamic model - in [9]

2.2 Quadcopter dynamics

The quadcopter can be described as a rigid body with six degrees of freedom: three linear translations along the inertial axes x_1, x_2 and x_3 and three degrees of freedom describing the rotation of the frame attached to the body with respect to the inertial frame, described by the orthogonal matrix R. Thus the differential equations governing the flight can be written as:

$$\dot{x} = v \tag{2.1}$$

$$m\dot{v} = Re_3 f + mg \tag{2.2}$$

$$\dot{R} = R\hat{\Omega} \tag{2.3}$$

$$J\dot{\Omega} + \Omega \times J\Omega = M \tag{2.4}$$

with $e_3 = [0 \quad 0 \quad 1]^T$ and $\hat{\Omega}$ the skew-symmetric matrix form of the vector cross product such that

$$\hat{\Omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{2.5}$$

Note that (2.1) and (2.2) are written in the inertial frame, whereas (2.4) is expressed in the body-fixed frame. The inputs of the system are the total thrust f and the three components of the torque $M = [M_1 \, M_2 \, M_3]$

The motors dynamics are much faster than the body dynamics, hence we assume that the thrust of each propeller is directly controlled, i.e., dynamics of rotors and propellers are not considered ([1]). The direction of the thrust is normal to quadrotor plane. It is also assumed that the torque generated by each propeller is directly proportional to its thrust.

$$\tau_i = (-1)^i c_{\tau f} f_i \tag{2.6}$$

where i is the motor index beginning from the front propeller and counting counterclockwise. $c_{\tau f}$ is the constant that relates the torque to the thrust. Once the needed total thrust f and the moments M_1, M_2, M_3 have been evaluated, the forces f_i that have to be provided by each single motor can be computed by simply inverting the following equation.

$$\begin{bmatrix} f \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & d & 0 & -d \\ -d & 0 & d & 0 \\ -c_{\tau f} & c_{\tau f} & -c_{\tau f} & c_{\tau f} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \tag{2.7}$$

where d is the distance of the motor with respect to the center of gravity of the quadcopter. Generally the propellers are equally spaced for which reason it is considered that their distances to the center are all given by d. The force produced by each propeller is related to its rotational speed as follows:

$$f_i = k_f \omega_i^2 \tag{2.8}$$

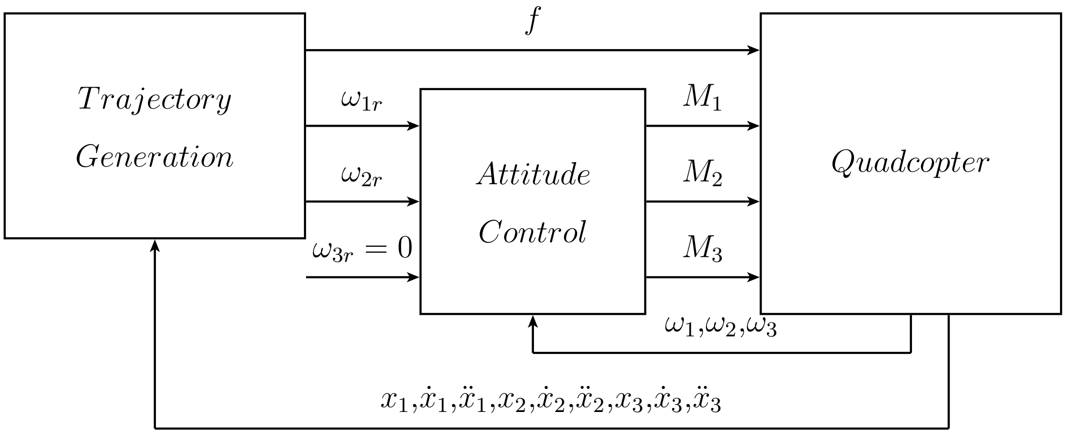
f
Trajectory Generation
ω_{1r}
ω_{2r}
$\omega_{3r} = 0$
Attitude Control
M_1
M_2
M_3
Quadcopter
ω_1,ω_2,ω_3
x_1,$\dot{x}_1$,$\ddot{x}_1$,x_2,$\dot{x}_2$,$\ddot{x}_2$,x_3,$\dot{x}_3$,$\ddot{x}_3$

Figure 2.3: Control Structure

This means, once the thrust is known, the angular velocity the motor has to provide can be easily computed. Every motor is then controlled by its own speed-controller. However, as mentioned before, the motor dynamics won't be considered in the system model as they are much faster then the quadcopter dynamics and can be therefore neglected.

2.3 Control Structure

To govern the quadcopter's flight through a model predictive strategy the same control scheme and control inputs as described in [9] are used, since this greatly simplifies the trajectory generation task such that the model predictive controller only has to deal with linear systems. The quadcopter has very low rotational inertia, and can produce high torques due to the outward mounting of the rotors. Therefore it can achieve very high rotational accelerations $\dot{\omega}_1$ and $\dot{\omega}_2$ on the order of 200 rad/s^2. The reponse time to changes in the desired rotational rate is very fast. It is therefore assumable that the vehicle body rates can be directly controlled, thus the rotational acceleration dynamics can be ignored by the model predictive controller. Hence, the control inputs for the MPC are chosen to be the generated total thrust f, given by the sum of the four propeller thrusts f_i, and the body rates expressed in the body-fixed frame as $\omega = [\omega_1 \, \omega_2 \, \omega_3]^T$. Thus the MPC will generate reference values for the total thrust and the rotational rates in order to bring the quadcopter to follow the desired trajectory. Then the two body rates will be tracked by an attitude control, which uses feedback from gyroscopes and returns values of the needed torques about the body-fixed axes. Finally, total thrust and torques will be mixed to individual motor thrusts exploting equation (2.7). The overall structure of the control strategy is depicted in figure 2.3.

2.4 Dynamic model for the model predictive controller

2.4.1 Control Inputs

As mentioned in section 2.3 the inputs can be chosen as the total thrust f and the body rates described in the body-fixed frame $\omega = [\omega_1\, \omega_2\, \omega_3]^T$, since the rotational accelerations are fast enough to decouple them from the body dynamics. For simplicity from now on the total thrust f is normalized by the vehicle mass, thus having units of acceleration. The differential equations can be rewritten leaving out (2.4).

$$\ddot{x} = Re_3 f + g \tag{2.9}$$

$$\dot{R} = R\hat{\Omega} \tag{2.10}$$

with $\hat{\Omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$

2.4.2 Reformulation in jerk

The aim is to consider the trajectories of the quadcopter in terms of the jerk (the first derivative of the acceleration) of the axes, allowing the system to be considered as a triple integrator in each axis and thus simplifying the trajectory generation task. Writing the chosen inputs of the system, namely the total thrust and the three rotational rates, as a function of the jerk, allows to compute them easily given a thrice differentiable trajectory $x(t)$, where the jerk is written as $j = \dddot{x} = [\dddot{x}_1\, \dddot{x}_2\, \dddot{x}_3]^T$.

The input thrust f is found by applying the Euclidean norm $\|\cdot\|$ to (2.9),

$$f = \|\ddot{x} - g\| \tag{2.11}$$

By squaring (2.11), taking the derivative and substituting for (2.9) the following equation is obtained:

$$2f\dot{f} = 2(\ddot{x} - g)^T j = 2(Re_3 f)^T j \tag{2.12}$$

$$\dot{f} = e_3^T R^T j \tag{2.13}$$

Derivating (2.9) yields

$$j = R\hat{\Omega}e_3 f + Re_3\dot{f} \tag{2.14}$$

Substituting $\dot{f}$ with (2.13) and evaluating the product $R\hat{\Omega}e_3$ one can finally write:

$$\begin{bmatrix} \omega_2 \\ -\omega_1 \\ 0 \end{bmatrix} = \frac{1}{f}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} R^T j \tag{2.15}$$

As shown in (2.15) the jerk j and thrust f fix two components of the body rates. Indeed the third component is not necessary, as a rotation about the e_3 axis does not affect the translational acceleration (2.9). That means that ω_3 can be chosen freely, for example one can control ω_3 such that the x_1-axis of the body-fixed frame remains oriented with the direction of movement. For simplicity it will be chosen that $\omega_3 = 0$. Given a trajectory described by the three components of the jerk and the initial states x_0, $\dot{x}_0$ and $\ddot{x}_0$, also the trajectories for acceleration, speed and position can be computed. Thus, one can evaluate the inputs of the system using (2.11) and (2.15).

2.4.3 Feasibility constraints

A trajectory is considered to be feasible if the inputs lie within a feasible range. Necessarily there are some boundaries on the total thrust and the magnitude of the body rates that result from physical limitations and therefore have to be considered when calculating a trajectory. As previously stated one can consider a trajectory described by the jerk and from that it is possible to evaluate the needed total thrust and body rates using equations (2.11) and (2.15). As can be seen the total thrust depends on the acceleration, which obviously can be calculated by simply integrating the jerk trajectory. The aim is to find boundary conditions on acceleration and jerk in order to satisfy the limitations on total thrust and body rates, that will depend on physical bounds and can be obtained from experimental results of the considered vehicle. Note, that the constraints must be convex and linear due to the requirements of the quadratic programming method exploited by the model predictive control and described in chapter 3. This restrictions on the class of boundary conditions require to write conservative constraints that in some cases will not exploit the whole potentiality of the system.

First, the boundary conditions on the inputs of the system can be expressed as:

$$0 < \; f_{min} \leq f \leq f_{max} \tag{2.16}$$

$$\|\omega\| \leq \omega_{max} \tag{2.17}$$

where $f_{min} > \; 0$ is typical for fixed-pitch propellers with a fixed direction of rotation. In fact, the propellers can't reverse their rotation direction during flight and can not rotate under a certain rotational rate. Thus the minimum achievable total thrust corresponds to the case in which all the motors are rotating at their minimum speed. Obviously the maximum total thrust can be achieved when all motors are rotating at their maximum speed. By squaring (2.11) and writing it in its component these limits can be translated to limits on the acceleration, and thus on the jerk trajectory.

$$f_{min}^2 \leq \ddot{x}_1^2 + \ddot{x}_2^2 + (\ddot{x}_3 + g)^2 \leq f_{max}^2 \tag{2.18}$$

Note that these constraints on the acceleration are neither convex nor linear. Hence it is necessary to find a convex area described by linear constraints. The following inequalities give a conservative set of convex box-constraints, one for each axis.

$$\ddot{x}_{min1} = -\ddot{x}_{max1} \; \leq \ddot{x}_1 \leq \ddot{x}_{max1} \tag{2.19}$$

$$\ddot{x}_{min2} = -\ddot{x}_{max2} \; \leq \ddot{x}_2 \leq \ddot{x}_{max2} \tag{2.20}$$

$$\ddot{x}_{min3} = f_{min} - g \leq \ddot{x}_3 \leq \ddot{x}_{max3} \tag{2.21}$$

As will be explained, in chapter 3 writing constraints for each axis separately allows to

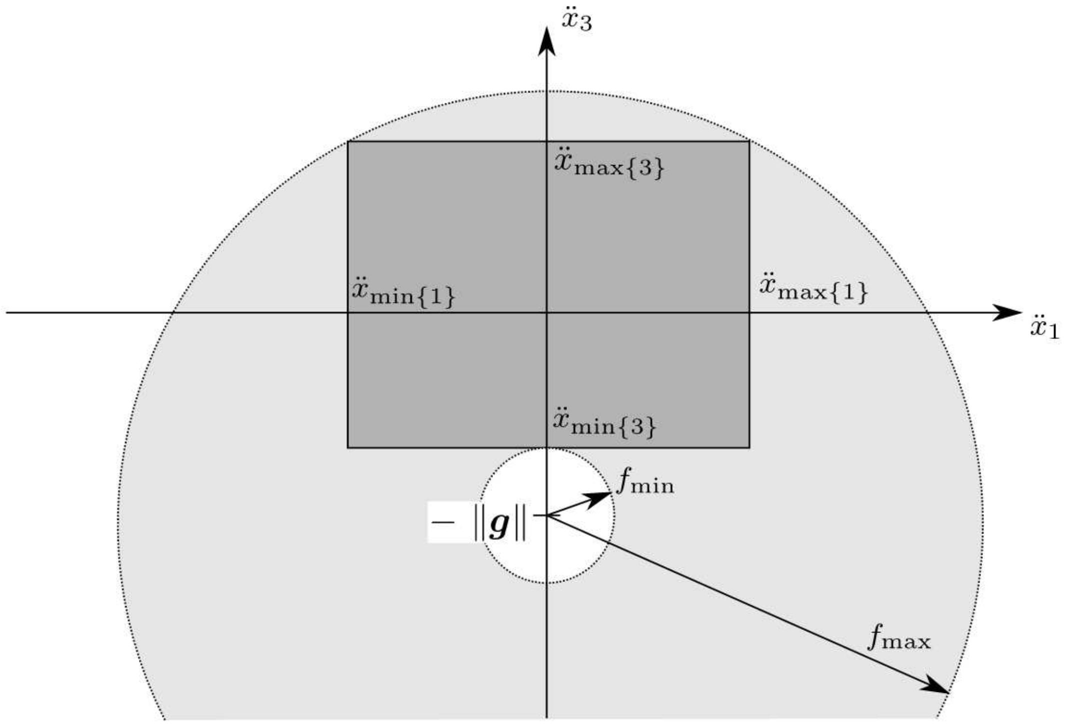

Figure 2.4: Cross-section of the feasible acceleration sets - [9]

decouple the axes, which will greatly simplify the trajectory generation task. Figure 2.4 depicts a cross-section in the x_1-x_3 plane of the true thrust limits (lightly shaded) and the decoupled per-axis acceleration limits (darker rectangular area). As shown in the graph, the box-constraints leave out a big part of the true feasible area in order to be convex and linear. As mentioned before the performance of the quadcopter can't be exploited to its full extent and conservative boundary conditions have to be considered. Note that the direction of a generic thrust vector drawn in figure 2.4 also represents the real direction of the quadcopter's body fixed x_3-axis, since it is always directed with the total thrust. This means that none of the feasible three-dimensional accelerations considered brings the quadcopter to overturn. Hence this will also limit the angle by which the vehicle's frame can be rotated, with a maximum value owned for the points corresponding to the lower vertices of the rectangular area. The resulting trajectories are guaranteed to be feasible with respect to the thrust limit if

$$\ddot{x}_{max1}^2 + \ddot{x}_{max2}^2 + (\ddot{x}_{max3} + g)^2 \leq f_{max}^2 \tag{2.22}$$

which brings the box-constraints to lie inside the true feasible region. An upper bound for the body rates can be found as a function of the jerk by taking the (induced) norm of (2.15)

$$\|\omega\| \leq \frac{1}{f}\|j\| \leq \frac{1}{f_{min}}\|j\| \tag{2.23}$$

Applying the limit (2.17) to the above and rearranging the terms yields:

$$\|j\| = \sqrt{j_1^2 + j_2^2 + j_3^2} \leq f_{min}\omega_{max} \tag{2.24}$$

To obtain linear convex decoupled constraints on the jerk, the worst case in which all three axes produce the maximum allowable jerk j_{max} is evaluated. In this case the boundary conditions expressed in (2.24) still have to be satisfied. This yields an upper bound on the allowable jerk per axis

$$j_{max} = \frac{1}{\sqrt{3}} f_{min}\omega_{max} \tag{2.25}$$

As said before, ω_{max}, f_{min} and f_{max} come from physical limits and can be obtained by experimental results. Furthermore, by exploting equations (2.25), (2.22) and (2.19)-(2.21), convex limits on jerk and acceleration are found and can be applied directly into the optimization problem mentioned in chapter 3 due to their linear nature.

3 Model predictive online trajectory generation

In this chapter a method for generating online trajectories exploting the potentiality of model predictive control (MPC) is described. As mentioned in chapter 2 MPC will perform the trajectory generation task and will act as a high-level control. The output will be a trajectory described in terms of jerk, out of which the total thrust and two body rates can be easily computed using equations (2.11) and (2.15). Then the body rates will be tracked by a low-level control which exploits feedback from gyroscopes. First, simple trajectories like reaching a target point starting from resting conditions or approaching distant positions are performed. The results achieved in this chapter will be then used in chapter 4 in order to guide the quadcopter through more complex situations like flying among static and moving obstacles.

The remainder of this chapter is organized as follows. After explaining the main principle of model predictive control (section 3.1), a particular class of MPC will be chosen in order to satisfy the strong requirements set on the computational load for real-time applications (section 3.2). The basics of linear discrete-time MPC and its implementation will be described. In section 3.4 first trials for simple trajectory generation tasks along with the choice of the parameters will be illustrated and discussed.

3.1 Model predictive control overview

Model predictive control was introduced in the early 80's. First, it was developed to control chemical industrial plants as well as refineries. The basic principle of MPC is to control a system by making predictions on its future behavior and therefore being able to optimize a certain cost function along the predicted horizon. The main advantage of this control method is that it is able to calculate the optimal control inputs taking into account the system dynamics and considering the physical limitations that may be imposed on certain variables. Considering, for example, a tank whose pressure is controlled by a valve, MPC makes it possible to calculate all the future positions of the valve needed for the pressure to follow a given reference. The evaluated trajectory for the inputs can easily handle all the constraints on the manipulated variable itself as well as the constraints on the output (or generally on the states of the plant). For example it can be considered, that the valve is only allowed to close with a certain speed or that the value of the pressure should remain in a given range. As said before the MPC generates a trajectory for the control variable minimizing a given functional. Typically this objective function is the error of the output variables with respect to a given reference as well as terms considering the overall energy needed to follow the trajectory. Once the control actions for the future horizon are computed by an optimization solver, only the first step of the input trajectory is applied to the system. Then the process is repeated and a new trajectory is evaluated starting from the new actual state of the plant. This procedure is called "receding strategy" and allows to cope with problems associated to modeling errors or unpredictable forces acting on the system, therefore allowing a proper feedback action. In fact, if all the system parameters as well as the future behavior of all the external agents were perfectly known, it would be sufficient to calculate an optimal trajectory of the control actions once and for all. MPC is a very powerful tool, but on the other hand it has to cope with two main challanges. The first is the need for an accurate knowledge of the system model, which in most cases is governed by a significant number of nonlinear differential equations. It is also difficult to obtain a precise estimation of all the parameters describing the model. The second drawback is the huge computational load needed by the optimization solver to generate a solution. This is also the reason why MPC was initially applied only to plants governed by slow dynamics, where the update rate of the control action can be low enough to solve an optimization problem. Due to the progress achieved in the last decades on the computational performances of the processors as well as on efficient optimization algorithms, it is today possible to control systems with fast dynamics with model predictive techniques. MPC has therefore become an interesting object of research for what concerns real-time generation of optimal trajectories for autonomous vehicles such as road vehicles or UAVs.

3.2 Linear discrete-time model predictive control

As explained in section 3.1 one of the main drawbacks of MPC is the time needed to solve an optimization problem. A useful solution to keep a low computational load is to use a linear discrete-time MPC. The latter is characterized by minimizing a quadratic objective function subject to linear constraints and using a linearized model of the system. Linear equations are much easier to cope with and there is a great multitude of fast algorithms that can solve quadratic optimization problems with linear constraints. Moreover, by discretizing the system in the time domain it is possible to transform the dynamic optimization problem, which needs to be solved in an analytical way, into a static optimization problem, which can be easily managed numerically with fast algorithms.

After evaluating the future control actions, only the first step is applied to the system. This control input is held until the next one is calculated, which means that the control input is step-shaped. For that reason a discrete-time model of the system is perfectly suited to the step-working control logic of MPC.

3.2.1 Quadratic Problem Formulation

Most of the software for quadratic programming accept the following formulation of the problem:

$$\min_{x} \quad J = \frac{1}{2}x^T H x + x^T g \tag{3.1}$$

$$s.t. \quad lb_A \leq \; A_{ineq} \cdot x \leq \; ub_A \tag{3.2}$$

$$lb \leq \; x \leq \; ub \tag{3.3}$$

where x is the vector of the manipulated variables, H is the symmetric and positive (semi-)definite Hessian matrix, which represents the quadratic term of the objective function, g is the gradient vector, representing the linear term. A_{ineq} is the matrix, which multiplied by x returns the affine functions for the linear constraints. lb and ub are respectively the lower and upper bounds that act directly on the variables, whereas lb_A and ub_A are the lower and upper bounds for the linear constraints.

3.2.2 Time-discretization of the system

The aim is to generate a trajectory in order to minimize a specific quadratic cost function along the considered time window. Typically this cost function consists of a term that depends on the states of the system and one that depends on the control input. Given the linear continuous-time system

$$\dot{x}(t) = A^* x(t) + B^* u(t) \tag{3.4}$$

with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$, the quadratic objective function is

$$J = \frac{1}{2} \int_0^{T_H} x^T Q x + u^T R u \, dt \tag{3.5}$$

where Q is the symmetric positive (semi-)definite matrix containing the weighting terms for the states and R is the symmetric positive definite matrix that weights the control inputs. The dynamic optimization problem of minimizing (3.5) subject to the dynamics (3.4) has to be transformed into a static optimization problem of the form (3.1) - (3.3). This can be done by exploiting the following equations:

$$A_d = e^{A^* \cdot t_s} \tag{3.6}$$

$$B_d = \left(\int_{\tau=0}^{t_s} e^{A^* \cdot \tau} \, d\tau \right) B^* \tag{3.7}$$

where A_d and B_d are the system matrices of the discrete-time system given by

$$x_{k+1} = A_d x_k + B_d u_k \tag{3.8}$$

For convenience A_d and B_d will be renamed as A and B. The formulation of the quadratic optimization problem in discrete-time form is:

$$\min_{u_0 \cdots u_{N-1}} \quad \frac{1}{2} \sum_{k=0}^{n-1} x_{k+1}^T Q x_{k+1} + u_k^T R u_k \tag{3.9}$$

$$s.t. \quad x_{k+1} = Ax_k + Bu_k \tag{3.10}$$

$$lb_{x_k} \leq C_{ineq_k} x_k \leq ub_{x_k} \qquad k = 1 \cdots N \tag{3.11}$$

$$lb_{u_k} \leq u_k \leq ub_{u_k} \qquad k = 0 \cdots N-1 \tag{3.12}$$

where N represents the number of steps composing the prediction horizon and C_{ineq_k} represents the matrix needed to impose linear contraints on the states for the step k. Writing (3.10) for each step of the horizon and substituting each equation in the following one yields:

$$\begin{aligned} x_1 &= Ax_0 + Bu_0 \\ x_2 &= Ax_1 + Bu_1 && = A^2x_0 + ABu_0 + Bu_1 \\ &\vdots \\ x_N &= Ax_{N-1} + Bu_{N-1} && = A^N x_0 + A^{N-1}Bu_0 + A^{N-2}Bu_1 + \cdots + Bu_{N-1} \end{aligned} \tag{3.13}$$

Equation (3.13) can be summarized in matrix form

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}}_{\underline{X}} = \underbrace{\begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}}_{\underline{A}} x_0 + \underbrace{\begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ A^2B & AB & B & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ A^{N-1}B & & \cdots & & B \end{bmatrix}}_{\underline{B}} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{\underline{U}} \tag{3.14}$$

$\underline{X}$ is the vector containing the states of all N steps of the forecasted horizon, whereas $\underline{U}$ contains the $N \cdot m$ control inputs within the prediction time window. Finally it can be written:

$$\underline{X} = \underline{A}x_0 + \underline{B}\,\underline{U} \tag{3.15}$$

which summarizes the response of the discrete-time system given the $N \cdot m$ control inputs of the predicted horizon. The quadratic problem can be rewritten as:

$$\min_{\underline{U}} \quad \frac{1}{2}(\underline{X}^T \underline{Q}\,\underline{X} + \underline{U}^T \underline{R}\,\underline{U}) \tag{3.16}$$

$$s.t. \quad LB_X \leq \underline{C}_{ineq} \underline{X} \leq UB_X \tag{3.17}$$

$$LB_U \leq \underline{U} \leq UB_U \tag{3.18}$$

with

$$LB_X = \begin{bmatrix} lb_{x_1} \\ lb_{x_2} \\ \vdots \\ lb_{x_N} \end{bmatrix}, UB_X = \begin{bmatrix} ub_{x_1} \\ ub_{x_2} \\ \vdots \\ ub_{x_N} \end{bmatrix}, LB_U = \begin{bmatrix} lb_{u_0} \\ lb_{u_1} \\ \vdots \\ lb_{u_{N-1}} \end{bmatrix}, UB_U = \begin{bmatrix} ub_{u_0} \\ ub_{u_1} \\ \vdots \\ ub_{u_{N-1}} \end{bmatrix}$$

$$\underline{Q} = \begin{bmatrix} Q & 0 & \cdots & 0 \\ 0 & Q & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & Q \end{bmatrix}, \underline{R} = \begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & R \end{bmatrix} \text{ and } \underline{C}_{ineq} = \begin{bmatrix} C_{ineq_1} & 0 & \cdots & 0 \\ 0 & C_{ineq_2} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & C_{ineq_N} \end{bmatrix}$$

The aim is now to obtain an objective function that depends only on the manipulated variables of the N steps of the prediction horizon. To do this, (3.15) is substituted in the cost function described in (3.16), which leads to:

$$J = \frac{1}{2}\left[\left(\underline{A}x_0 + \underline{B}\,\underline{U}\right)^T \underline{Q}\left(\underline{A}x_0 + \underline{B}\,\underline{U}\right) + \underline{U}^T \underline{R}\,\underline{U}\right] \tag{3.19}$$

Rearranging the terms yields:

$$J = \frac{1}{2}\underline{U}^T\left(\underline{B}^T \underline{Q}\,\underline{B} + \underline{R}\right)\underline{U} + \underline{U}^T \underline{B}^T \underline{Q}\,\underline{A}x_0 + x_0^T \underline{A}^T \underline{Q}\,\underline{A}x_0 \tag{3.20}$$

The term $x_0^T \underline{A}^T \underline{Q}\,\underline{A}x_0$ is a constant term, which means it does not influence the minimization process. Therefore it can be ignored, so that the new cost function can be defined as:

$$J^* = \frac{1}{2}\underline{U}^T \underbrace{\left(\underline{B}^T \underline{Q}\,\underline{B} + \underline{R}\right)}_{H}\underline{U} + \underline{U}^T \underbrace{\underline{B}^T \underline{Q}\,\underline{A}x_0}_{g} \tag{3.21}$$

By substituting (3.15) in (3.17) and rearranging the terms the linear constraints on $\underline{X}$ can be rewritten as constraints on $\underline{U}$:

$$\underbrace{LB_X - \underline{C}_{ineq}\underline{A}x_0}_{lb_A} \leq \underbrace{\underline{C}_{ineq}\underline{B}}_{A_{ineq}}\underline{U} \leq \underbrace{UB_X - \underline{C}_{ineq}\underline{A}x_0}_{ub_A} \tag{3.22}$$

Finally, the optimization problem can be expressed as a function of the control action $\underline{U}$, thus with the formulation needed by the solver.

$$\min_x \frac{1}{2}\underline{U}^T H \underline{U} + \underline{U}^T g \tag{3.23}$$

$$s.t. \quad lb_A \leq \; A_{ineq} \cdot \underline{U} \leq \; ub_A \tag{3.24}$$

$$lb \leq \; \underline{U} \leq \; ub \tag{3.25}$$

with

$$H = \underline{B}^T \underline{Q}\, \underline{B} + \underline{R} \tag{3.26}$$

$$g = \underline{B}^T \underline{Q}\, \underline{A} x_0 \tag{3.27}$$

$$lb_A = LB_X - \underline{C}_{ineq} \underline{A} x_0 \tag{3.28}$$

$$ub_A = UB_X - \underline{C}_{ineq} \underline{A} x_0 \tag{3.29}$$

$$A_{ineq} = \underline{C}_{ineq} \underline{B} \tag{3.30}$$

3.2.3 Time parameters

Mainly two time parameters influence the applicabilty of MPC. The first is the sampling time t_S; that is the update rate whereby the controller generates a new control action. For real-time capability all computation has to be performed within the sample time. This value has to be set low enough to give the controller the capability to capture the fast dynamics of the plant. If the discretization is chosen too long, there is a risk to filter out important dynamics leading to stability problems.

The second fundamental component is the prediction horizon N. This sets the time window in which the optimization is performed, giving the length of the trajectory that has to be calculated. The horizon is calculated by $T_H = N \cdot t_S$. It has to be chosen long enough to consider the long dynamics of the plant. For example, if the horizon is set to $N = 50$ and the sampling time to $t_S = 30ms$, the time window has a length of $T_H = 1.5s$. Hence, the prediction horizon defines the number of variables that the optimization solver has to manage. It is easy to understand that the longer the prediction horizon is, the longer the algorithm will take to solve the problem and calculate the optimal trajectory of the manipulated variable. On the other hand it has to be kept as long as possible in order to extend the time window of the optimization.

3.3 Optimization problem and decoupled axes

In this section the optimization problem that underlies the trajectory generation task is depicted. As described in subsection 2.4.2 the system input can be considered to be the three-dimensional jerk, since the values of the thrust and two body rates can be easily computed as functions of it. Hence the quadcopter dynamics considered by the MPC become a set of three triple integrators, one on each axis, with position, velocity and acceleration as states. As suggested in [9] the cost function is chosen as:

$$J_{coupled} = \int_0^T \left(j_1(t)^2 + j_2(t)^2 + j_3(t)^2\right) dt. \tag{3.31}$$

Indeed, rearranging (2.23) the cost function results in an upper bound for a product of the inputs:

$$f^2 \|\omega\|^2 \leq j_1^2 + j_2^2 + j_3^2 \tag{3.32}$$

This implies that the problem can be split, thus minimizing the jerk separately for each axis without losing the meaning in the context of the coupled three-dimensional problem. The motivation for decoupling the axis is to have the simplest possible model in order to compute a solution for the optimization problem as fast as possible. Indeed it can be proved that solving three separate problems of N variables implies a much lower computational load than solving a single problem of $3N$ variables.

Note that the axes can be fully decoupled due to the holonomic nature of the quadcopter regarding its three translational degrees of freedom. In other words, the longitudinal, lateral and vertical movements can be controlled separately. To understand this consider the counterexample of a nonholonomic (or anholonomic) vehicle such as a car. In order to move left or right it has necessarily to move longitudinally. The most famous example is the parallel parking problem, where in order to get into the parking space the car first has to move forward, then it has to reverse its direction while steering. A quadcopter, instead, can directly move laterally. Thus, a point in space can be reached independently from the followed path. For that reason, the quadcopter will be controlled by three separate MPCs, each controlling a different axis separately.

Thus, three decoupled trajectory generation problems can be solved. For each axis a discrete-time linear, time invariant system given by a triple integrator, can be written as:

$$z_{k+1} = \begin{bmatrix} 1 & t_s & \frac{1}{2}t_s^2 \\ 0 & 1 & t_s \\ 0 & 0 & 1 \end{bmatrix} z_k + \begin{bmatrix} \frac{1}{6}t_s^3 \\ \frac{1}{2}t_s^2 \\ t_s \end{bmatrix} j_k \tag{3.33}$$

$$j_k = \dddot{x}(kt_s) \tag{3.34}$$

$$z_k = \begin{bmatrix} x(kt_s) \\ \dot{x}(kt_s) \\ \ddot{x}(kt_s) \end{bmatrix} \tag{3.35}$$

with jerk $j = \dddot{x}$ as input and position, velocity and acceleration as states. These are summarized in the state vector z_k. The axis subscripts have been neglected for convenience, since the formulation is exactly the same for all three axes. The discretization step t_s is chosen as usual equal to the sampling time of the MPC, since it represents also the real update rate of the control input. Now, that the system has been described, the cost function has to be defined. This is done by simply discretizing (3.31) and taking only the term corresponding to the evaluated axis.

$$J = \sum_{k=0}^{N-1} j_k^2 \tag{3.36}$$

where N is the number of steps composing the considered prediction horizon. The optimal control problem has to satisfy boundary conditions associated to acceleration and jerk limits as defined in subsection 2.4.3.

$$\ddot{x}_{min} \leq \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} z_k \leq \ddot{x}_{max} \tag{3.37}$$

$$j_{min} \leq j_k \leq j_{max} \tag{3.38}$$

The optimization problem for trajectory generation is defined by minimizing separately for each axis the cost function (3.36) subject to the system dynamics (3.33)-(3.35) and to the constraints on acceleration (3.37) and jerk (3.38). In order to perform tasks like intercepting a state or following a reference, some constraints and cost function terms still need to be added, as will be described in the next sections.

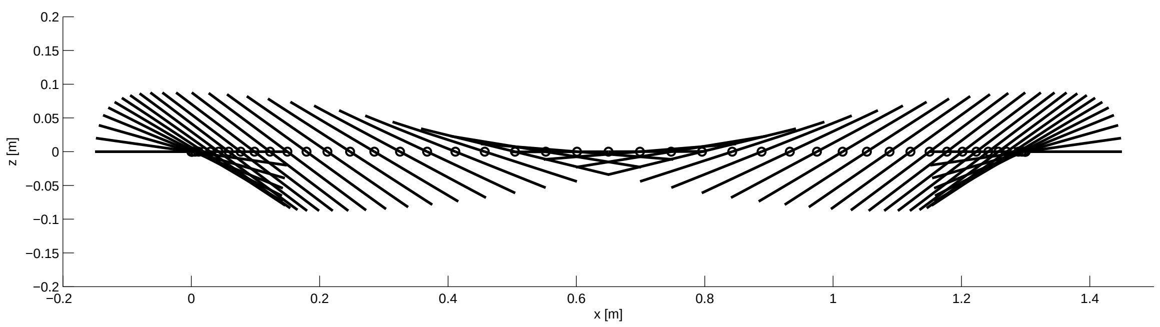

Figure 3.1: 1 meter rest to rest translation within 1 second - Trajectory of the frame in x_1-x_3 plane

3.4 Trajectory generation for state interception

3.4.1 Rest position to rest position

First, the case in which the quadcopter has to fly from a point to another in a time that corresponds to the prediction horizon T_H, is considered. In this case the time window considered by the MPC covers the whole maneuver. Generally this is not the case, because MPC takes into account only a time window limited to its prediction horizon, although the whole maneuver may last for a longer period. To force the trajectory to reach the desired target, a constraint on the last state vector has to be imposed.

$$z_N = [x_T \quad \dot{x}_T \quad \ddot{x}_T]^T \tag{3.39}$$

Because the optimization solvers usually only deal with inequality constraints, it is necessary to set the same values for the upper and lower bound in order to make an equality constraint out of two inequalities.

$$[x_T \quad \dot{x}_T \quad \ddot{x}_T]^T \leq z_N \leq [x_T \quad \dot{x}_T \quad \ddot{x}_T]^T \tag{3.40}$$

where $z_T = [x_T \quad \dot{x}_T \quad \ddot{x}_T]^T$ is the desired target state that has to be reached.

The first experiment considered is the displacement of 1.3m about the x_1-axis done in a time lapse of 1s from resting condition at start to resting condition at target. It should start and arrive with null speed and null acceleration. The sampling time is chosen to be $t_s = 20ms$. This means that the prediction horizon can be computed as $N = \frac{T_H}{t_s} = 50$. Having defined the whole optimization problem consisting of the system

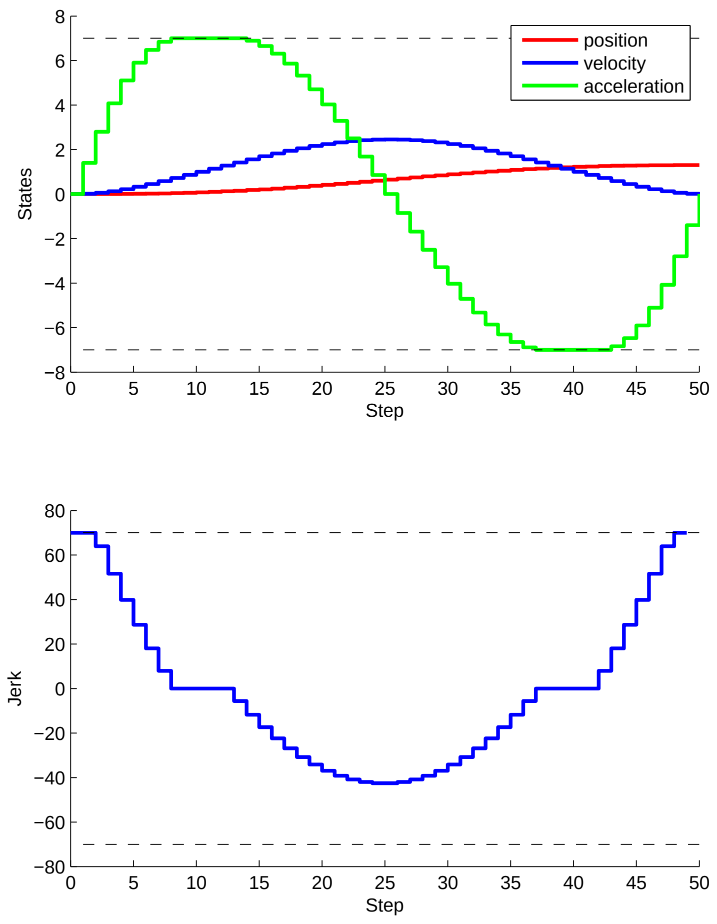

Figure 3.2: 1.3 meters rest to rest translation within 1 second - Generated trajectories for States and Jerk

dynamics (3.33) - (3.35), the cost function to minimize (3.36) and the constraints on input and states (3.37), (3.38) and (3.40) and imposing $z_T = [1.3m \quad 0 \quad 0]^T$, it can be implemented in the optimization solver. The acceleration limits are set to $\ddot{x}_{max} = -\ddot{x}_{min} = 7\frac{m}{s^2}$ and the jerk limits to $j_{max} = -j_{min} = 70\frac{m}{s^3}$. As optimization solver qpOASES [14] is chosen, which employs efficient online active set methods, generating solutions in very short times and therefore being very useful for model predictive control applications. In this section it will be discussed about the solutions that the solver computes for the first optimization, namely the one for $t = 0$. Hence the receding strategy acting online, will not be implemented yet.

In figure 3.1 the movement of the quadcopter in the two-dimensional plane x_1-

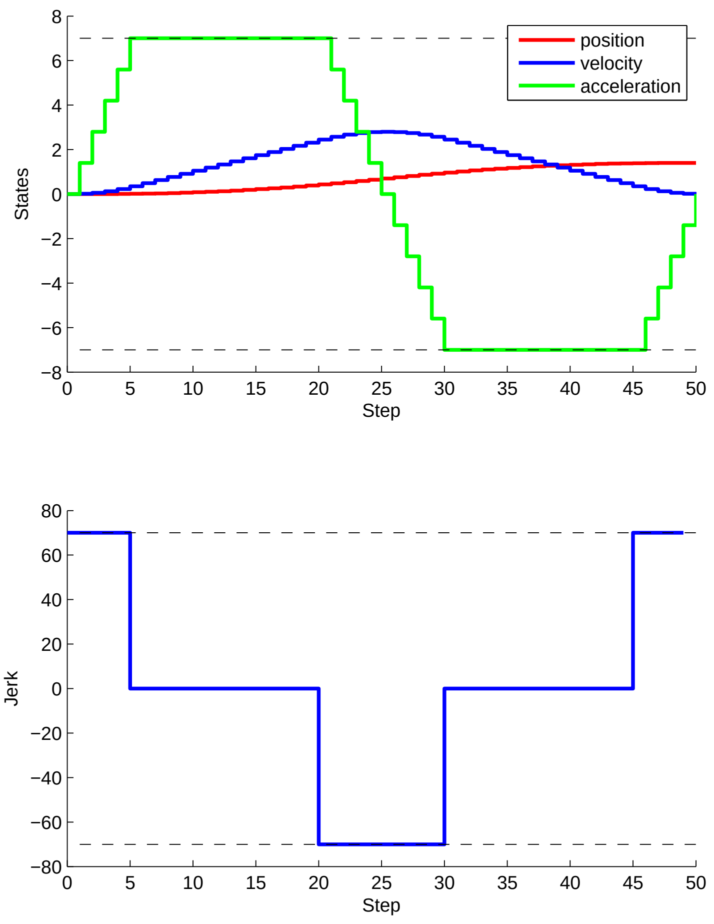

Figure 3.3: 1.4 meters rest to rest translation within 1 second - Generated trajectories for states and jerk

x_3 is illustrated in order to understand how it is rotating its frame while following the trajectory. It is important to notice that a different acceleration is associated to each attitude of the frame. Hence the greater the angle about the x_2-axis is, the more the quadcopter will accelerate in x_1-direction.

As illustrated in figure 3.2 the trajectory correctly reaches the 1.3m position. The quadcopter starts at rest position and arrives at resting conditions, as the initial and final velocities show. The saturations of acceleration and jerk are evident. The accelaration reaches its upper limit of $7\frac{m}{s}$ at time step $k = 8$ and its lower limit for $k = 37$ while decelerating.

Due to the hard constraints on the final state as well as to the limits on the maximal values reachable by jerk and acceleration, it is obvious that there will be a maximal translation achievable within the prediction horizon $T_H = 1s$. To find out its value various trajectories having different final positions to reach were solved. Starting with $x_T = 1m$ and increasing the value by 0.001m for every iteration, the last feasible trajectory, i.e. where the solver still returns a valid solution, is found to be the one for $x_T = 1.4m$. This means that no greater translations can be achieved within 1 second starting from rest and arriving at rest conditions. The results for this solution are plotted in figure 3.3. As shown, first the jerk is saturated at $j = 70\frac{m}{s^3}$. As soon as the acceleration reaches its maximum the jerk has to assume the zero value, because a further increasing of $\ddot{x}$ is not feasible. Then the acceleration has to switch from a positive to a negative value in order to decelerate and bring the quadcopter to rest in final position. This is done with the maximum achievable negative gradient, that is the minimum value of the jerk: $j_{min} = -70\frac{m}{s^3}$, bringing $\ddot{x}$ again to saturation. This is the case for which the quadcopter employs its maximum performance, satisfying the given boundary conditions.

The only way to increase the maximum achievable distance is to set a longer prediction time T_H, as the constraints on jerk and acceleration come from physical limitations and therefore can't be modified. To do this there are two main options. The first is to choose a longer prediction horizon N. Although, this will greatly increase the time needed for the solver to generate a solution, due to the large number of manipulated variables. The second option consists in setting a longer sampling time t_s, although it can't be pushed beyond a certain limit, due to the stability problems mentioned in subsection 3.2.1. Trials were done with $t_s = 30ms$ maintaining the value of the prediction horizon at $N = 50$. Again the experiments were started with $x_T = 1m$ and the final position was increased by 0.001m each iteration to see which was the maximum translation from rest to rest that could be reached. The result was that the last feasible solution was the one for $x_T = 3.4m$. This means that by increasing the sampling time by only 50% it has been achieved to extend the reachable position by ca. 143%.

It should not be forgotten, that for now it has been discussed about the first optimization problem that has to be solved, that is for the instant $t = 0$. In real MPC applications the solver will have to optimize a trajectory every t_s seconds from the actual position and state of the vehicle, in order to perform the feedback task.

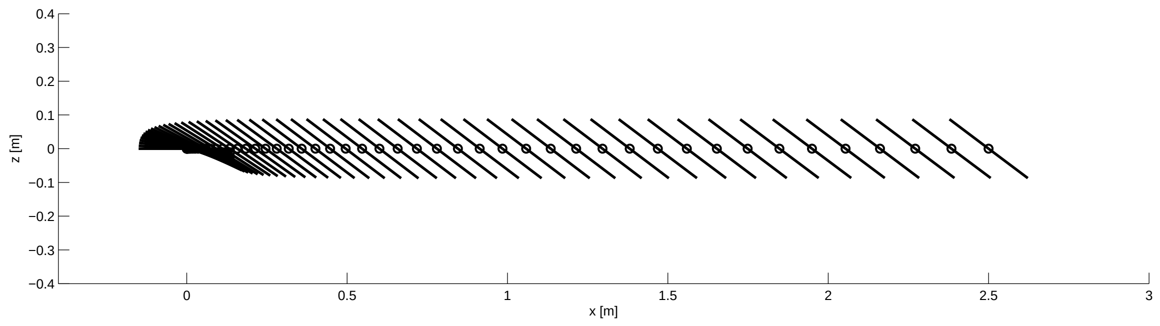

Figure 3.4: 2.5 meters rest to target translation - Trajectory of the frame in x-z plane

3.4.2 Rest position to target

In this subsection the case in which a position has to be reached within the horizon time starting from rest, is considered. Here there are no constraints on the velocity and acceleration of the final state, which means the only task is to reach the target without caring about the speed or acceleration owned in $k = N$. This means that constraint (3.40) will be replaced with:

$$x_T \leq \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} z_N \leq x_T \tag{3.41}$$

with $z_n = \begin{bmatrix} x_N \\ \dot{x}_N \\ \ddot{x}_N \end{bmatrix}$ and x_T coordinate on the x_1-axis of the target to be reached. The first trial considers $x_T = 2.5m$. Again sampling time and horizon are set to be $t_s = 20ms$ and $N = 50$ and the constraints on jerk and acceleration are the same as before. Figure 3.5 shows the results. As it is noticeable velocity and acceleration aren't equal to zero at the final step. The jerk is minimized and the constraints are satisfied, e.g. the acceleration is saturated starting at $k = 27$. The position is reached at the final step while still accelerating. Figure 3.4 shows the rotation of the quadcopter frame in the x_1-x_3 plane. As before, trials were done in order to find out what the maximal reacheable displacement in a time lapse of 1 second is: the resulting value was $3.2m$. Setting $t_s = 30ms$ and therefore having a horizon $T_H = 1.5s$ a maximal translation of $7.4m$ is gained.

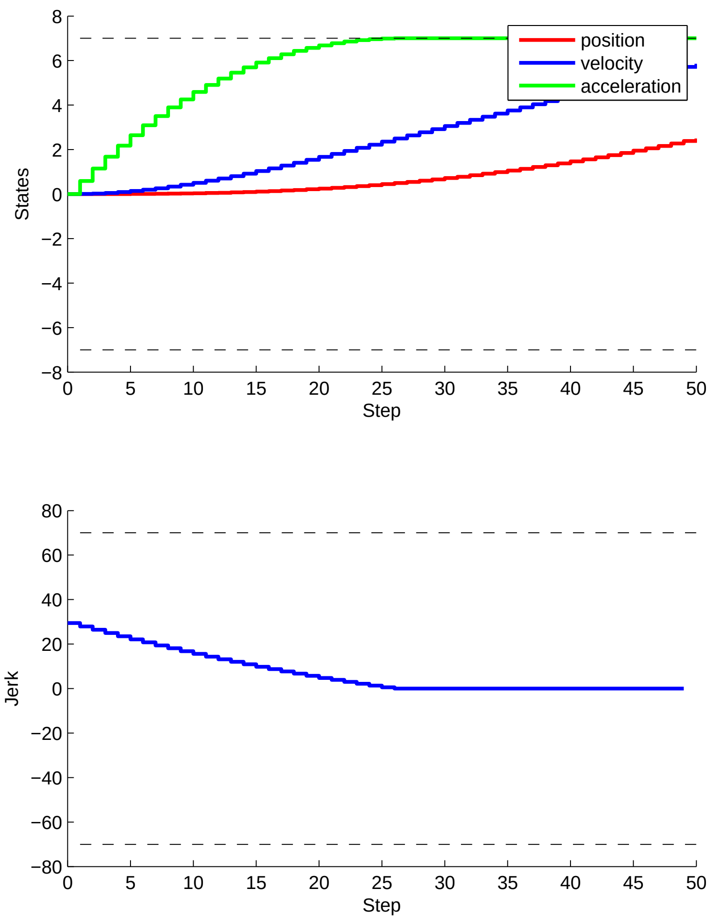

Figure 3.5: 2.5 meters rest to target translation - generated trajectories for states and jerk

3.5 Trajectory planning for distant targets

In this section the case is examined, in which the prediction horizon isn't able to cover the whole maneuver. For example, consider the case in which the target is located 15 meters away from the initial position of the quadcopter. For this situation it is impossible to set hard constraints on the final step of the prediction horizon, due to the limitations seen in section 3.4 as regards the maximal translation achievable within the predictive horizon. Hence it is necessary to consider the target in the objective function and no more as a constraint. To do this it is convenient to weight the deviation between the predicted states and the state to be achieved into the cost function. For the considerations made in section 3.3 the problem will be separated again in three different optimization tasks, one for each axis. This means that each of the three solvers takes care of finding an optimal trajectory for its own axis. Note that in this case the first trajectory that will be calculated won't reach the target but will only come closer to it. However the MPC strategy works online, which means that after applying the first control action to the system, a new solution will be calculated. This is done every t_s seconds, which means that every new trajectory will get closer to the target, since the vehicle is moving towards it.

3.5.1 Defining the cost function

As mentioned before, the deviation to target has to be considered into the cost function. Thus a term is added to the objective function depending on the predicted states. The subscripts are neglected for convenience, since the problem can be written in the same way for all three axes.

$$J_z = \frac{1}{2}\sum_{k=1}^{N}(z_T - z_k)^T Q (z_T - z_k) \tag{3.42}$$

$$Q = \begin{bmatrix} w_x & 0 & 0 \\ 0 & w_{\dot{x}} & 0 \\ 0 & 0 & w_{\ddot{x}} \end{bmatrix} \tag{3.43}$$

$$z_T = \begin{bmatrix} x_T \\ \dot{x}_T \\ \ddot{x}_T \end{bmatrix} \tag{3.44}$$

The vector z_T contains the position, velocity and acceleration to be reached. The terms w_x, $w_{\dot{x}}$ and $w_{\ddot{x}}$ represent respectively the weights on position, velocity and acceleration error. Minimizing the cost function (3.42) means trying to reduce the deviation between predicted state and target state for each step of the horizon, which leads the trajectory to approach the target. The aim is to choose the values of w_x, $w_{\dot{x}}$ and $w_{\ddot{x}}$ in order to achieve the best performances as regards the reaching of the arrival point. Again it can be decided to also minimize the jerk, so that the final cost function can be written as:

$$J = \frac{1}{2}\left(\sum_{k=1}^{N}(z_T - z_k)^T Q (z_T - z_k) + \sum_{k=0}^{N-1} w_j j_k^2\right) \tag{3.45}$$

Here a term w_j was added in order to be able to weigh the jerk relatively to the states. As specified in subsection 3.2.2 the cost function has to be formulated in terms of the manipulated variables. To do this, (3.45) will be first written in matrix form:

$$J = \frac{1}{2}\left[(\underline{Z}_T - \underline{Z})^T \underline{Q}\,(\underline{Z}_T - \underline{Z}) + \underline{U}^T \underline{R}\,\underline{U}\right] \tag{3.46}$$

with $\underline{Q} = \begin{bmatrix} Q & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q \end{bmatrix}$, $\underline{R} = \begin{bmatrix} w_j & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & w_j \end{bmatrix}$ and $\underline{Z}_T = \begin{bmatrix} z_T \\ \vdots \\ z_T \end{bmatrix}$

$\underline{Z}$ contains the predicted states along the horizon, whereas $\underline{U}$ contains all the manipulated variables. Substituting (3.15) in (3.46), rearranging the terms and eliminating the constant terms, as they do not affect the minimization process, yields:

$$J^* = \frac{1}{2}\underline{U}^T \left(\underbrace{\underline{B}^T \underline{Q}\,\underline{B} + \underline{R}}_{H}\right) \underline{U} + \underline{U}^T \underbrace{\underline{B}^T \underline{Q}\,(\underline{A} z_0 - \underline{Z}_T)}_{g} \tag{3.47}$$

3.5.2 Setting the parameters

Having defined the cost function the values of the weights have to be chosen in order to achieve the desired performances. The reference distance will be set to $x_T = 15m$, which has to be reached at rest. Thus the target state will be $z_T = [15m\, 0 \quad 0]^T$. Again $t_s = 20ms$ and $N = 50$ are set. Initially only the position error is weighed, which means $w_x = 1$ and $w_{\dot{x}} = w_{\ddot{x}} = w_j = 0$. Note that in this case the value of w_x is irrelevant, provided it is different to zero. In fact the weights in the cost function merely have a relational meaning. Implementing the MPC control strategy into the system model and simulating leads to the results depicted in figures 3.6 and 3.7.

As can be seen, the translation has a strong overshoot about the reference point. This is due to the fact that only the position is weighed, hence the controller first gives full acceleration to the system in order to get to the target as soon as possible. MPC will start to decelerate only shortly before the position is reached, namely at 1.72 seconds,

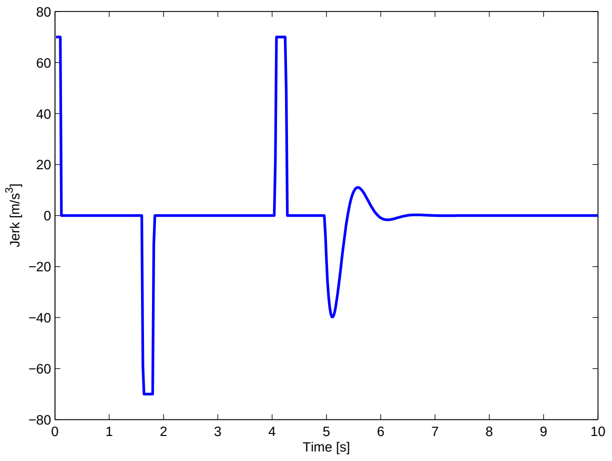

Figure 3.6: 15 meters translation - Jerk - Only the position is weighed in the cost function - $w_x = 1; w_{\dot{x}} = w_{\ddot{x}} = w_j = 0$

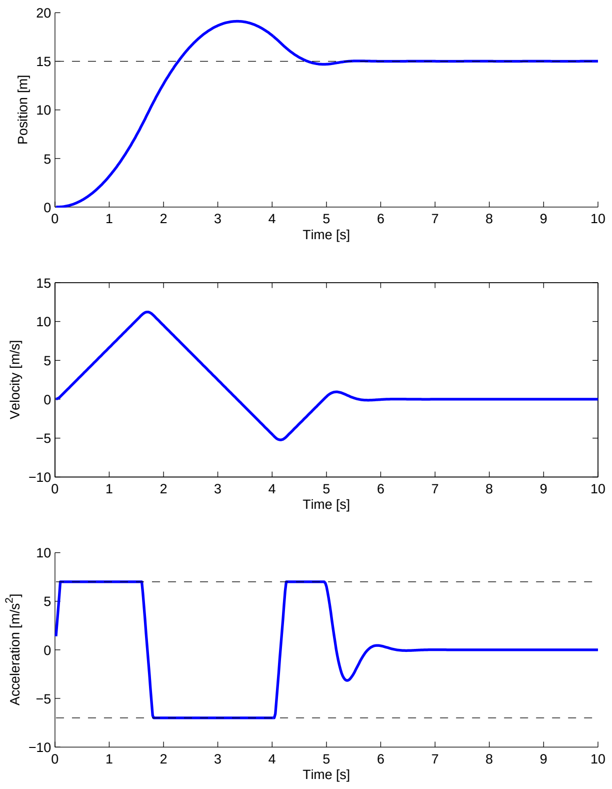

Figure 3.7: 15 meters translation - States - Only the position is weighed in the cost function - $w_x = 1; w_{\dot{x}} = w_{\ddot{x}} = w_j = 0$

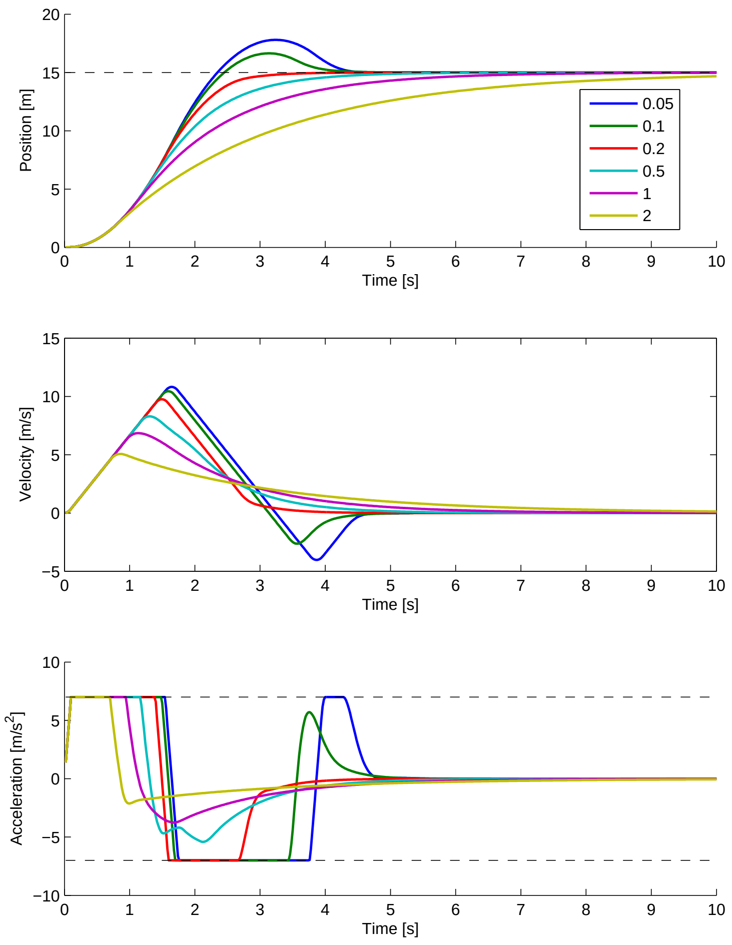

Figure 3.8: 15 meters translation - States for different velocity weighting terms $w_{\dot{x}}$ - $w_x = 1; w_{\ddot{x}} = w_j = 0$

which however is too late to avoide an overshoot. The target is first approached at ca. 2.28 seconds, whereas the acceleration starts to decrease at 1.62 seconds. This means the MPC reacts to the approach of the target only 0.66 seconds before reaching it. To avoid an overshoot it is therefore necessary to weigh also the velocity error in order to keep it under control and give the system a damping action.

Figure 3.8 shows the response of the system for different values of the weighting term $w_{\dot{x}}$. As can be seen, the more the velocity error is weighed the more the system response is damped. Too low values of $w_{\dot{x}}$ lead to overshoots, whereas too high values lead to too long transient times for the quadcopter to reach the reference. In this case

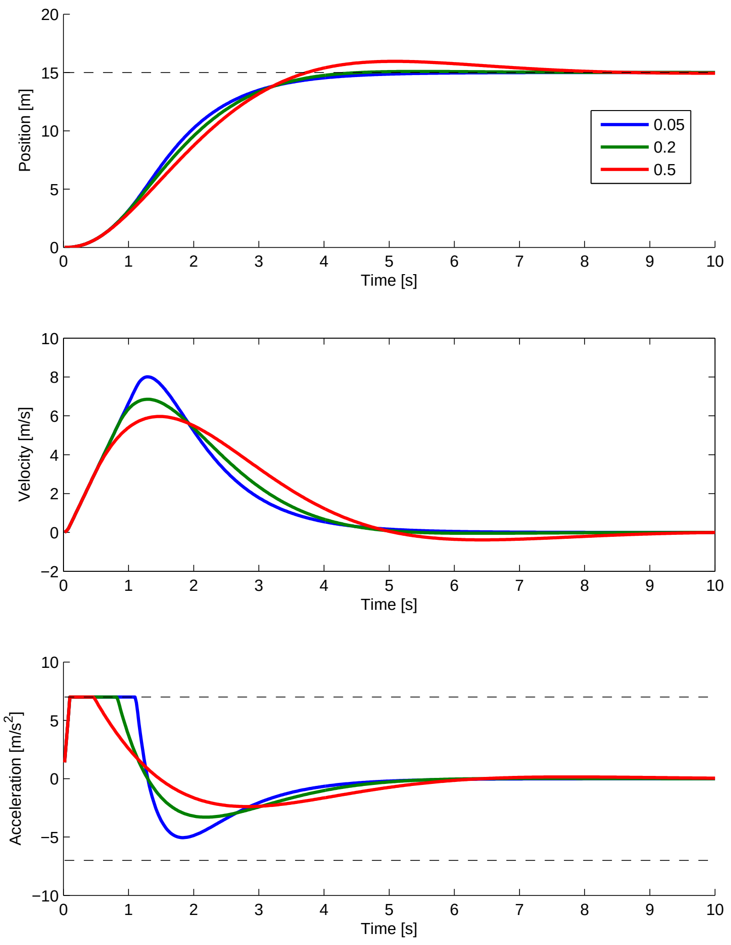

Figure 3.9: 15 meters translation - States for different acceleration weighting terms $w_{\ddot{x}}$ - $w_x = 1; w_{\dot{x}} = 0.5; w_j = 0$

the choice $w_{\dot{x}} = 0.5$ seems to be a good compromise. Note that the larger the value of the velocity-weighting term, the earlier the control reacts to an approaching of the target. For example for $w_{\dot{x}} = 2$ the quadcopter already starts decelerating at 0.9 seconds.

Figure 3.9 shows the trajectories for different values of the acceleration-weighting term $w_{\ddot{x}}$. The other terms are chosen as $w_x = 1$, $w_{\dot{x}} = 0.5$ and $w_j = 0$. Considering the acceleration into the cost function leads to a smoother progress of the latter and in some cases can help to achieve better performances. However, a too large value can lead to overshoots. To remark the effect of the prediction horizon, different simulations with various values of N were done. In figure 3.10 some of the results are plotted. For this

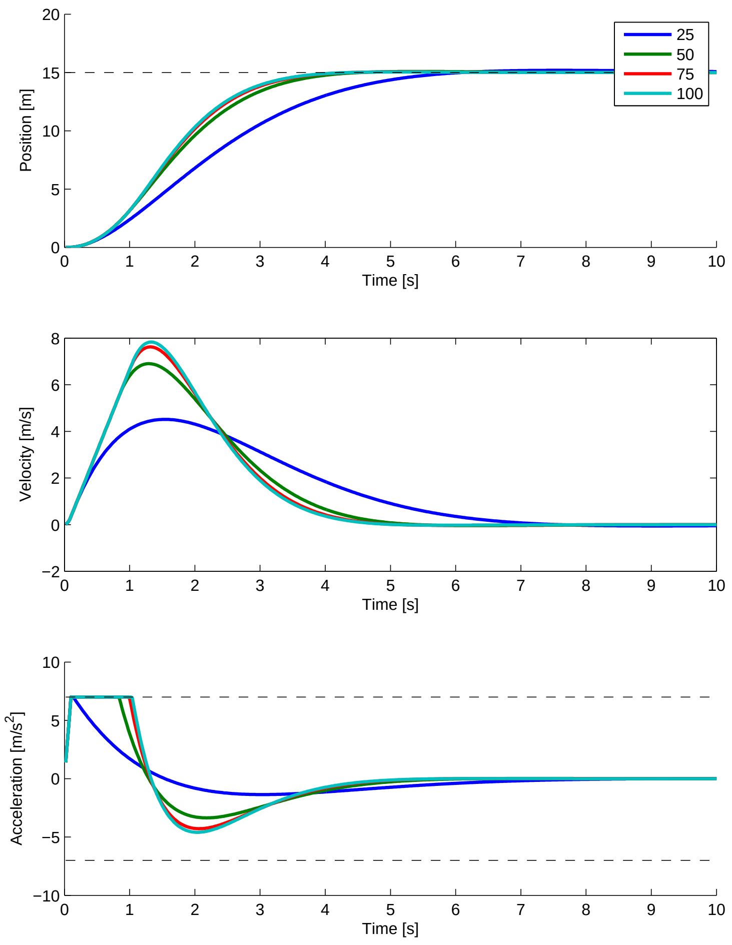

Figure 3.10: 15 meters translation - States for different predictive horizons N - $w_x = 1; w_{\dot{x}} = 0.5; w_{\ddot{x}} = 0.2; w_j = 0$

tests the weighting terms are set to $w_x = 1$, $w_{\dot{x}} = 0.5$,$w_{\ddot{x}} = 0.2$ and $w_j = 0$. As it is notable shorter prediction horizons bring the system to reach the target in a longer time. Consider that a larger prediction time involves a greater number of variables and therefore a longer computational time. As can be seen, changing the prediction horizon from $N = 50$ to $N = 100$ doesn't lead to great improvements. Thus a lengthening of the time window does not justify an increase of the computational time.

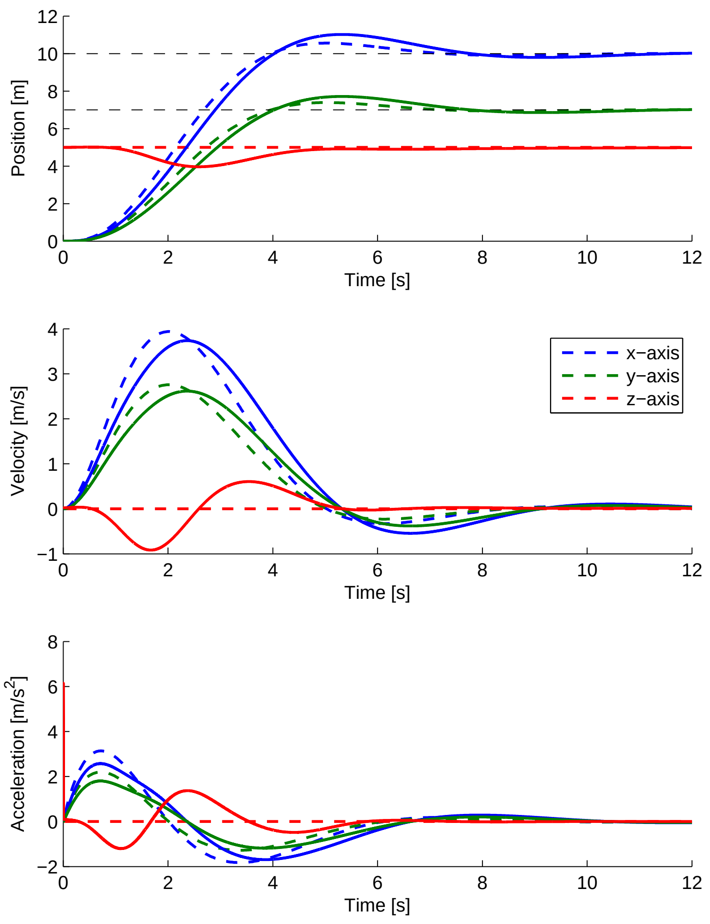

Figure 3.11: Comparison between complex system model and MPC system model

3.5.3 Simulations with complete model

To verify the performance of the controller it was implemented in a complex simulation model that takes into account also the aerodynamic effects as well as the rotational dynamics of the quadcopter. It also considers the controller employed to track the required rotational rates. To point out the simplifications made for the plant model used for the MPC the following case is considered. The quadcopter starts at rest in the position $x_0 = [0 \quad 0 \quad 5m]^T$ and has to reach the point $x_T = [10m \quad 7m \quad 5m]^T$ at rest. The following weights for position, velocity and acceleration error were set: $w_x = 1$, $w_{\dot{x}} = 0.5$, $w_{\ddot{x}} = 0.2$. This time the choice was to weigh also the jerk with $w_j = 0.1$. Figure 3.11 compares the results obtained by applying the controller to the complete model with those gained implementing it in the model considered by the model predictive controller itself (dashed lines). As can be seen the quadcopter loses altitude by ca. 1 meter. This is due to the aerodynamic forces, as at that point the system reaches a peak speed (ca. $4.6\frac{m}{s}$) at which aerodynamic effects contribute significantly to the system dynamics. Note that the drop of the vertical coordinate arises as the speed module reaches its maximum. A plausible explanation could be that the controllers for the two horizontal axes make sure to provide a rotational rate needed to counteract the aerodynamic forces. This implies a larger rotation of the frame in order to increase the horizontal acceleration. Hence the vertical component of the total thrust decreases, leading to a drop of altitude the vertical controller will have to cope with. Note that the total thrust is calculated without considering the aerdynamical forces (See (2.11)). Fortunately, as mentioned in section 3.1, MPC is able to compensate for modeling errors thanks to its online recalculation of trajectories. The case was presented on purpose in order to highlight the problems that could arise neglecting the aerodynamic effects. Obviously the performance of the altitude tracking can be improved for example by increasing the position weighting term w_x of the MPC acting on the vertical axis. However this could lead to overshoots in case the reference altitude changes too rapidly (see 3.5.2).

4 Obstacle avoidance

In Chapter 3 a method to generate trajectories online for reaching given reference states and positions is described. Consider now the task for which the quadcopter has to fly in a given direction, for example towards a given target point, while avoiding the obstacles that may present on the way. It is assumed that a measurement system, capable of detecting obstacles, is passing informations to the trajectory planner about their position, dimension and possibly velocity in case of moving obstacles. While the height and width of obstracting objects may be measured easily for example exploiting cameras or laser scanners directly mounted on the vehicle, obtaining informations about their length may look like a strong hypothesis. In reality, it is common to make predictions based on previous experience. So it is normal to expect for example a tree beeing approximately equally spreaded in all its horizontal directions.

There are mainly two ways to consider obstacles into an optimization problem. The first is to describe it in the cost function. To do this it is necessary to have a nonlinear objective function that considers the distance to the obstacle. In [13] the collision-avoidance is carried out by considering the inverse-square of the distance into the cost function. Thus minimizing this term results in maximizing the absolute value of the distance to an obstacle leading the vehicle to stay away from it. Unfortunately, since we are considering a quadratic cost function, this is not possible. This leads to the second way to cope with the collision-avoidance task, that is considering the obstacles as hard constraints. Dealing with a linear discrete-time MPC this is not a simple task, since the equations that may describe an obstacle are strongly non-linear. Furthermore the region described by the boundary conditions has to be convex. This two conditions bring to make strong approximations on the flyable region. In fact, considering for simplicity the two-dimensional case, linear constraints on the positions can only consider polygonal flyable areas.

In this chapter the inertial axes are referred to as longitudinal, lateral and vertical axis and are denoted as x,y and z.

4.1 Convex polyhedral approximation approach

A method to solve the obstacle avoidance task with MPC exploiting linear contraints is proposed in [8]. The method consists of approximating the non-convex feasible space of interest for navigation with convex polyhedra, defined for every point in time. The main idea is to find a convex polyhedron that does not contain any of the obstacles, in order to constrain the solution to lie within the latter. For every instant in which a new trajectory is generated, a different polyhedron is evaluated according to the actual position of the vehicle. All the positions composing the prediction horizon must lie within the considered polyhedron.

4.1.1 Definition of the polyherdon

Let $p = [x\ y\ z]^T$ denote the position of the vehicle and let M denote the number of obstacles to be avoided. Each obstacle is described by a convex polyhedron $W_i \subset \mathbb{R}^3$ centered on a different point $q_i \in \mathbb{R}^3$. Thus the set $\{q_i\} \oplus W_i$ is considered as infeasible.

In order to impose linear constraints, the nonconvex feasible space where the vehicle can navigate must be under-approximated by a convex polyhedron. An algorithm that maximizes the size of a polyhedron not containing a set of points is described as follows for a generic space-dimension d:

Let $p_0, q_1, q_2, ..., q_M \in \mathbb{R}^d$, with $p_0 \neq q_i, \forall i = 1, ..., M$.

The polyhedron $P = \{p \in \mathbb{R}^d : \quad A_c p \leq b_c\}$ with

$$A_c = \begin{bmatrix} (q_1 - p_0)^T \\ \vdots \\ (q_M - p_0)^T \end{bmatrix} \tag{4.1}$$

$$b_c = \begin{bmatrix} (q_1 - p_0)^T q_1 \\ \vdots \\ (q_M - p_0)^T q_M \end{bmatrix} \tag{4.2}$$

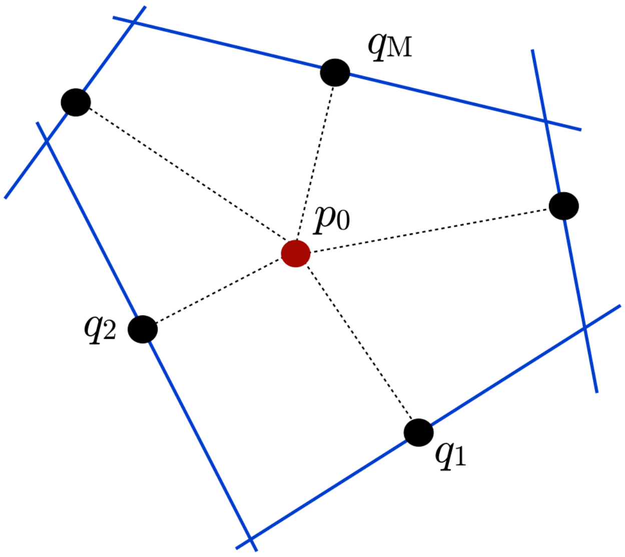

Figure 4.1: Convex polyhedron avoiding points $q_1, ...q_M$

contains p_0 in its interior and does not contain any of the points q_i, for all $i = 1, ..., M$. The proof is described in [8]. Every boundary condition $(q_i - p_0)^T p \leq (q_i - p_0)^T q_i$ represents a halfspace, on the boundary of which lies the point q_i. Furthermore, its boundary is orthogonal to $q_i - p_0$ and p_0 is in the interior of the halfspace. The intersection of more halfspaces, each of which excludes a point q_i, results in a polyhedron that does not contain any of the points q_i. Figure 4.1 gives a graphical representation of this concept in two dimensions. Every line represents a boundary condition cutting off a halfspace. Note that (A_c, b_c) in (4.1) and (4.2) may not be a minimal hyperplane representation of P, since some points may be already left out from other halfspaces.

However, the obstacles are not points but convex polyhedra. Hence the polyhedron describing the feasible space has to be defined in the following way.
Let $p_0, q_1, q_2, ..., q_M \in \mathbb{R}^3$, with $p_0 \neq q_i, \forall i = 1, ..., M$ and let $W_1, ..., W_M$ be polyhedra in $\mathbb{R}^d$. Let A_c,b_c be defined as in (4.1) and (4.2) and let $g \in \mathbb{R}^M$ such that its j-th component g^j is defined as

$$g^j = \min_{w \in \mathbb{R}^d} \quad A_c^j w \qquad\qquad s.t. \quad w \in W_j \tag{4.3}$$

for $j = 1, ..., M$. Then the polyhedron $P = \{p \in \mathbb{R}^d : \quad A_c p \leq b_c + g\}$ does not contain any polyhedron $B_j = \{q_i\} \oplus W_i$ in its interior, $\forall i = 1, ..., M$. The proof is described in [8]. If W_j's are polytopes and their vertex representation is $W_j = \text{conv}\{w_{j1}, ..., w_{js_j}\}$, then

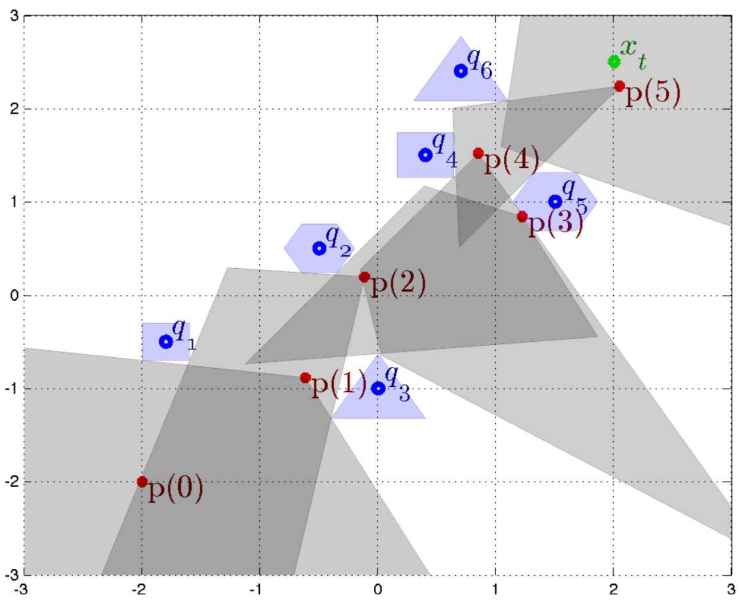

Figure 4.2: Example of feasible polyhedra - in [8]

(4.3) can be solved as

$$g^j = \min_{h=1,\ldots,s_j} \quad A_c^j w_{jh} \tag{4.4}$$

Hence every boundary condition represents a halfspace on the boundary of which lies the nearest vertex of an obstacle with respect to the position p_0 of the quadcopter. Figure 4.2 illustrates a two-dimensional example, where each polyhedron is evaluated starting from the point of the previous polyhedron, which minimizes the Euclidean distance from x_t.

For every time step a new convex polyhedron is evaluated, in which all the positions of the prediction horizon must lie. Taking into account all the polyhedra that have been considered from starting point to target, the union of all pointwise-in-time convex approximations provides a rather good non-convex approximation of the feasible space of interest for navigation.

4.1.2 Implementation in model predictive control

In [8] the overall control structure consists of a two-layer MPC. A linear time varying MPC, operating with a sampling time of $T_{sn} = 1.5s$, generates reference positions in order to avoid the obstacles and to reach a given target. The desired positions are then followed by a linear MPC, which is responsible for stabilization and position tracking and which generates a control action every $T_s = 71ms$. The convex approximation strategy for obstacle avoidance is implemented in the high level LTV-MPC. Furthermore, a maximum rate of change of the desired position is set to $\Delta_{max} = -\Delta_{min} = 0.5m$ per axis. Since the LTV-MPC has a sampling rate of $T_{sn} = 1.5s$. This means that the velocity is implicitly limited to $v_{max} = \frac{\Delta_{max}}{T_{sn}} \approx 0.33\frac{m}{s}$ per axis, which causes the system to be quite slow.

In this work the convex polyhedral approximation approach is instead implemented in the model predictive controller used so far, which takes the jerk trajectory as input of the system. The only difference is that the optimization problem can't be decoupled any more, since the boundary conditions for obstacle avoidance relate all of the spacial coordinates within the same inequality. The bounds on jerk and acceleration as well as the system model are the same as in chapter 3. The overall optimization problem can be written as:

$$min \quad \frac{1}{2}\left(\sum_{k=1}^{N}(\xi_T - \xi_k)^T Q(\xi_T - \xi_k) + \sum_{k=0}^{N-1} u_k^T R u_k \quad + w_e e^2\right) \tag{4.5}$$

$$s.t. \quad \xi_{k+1} = \tilde{A}\xi_k + \tilde{B}u_k \tag{4.6}$$

$$v_{min} \leq v_k \leq v_{max} \tag{4.7}$$

$$a_{min} \leq a_k \leq a_{max} \tag{4.8}$$

$$u_{min} \leq u_k \leq u_{max} \tag{4.9}$$

$$A_c(t)p_k \leq b_c(t) + g(t) + \mathbb{1}e \tag{4.10}$$

with $p_k = [x_k\, y_k\, z_k]^T$, $v_k = [\dot{x}_k\, \dot{y}_k\, \dot{z}_k]^T$, $a_k = [\ddot{x}_k\, \ddot{y}_k\, \ddot{z}_k]^T$, $\xi_k = [p_k\, v_k\, a_k]^T$, $u_k = [j_{xk}\, j_{yk}\, j_{zk}]^T$. In order to approach the target a term that weighs the error between state vector ξ_k and target state vector ξ_T is added to the cost function. ξ_T contains the target position and zero entries in order to be able to weigh also velocity and acceleration values: $\xi_T = [x_T \;\, y_T \;\, z_T \;\, 0 \;\, 0 \;\, 0 \;\, 0 \;\, 0 \;\, 0]^T$. Q and R are the weighting matrices for states and inputs:

$$Q = \begin{bmatrix} w_x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_y & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_z & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{\dot{x}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{\dot{y}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{\dot{z}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{\ddot{x}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{\ddot{y}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{\ddot{z}} \end{bmatrix} ; \quad R = \begin{bmatrix} w_{jx} & 0 & 0 \\ 0 & w_{jy} & 0 \\ 0 & 0 & w_{jz} \end{bmatrix} \tag{4.11}$$

Inequality (4.10) defines the boundary conditions for obstacle avoidance as described in subsection 4.1.1. Note that the same boundary conditions are applied to all the horizon steps k, since all positions must lie within the same polyhedron. Furthermore, a slack variable is added in order to soften the constraints, avoiding that the optimization problem is infeasible. The latter is penalized by a large weight w_e in the cost function. The limits v_{min}, v_{max}, a_{min}, a_{max}, u_{min} and u_{max} all contain three elements, one for each axis. Note that a boundary condition is added in order to limit the maximum and minimum velocity.

$\tilde{A}$ and $\tilde{B}$ define the system model and can be inferred from (3.33) by taking into account all three axes and rearranging the terms:

$$\tilde{A} = \begin{bmatrix} 1 & 0 & 0 & t_s & 0 & 0 & \frac{1}{2}t_s^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & t_s & 0 & 0 & \frac{1}{2}t_s^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & t_s & 0 & 0 & \frac{1}{2}t_s^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & t_s & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & t_s & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & t_s \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} ; \quad \tilde{B} = \begin{bmatrix} \frac{1}{6}t_s^3 & 0 & 0 \\ 0 & \frac{1}{6}t_s^3 & 0 \\ 0 & 0 & \frac{1}{6}t_s^3 \\ \frac{1}{2}t_s^2 & 0 & 0 \\ 0 & \frac{1}{2}t_s^2 & 0 \\ 0 & 0 & \frac{1}{2}t_s^2 \\ t_s & 0 & 0 \\ 0 & t_s & 0 \\ 0 & 0 & t_s \end{bmatrix} \tag{4.12}$$

Note that $A_c(t)$, $b_c(t)$ and $g(t)$ depend on time, since a new polyhedron has to be evaluated every time a new trajectory is solved. Thus the control strategy becomes a linear

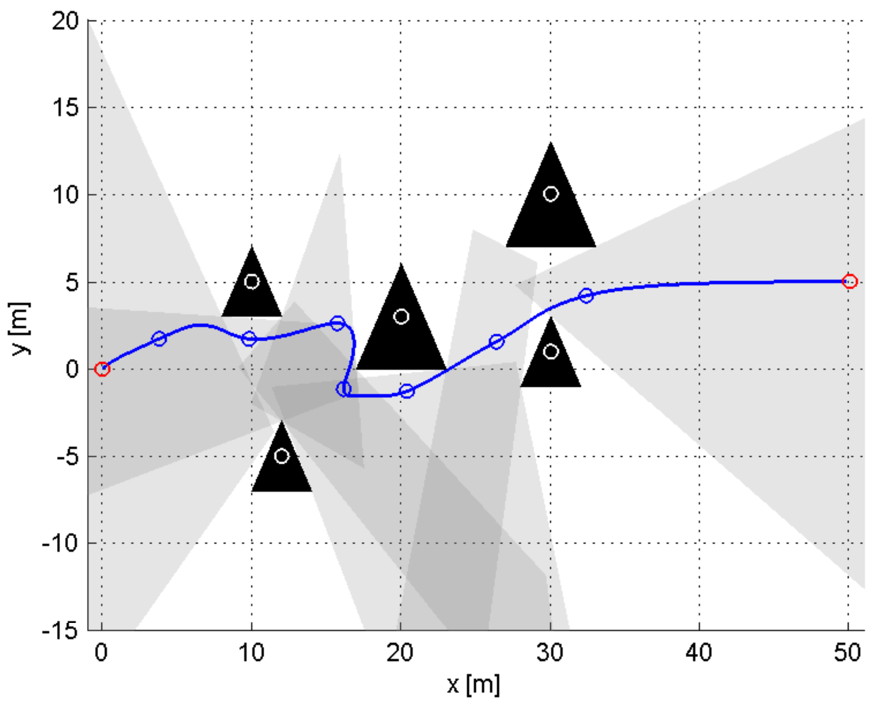

Figure 4.3: Trajectory avoiding obstacles - some polyhedra have been plotted

time varying model predictive control (LTV-MPC). $A_c(t)$, $b_c(t)$ and $g(t)$ are obtained by equations (4.1), (4.2) and (4.4) by setting $p_0 = p(t)$ (current vehicle position) and by knowing positions and dimensions of the obstacles: q_i, $W_j = \text{conv}\{w_{j1}, ..., w_{js_j}\}$.

4.1.3 Simulation results

The first trial considers a two-dimensional environment with five triangular obstacles. The vehicle starts at $p_0 = [0 \;\; 0]^T$ and has to reach the target position $p_T = [50 \;\; 5]^T$. The prediction horizon is choosen to $N = 20$ and the sampling time to $t_s = 30ms$, thus the prediction window is $T_H = 0.6s$. The limits on jerk, acceleration and velocity per axis are set to $j_{max} = -j_{min} = 70\frac{m}{s^3}$, $a_{max} = -a_{min} = 7\frac{m}{s^2}$, $v_{max} = -v_{min} = 5\frac{m}{s}$. The states-weighting terms are choosen equally for both axes: $w_x = w_y = 1$, $w_{\dot{x}} = w_{\dot{y}} = 0.5$, $w_{\ddot{x}} = w_{\ddot{y}} = 0.1$, whereas the jerk hasn't been weighed at all.

Figure 4.3 shows the resulting trajectory. The convex polyhedra describing the feasible space have been plotted for certain points in time in order to give a graphical representation of the strategy. As shown in figure, the vehicle correctly avoids the obstacle

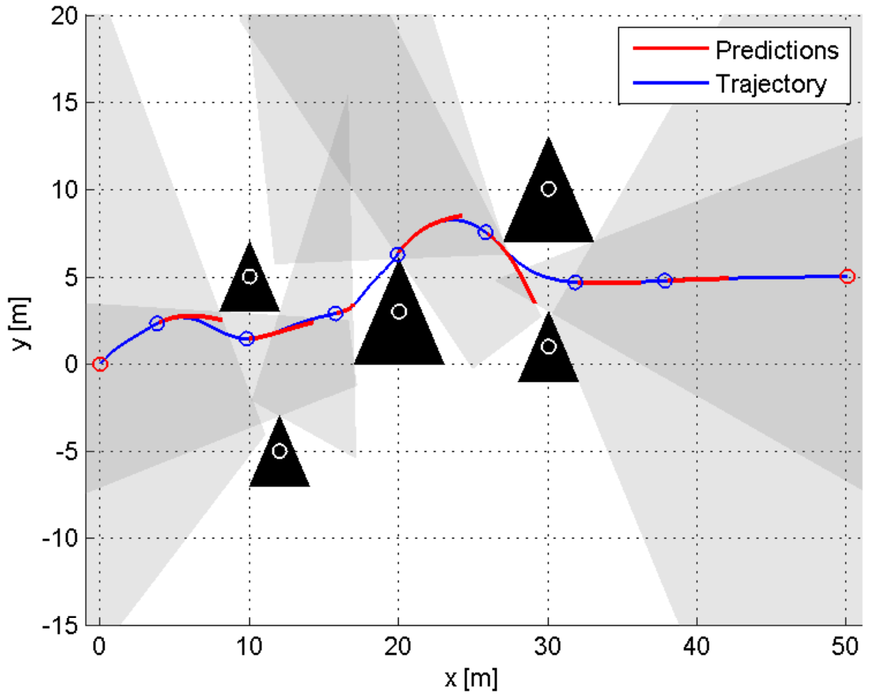

Figure 4.4: Trajectory avoiding obstacles - prediction horizon $N = 30$

and reaches the target. However, seen from a global point of view, the resulting path may not be an optimal solution. This is caused by the fact that the convex polyhedra leave out a significant part of the feasible space, which often leads the vehicle to choose disadvantageus trajectories. For instance, looking at figure 4.3, the vehicle passes the central obstacle to the right, though passing it to the left would have made more sense.

Figure 4.4 shows the same scenario, where the horizon was chosen to $N = 30$ instead. This extends the prediction time to $T_H = 0.9s$. The path chosen by the control is different from the previous one. This time the solution seems more resonable. The figure also depicts some of the predicted trajectories generated according to the plotted polyhedra. As can be seen, the boundary conditions enforce the trajectories to lie within the respective convex areas. This leads to two main disadvantages of this method, which are described below.

Figure 4.5 depicts the predictions made at two different time instants along the path. The first frame illustrates a typical problem of this strategy. As shown in figure, the boundaries cut off an important part of the real feasible space. The borders of the polyhedron squeeze the trajectory, in order to achieve that all predicted positions remain

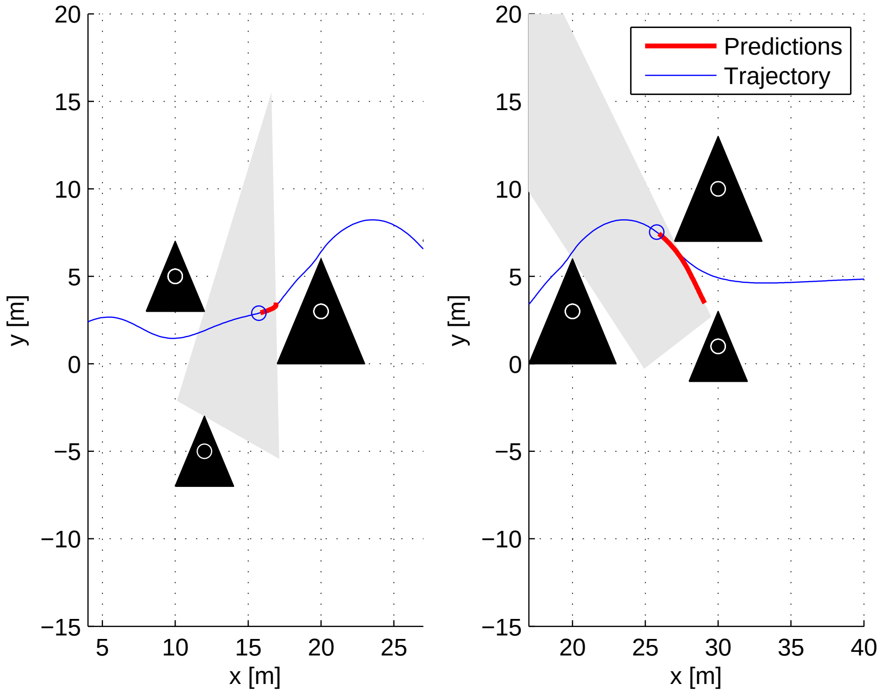

Figure 4.5: Trajectory avoiding obstacles - prediction horizon $N = 30$ - Two situations

inside the convex area. This leads to unnecessarily high decelerations. The problem can be curbed by reducing the prediction horizon. This implies to enforce a limited number of prediction points to lie within the polygon, avoiding to significantly reduce the velocities. However a too small value of N brings the trajectories to not react in time to fast changes of the polygonal boundaries. Moreover, nothing prevents the trajectory points reaching the border from having a nonzero velocity. It may therefore occur that the subsequent solution is not able to enforce its last point to satisfy the boundary conditions. Thus, small prediction times combined with high speeds may lead to trajectories lying partially out of bounds.

Another drawback of this method is represented in the second frame of figure 4.5. Since the polyhedra are a quite rough approximation of the real feasible space, it often happens, that the strategy conducts the vehicle off course. Looking at the illustration, it is noticeable how the generated trajectory tries to approach the vertex of the polyhedron which minimizes the Euclidean distance to the target. Although it is the optimal solution with respect to the boundary conditions, this may not be an appropriate direction seen from a global point of view. This inconvenience is partially reduced by the receding

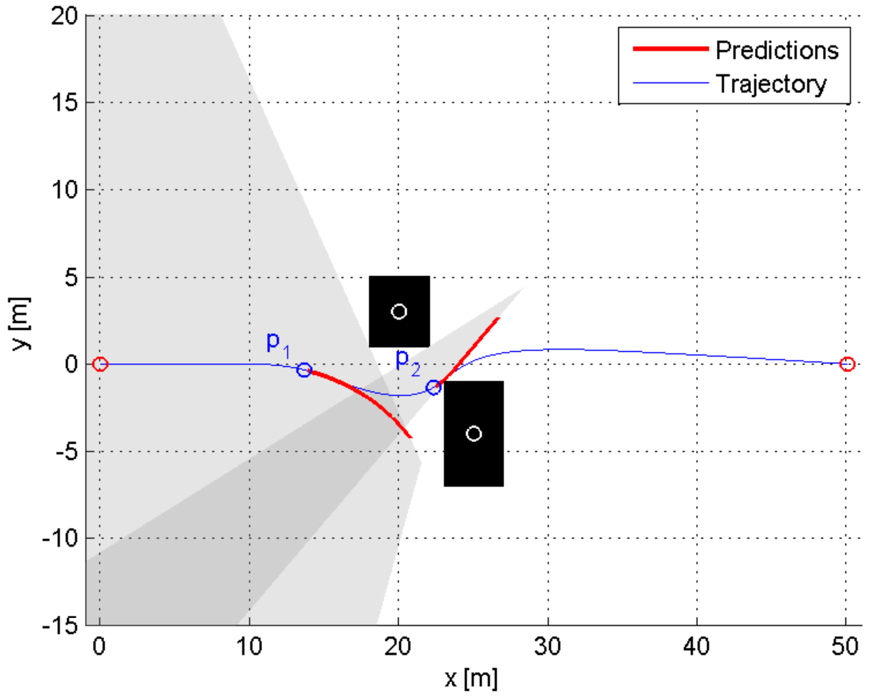

Figure 4.6: Trajectory avoiding two obstacles - prediction horizon $N = 30$

horizon strategy of MPC, which generates a new trajectory after a short time. Thus a new polygon is evaluated, which typically takes into account a new portion of the real feasible space.

4.1.4 Limits of the approach

Some trials were done in order to show the main problems associated with this method. Figure 4.6 illustrates a simple situation where the quadcopter has to move towards a target passing between two obstacles. The latter are placed so that no change in direction would be necessary in order to avoid them. However, as depicted, the trajectory bends when approaching them. The convex polyhedra representing the boundary conditions are plotted for $t_1 = 2.2s$ and $t_2 = 3.2s$. As shown in the picture, the feasible area around p_1 brings the trajectory to follow an improper direction. According to vehicle's position the polyhedron changes shape such that at a certain point the route is again adjusted, as shown by the predictions starting from p_2.

Figure 4.7 shows an even worse case. Here the prediction horizon was set to $N = 20$.

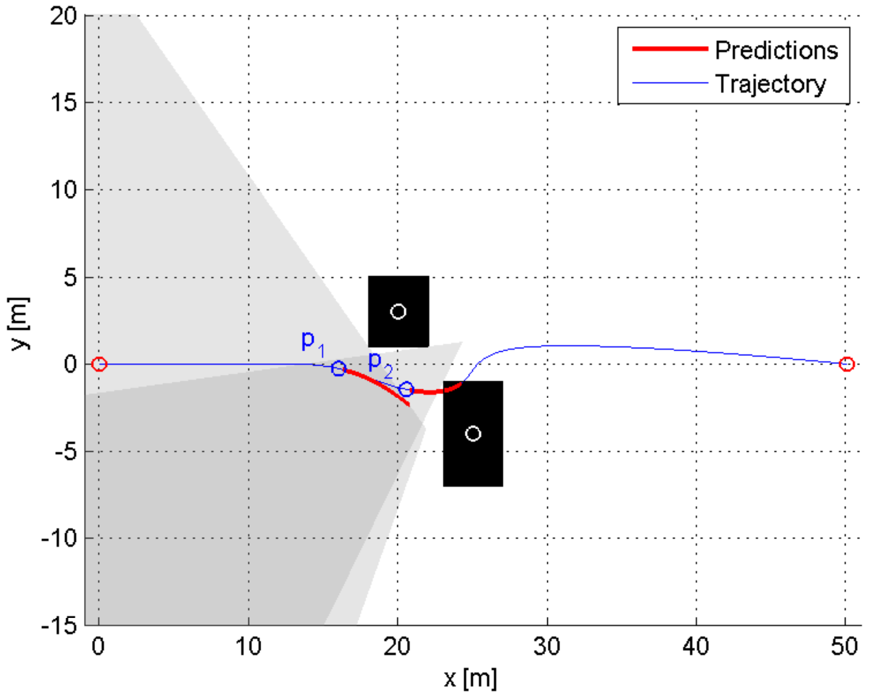

Figure 4.7: Trajectory avoiding two obstacles - prediction horizon $N = 20$

The prediction window is too short to react in time to the fast changes of the polyhedron's shape, so that, given the physical limitations on the maximum deceleration, the predicted positions are not able to lie within the convex area. This causes the vehicle to crash into the obstacle. The only solution to this problem is to set more stringent constraints on the maximum velocity, which on the other hand leads to not beeing able to manage fast flights.

Finally the case is presented in which the obstacle is located directly in front of the vehicle. For this trial the velocity is limited to $10\frac{m}{s}$, whereas the horizon parameters are chosen to $t_s = 30ms$ and $N = 50$. The obstacle is 8 meters wide and is positioned 38 meters away from the starting position of the quadcopter. Figure 4.8 shows the results captured at different time instants. The dashed lines represent the predictions. As can be seen, the boundary of the convex area always remains orthogonal to the direction of movement, since the center of the obstacle is located directly in front of the vehicle. This leads the quadcopter to not pass the obstacle at all. It correctly decelerates in order to remain within the feasible area and manages to avoid a collision, but it is not able to reach its target, since it is blocked in a deadlock situation. In fact, in order to minimize

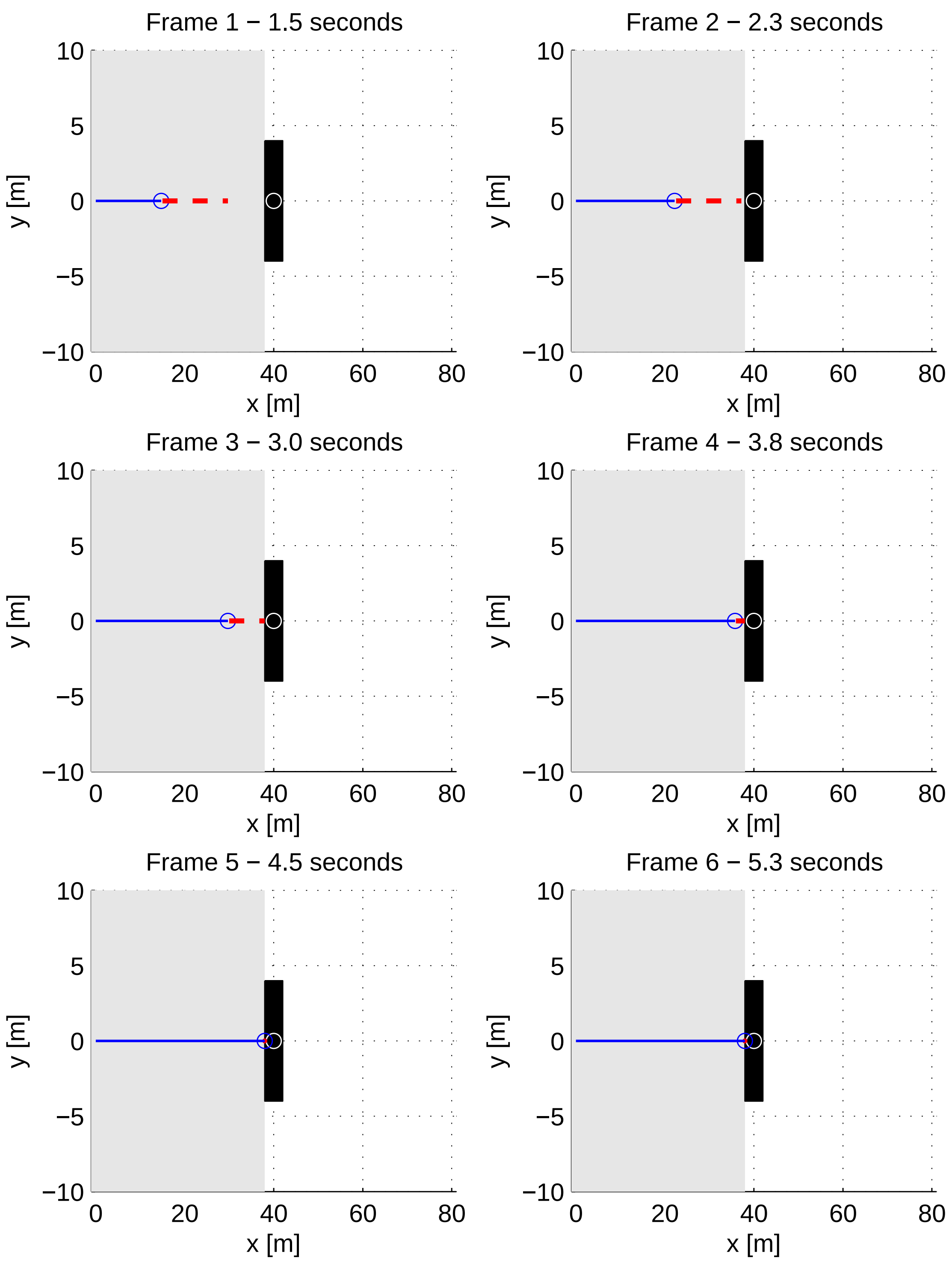

Figure 4.8: Frontal obstacle - frames

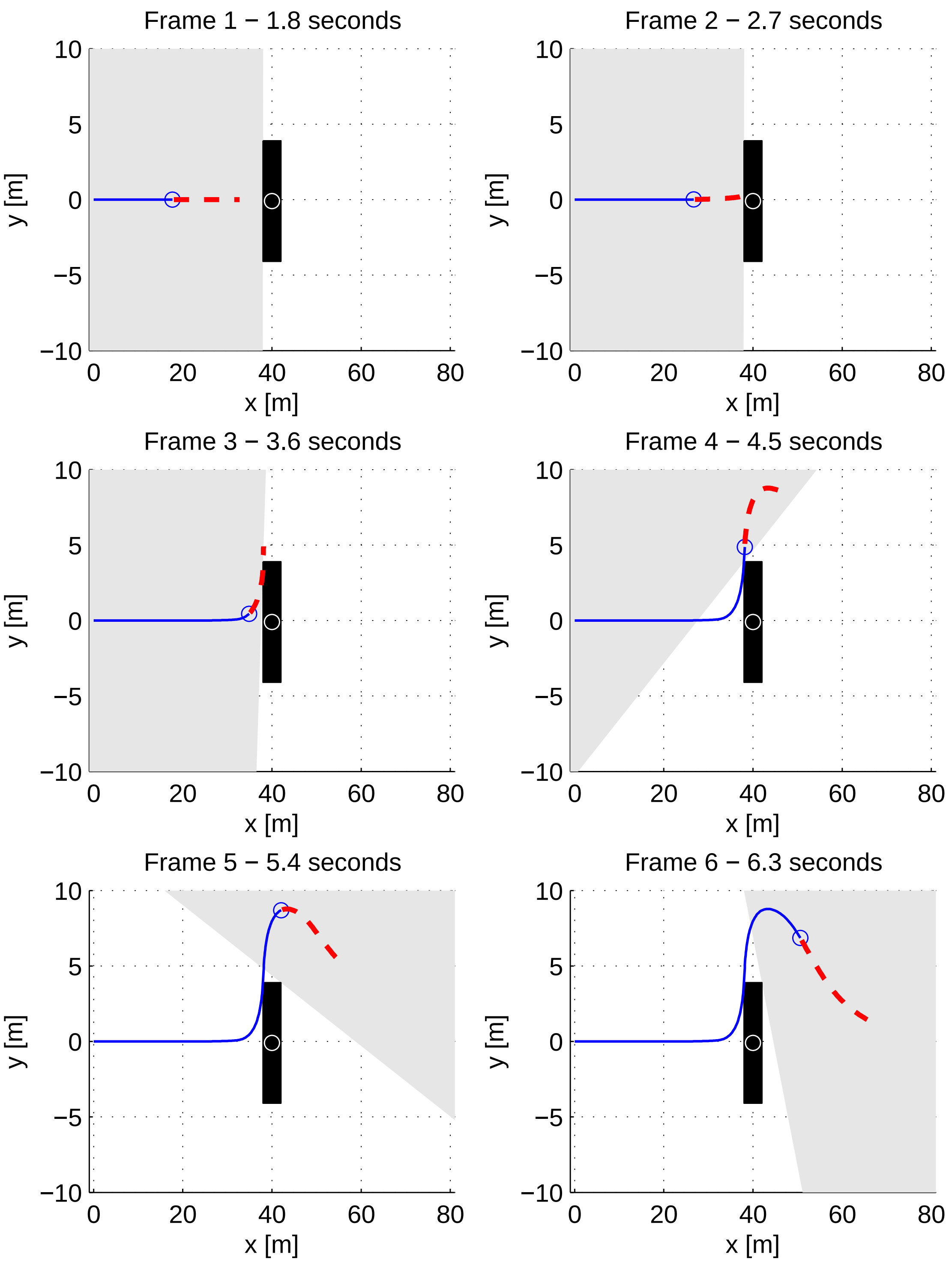

Figure 4.9: Offset obstacle - frames

the cost function, the solver searches for the position within the feasible polyhedron that minimizes the Euclidean distance to the target (which in this case is located 500 meters away from the vehicle). However this solution doesn't cause any change in the boundaries of the polyhedron, leading the vehicle to stop in front of the obstruction. Note that for shorter horizons, for example $N = 20$, the vehicle is not able to react in time to the presence of the obstacle, which leads to a collision.

In order to avoid a deadlock a similar trial was made, but this time the center of the obstacle was located slightly offset with respect to the direction of movement, namely with y-coordinate $y_c = -0.1m$. Figure 4.9 illustrates the resulting trajectory. As can be seen, the vehicle manages to pass the obstacle. However the achieved path is apparently not the best solution seen from a global point of view. The vehicle lowers its longitudinal velocity almost down to zero before starting to deviate its course in order to avoid a collision. This greatly increases the time needed to pass the obstacle. Furthermore, the y-coordinate presents a high overshoot due to the high lateral speed achieved.

All the limits of this strategy regarding the reliability of the followed paths along with the maximum speeds, that can be managed by the convex polyhedral approximation approach, brought to choose a complete different method for obstacle avoidance. The new strategy should provide smoother trajectories and more acceptable paths seen from a global point of view. Furthermore, the new control for objects avoidance should be able to cope with faster velocities, always providing collision-free paths. In the next sections the new approach is described. Moreover, at the end of this chapter, comparisons are made between the two strategies,

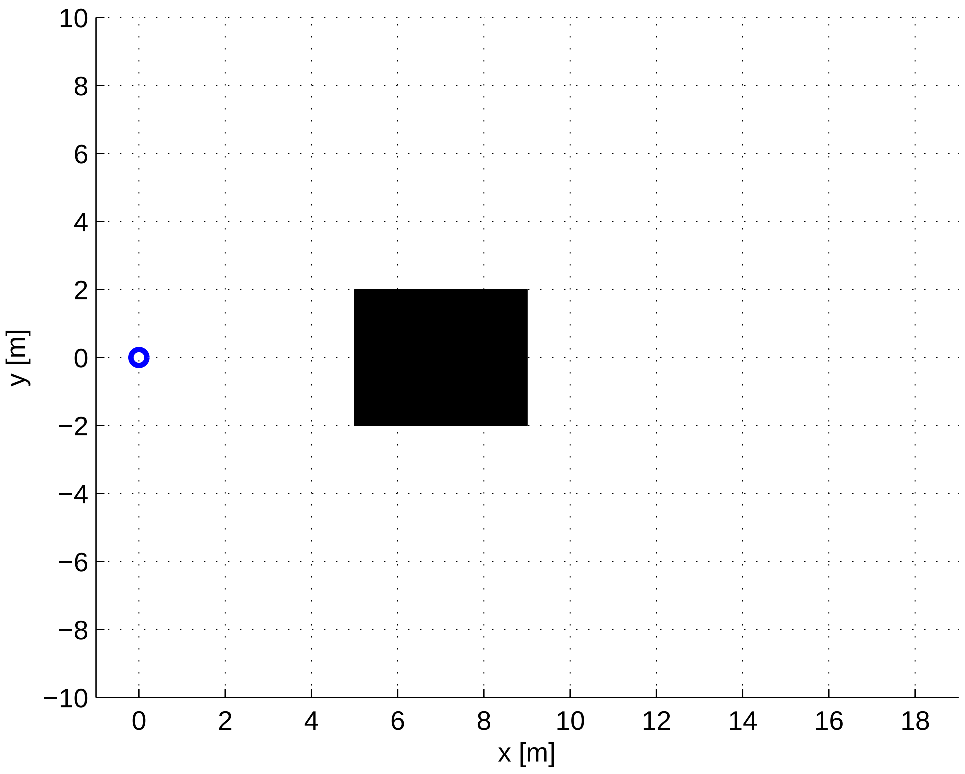

Figure 4.10: Rectangular obstacle in the x-y plane

4.2 Obstacle time-discretization method

Consider the case in which the quadcopter is flying towards a target. For simplicity it is moving in x direction. This is actually not a strong hypothesis, since one can decide to direct the coordinate system at will, thus for example with the x-axis directed towards the target. For now the two-dimensional problem is considered, as it is easier to understand. The aircraft has to move towards the target avoiding collisions. Consider now a rectangular obstacle placed five meters away from the quadcopter as depicted in figure 4.10. As mentioned in 3.2.2 it is possible to set upper and lower bounds on the system states, thus also on the positions. Once it has been decided whether to pass the object to the left or to the right one can set bounds on the y-coordinate in order to avoid a collision. To describe the obstacle in an exact way it would be necessary to write boundary conditions on the y-coordinates as function of the x-coordinate, as:

$$y_k \leq f(x_k) \qquad for \quad k = 1 \cdots n \tag{4.13}$$

Assuming to pass the object to the right the inequality described in (4.13) should limit the y-coordinate to lie under a certain value that depends on x. $f(x)$ would be a discontinuous equation. In fact it should assume the value $f = \infty$ for $x < 5$, since for this positions the quadcopter has not reached the obstacle yet and therefore no bounds should be applied, $f = -2$ for $5 \leq x \leq 11$ and again $f = \infty$ for $x > 11$. This function is not continuous, which means it can not be used as boundary condition, since convex constraints are needed. In general the equations describing the obstacle's shape are not even linear, due to various forms objects may have. Thus it is necessary to find an approximate way to describe the obstacle.

If the assumption of flying at constant speed in x direction $v_x = const.$ was made, it wouldn't be necessary to let the bounds depend on the x-coordinate. In fact, the obstacle could be discretized in the time domain setting bounds on the y-coordinate for each time step, according to the speed the quadcopter is flying with and assuming it is maintaining it for the whole predictive horizon. In order to understand this concept a numeric example is done considering the obstacle depicted in figure 4.10. Immagine to fly with a speed $v_x = 10\frac{m}{s}$ (remember this is only the x-component of the velocity) and assume to keep it constant during the whole time window considered. For this example the prediction horizon and the sampling time are set to be $N = 50$ and $t_s = 20ms$, which leads to a prediction time window of 1 second. This means the quadcopter will approach the obstacle at $t = 0.5s$, hence for time step $k = 25$, and will completely pass it at $k = 46$. Thus the boundary conditions on the position will be:

$$y_k \leq \infty \qquad for \quad k = 1 \cdots 24 \tag{4.14}$$

$$y_k \leq -2 \qquad for \quad k = 25 \cdots 45 \tag{4.15}$$

$$y_k \leq \infty \qquad for \quad k = 46 \cdots 50 \tag{4.16}$$

It is important not to confuse discontinuities of the bounds in the time domain with discontinuities in the spacial domain. Indeed for each time step k a different variable y_k is considered, which means that each y_k will have its own bounds. Thus inequality (4.13) can't be considered for the bounds depend on the x-coordinate in a discontinuous (and in general non-linear) way, but the bounds described in (4.14) - (4.16) can be applied for they do not depend on the x-coordinate at all, but only on the time. Thus, the axes can be decoupled, since the bounds on y_k do not depend on x_k. Remember that the MPC strategy works online, which means that after 20 milliseconds a new trajectory will be evaluated. Thus after $t_s = 20ms$ the quadcopter has moved 0.2 meters and the bounds for the new trajectory will be

$$y_k \leq \infty \qquad for \quad k = 1 \cdots 23 \tag{4.17}$$

$$y_k \leq -2 \qquad for \quad k = 24 \cdots 44 \tag{4.18}$$

$$y_k \leq \infty \qquad for \quad k = 45 \cdots 50 \tag{4.19}$$

The quadcopter is indeed closer to the obstacle and the bounds will be set at earlier time steps. The same procedure will be repeated every 20 milliseconds and while the obstacle comes closer it will be discretized at earlier time steps. It is fundamental to understand that, though an object can be seen far in advance from the measurement systems, it can be only taken into account for the trajectory planning if it is close enough to be included in the prediction horizon. For instance if the quadcopter is flying at $10\frac{m}{s}$ and having considered a prediction horizon of $T_H = 1s$ the obstacle can only be seen when it is closer than 10 meters, or better when it will be approached in less than 1 second according to the actual speed. Note that with this technique one can describe obstacles with various shapes. It will only be necessary to discretize it in the time domain according to the actual longitudinal speed. The discretization stepsize depends on the velocity the quadcopter is flying with and on the sampling time t_s. For example moving at $v_x = 10\frac{m}{s}$ and having a sampling time of $t_s = 20ms$ implies a stepsize of 0,2 meters. The greater the speed or the longer t_s the rougher is the discretization of the obstacle. Another thing that needs to be clarified is that the constraints can only define a convex region. For instance, given the obstacle described in figure 4.10, one can not set $-\infty \leq y_k \leq -2 \quad \cup \quad \infty \geq y_k \geq 2$, for it is not representing a convex region. For this reason whether to pass an object to the left or to the right has to be decided before solving the optimization problem. The decision was to implement an algorithm that evaluates the distances that have to be covered along the y-direction in order to pass the object, and according to the shortest way it decides whether to set lower or upper bounds on the y-coordinate.

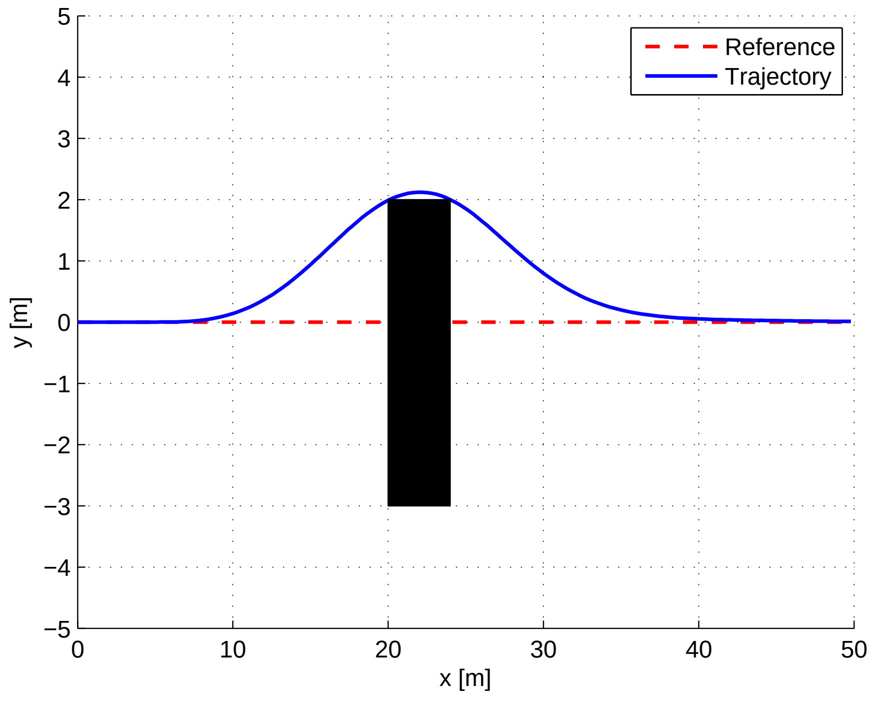

Figure 4.11: Trajectory avoiding single obstacle

4.3 Implementation in model predictive control

In 4.2 a way to implement the obstacle as boundary conditions on the y-coordinate has been defined. It is assumed that the x-axis is pointing towards the target and the longitudinal velocity v_x is maintained constant. The axes can be decoupled due to the considerations made. This means that for now, considering the two-dimensional problem, the only control responsible for the collision avoidance is the one acting on the y-axis and thus governing lateral movements. It is assumed that the only task of the MPC acting on the x-axis is to track a reference velocity in order to minimize the error made assuming a constant speed during the prediction horizon. A reference for the y-coordinate is still needed. Since the coordinate system considered has its origin at the quadcopter's starting point and the x-axis is pointing towards the target, it is obvious that in order to reach the target the reference should be $y_T = 0$. In order to follow the latter the following weighting terms in the cost function were set: $w_y = 1$, $w_{\dot{y}} = 0.2$, $w_{\ddot{y}} = 0$ and $w_j = 0.001$. For the first trial an obstacle has been located 20 meters away from the starting position of the quadcopter. It has rectangular shape, is 5 meters wide and 4 meters long. For defining

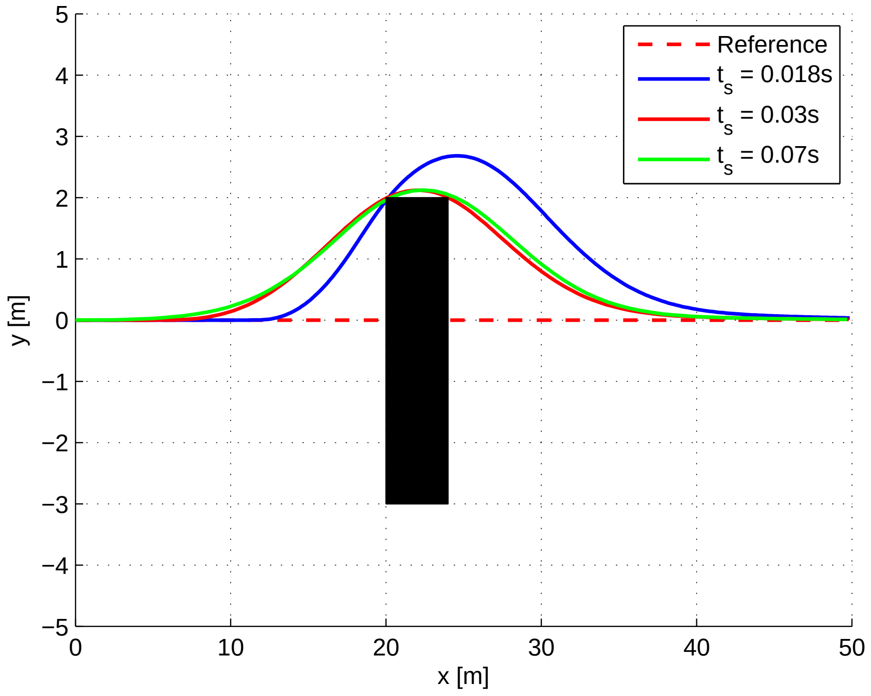

Figure 4.12: Obstacle avoidance - trajectories for various sampling times t_s

the predictive horizon $t_s = 30ms$ and $N = 50$ are set. The velocity along the x-axis is kept constant at $v_x = 10\frac{m}{s}$. Figure 4.11 shows the resulting trajectory. As can be seen, the decision whether to pass the obstacle to the left or to the right is correctly made by the mentioned algorithm, since passing to the left requires a smaller translation about the y-axis. Having avoided the collision, the trajectory starts again to track its reference. The great potentiality of this control strategy lies in the fact that by its nature it sets priorities while generating a trajectory. In fact the implementation of the obstacles as hard contraints bring the solver to avoid a collision in any case, even at the cost of leaving the reference and thus increasing the value of the objective function. Hence avoiding an object is more important than following the reference.

In order to understand the effect of the sampling time t_s and the prediction horizon N on the collision avoidance task various trials were made. Figure 4.12 depicts the trajectories for different values of t_s setting $N = 50$, whereas figure 4.13 shows the effect of the length of the predictive horizon, while $t_s = 30ms$. Comparing the two figures it can be seen that the two parameters have about the same effect on the trajectory. In fact, what really matters is the time window T_H covered by the prediction. As mentioned before

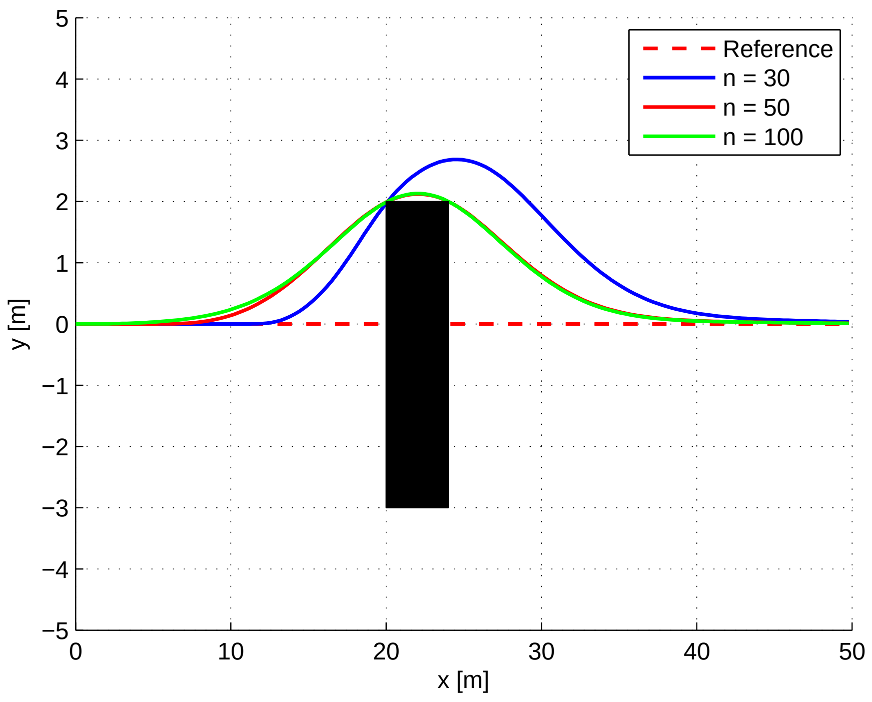

Figure 4.13: Obstacle avoidance - trajectories for various prediction horizons N

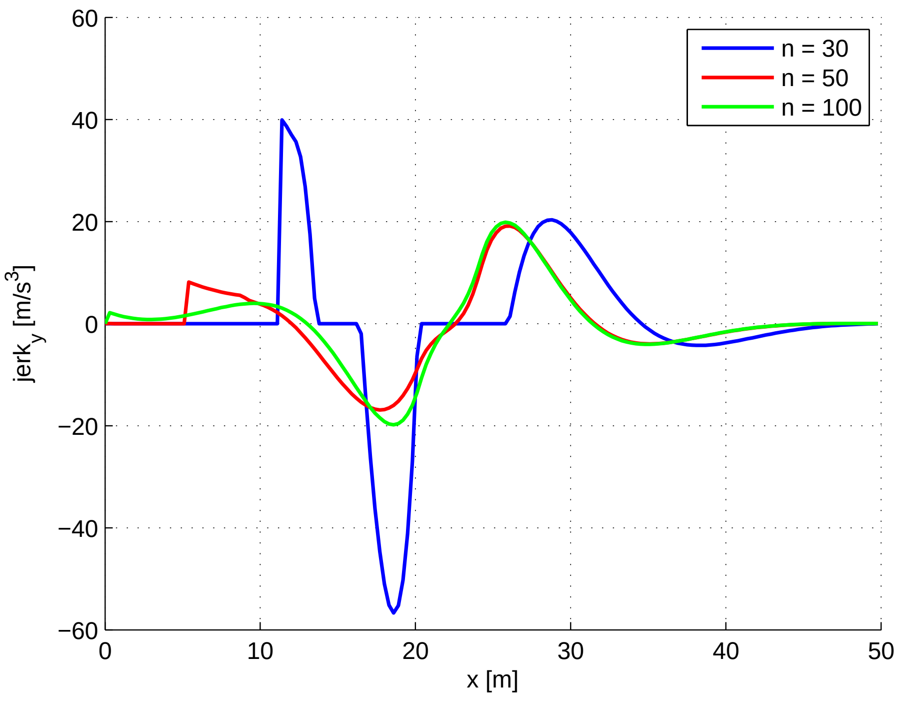

Figure 4.14: Obstacle avoidance - control actions for various prediction horizons N

the obstacle can only be seen by the MPC if contained in the considered horizon. This means that a longer prediction brings the quadcopter to react in advance to the presence of an object, whereas a shorter time window will see it later. For example for $N = 30$ and $t_s = 30ms$ the horizon is 0,9 seconds long, which means the trajectory planner reacts to the obstacle only 0,9 seconds before reaching it, that is 9 meters considering to fly at $10\frac{m}{s}$. Figure 4.14 shows the control action, namely the jerk on the y-axis for different values of N. As can be seen shorter prediction horizons bring the control to react later. Furthermore the input values will be greater, since the same translation must be provided in a shorter time. Smaller prediction times bring the trajectory to have bigger overshoots, because the velocity achieved by the quadcopter is greater due to the "harder" maneuver.

4.4 Different shapes and multiple obstacles

The method considered doesn't set any limitations on the obstacle's shape, it only needs to be discretized as mentioned in section 4.2. In order to comprehend the behavior of the control many trials with objects of different shapes were made. In figure 4.15 the results for a triangular obstacle are reported.

The control strategy is capable of dealing with multiple obstacles. Figure 4.16 illustrates the resulting path of the quadcopter avoiding four objects. The prediction horizon is set to be 1.5 seconds long with $t_s = 30ms$ and $N = 50$ and the quadcopter is flying with a speed of $v_x = 10\frac{m}{s}$. Note that for every trajectory solved, only the obstacles that are included in the time horizon can be considered. For instance the last object represented in figure 4.16 is only taken into account when the aircraft is at least 15 meters close to it. The control is also able to manage obstacles that are very close to each other, leading to small gaps in y-direction. In this case the upper object will set upper bounds, whereas the lower wider obstacle will set lower bounds on the y-coordinate. Both of them will be considered simultaneously leading to a convex flyable region (the small gap). Thus both the objects are correctly avoided.

It is interesting to notice the effect of the input weighting term w_j on the followed path. In figure 4.17 a set of trials with different values of the jerk weighting term are presented. It is shown that the value of w_j affects significantly the chosen path. Higher

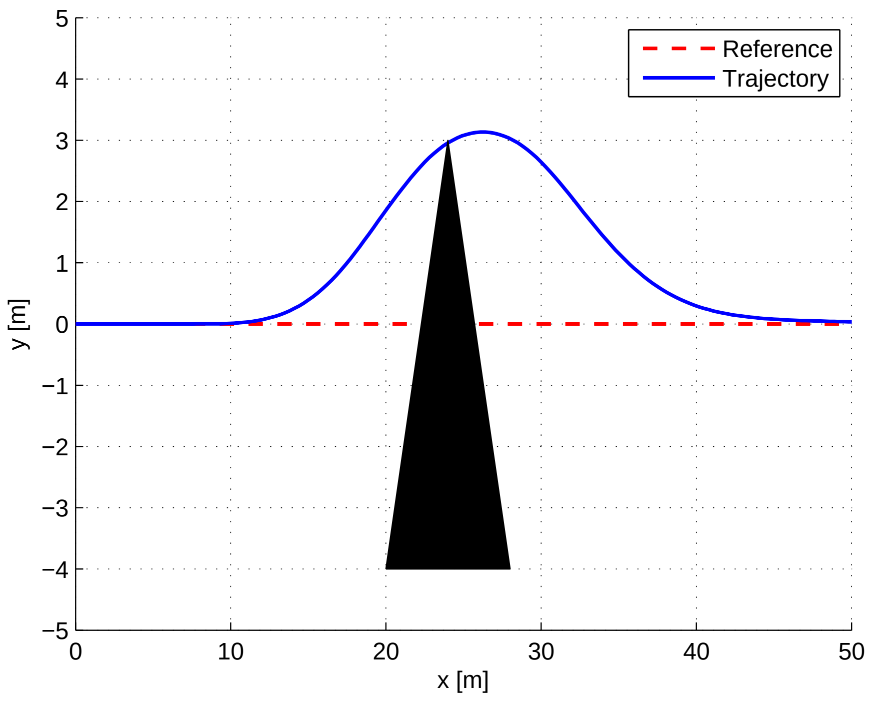

Figure 4.15: Triangular obstacle

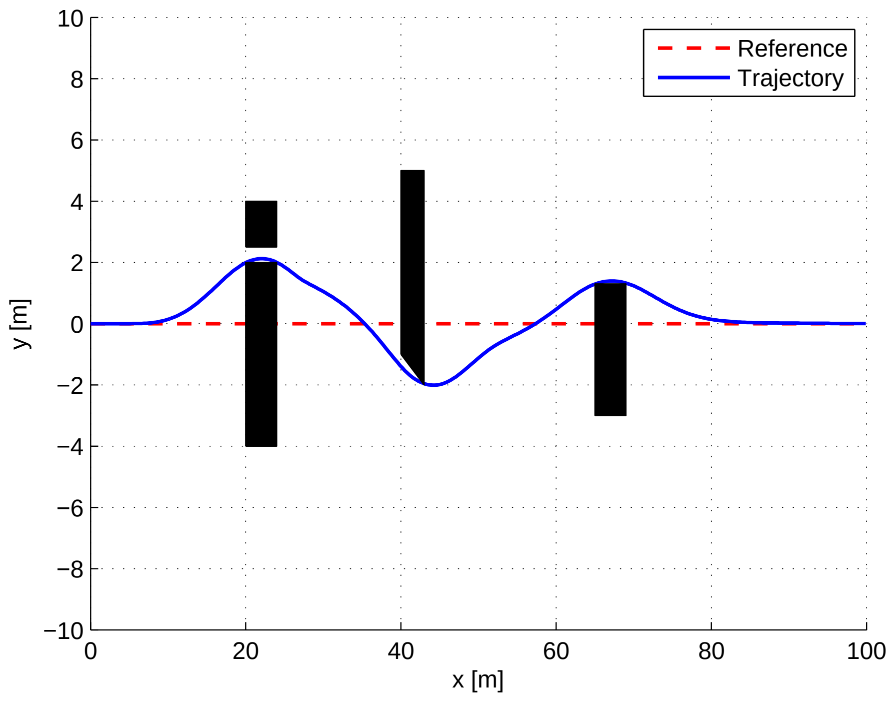

Figure 4.16: Multiple obstacles - trajectory

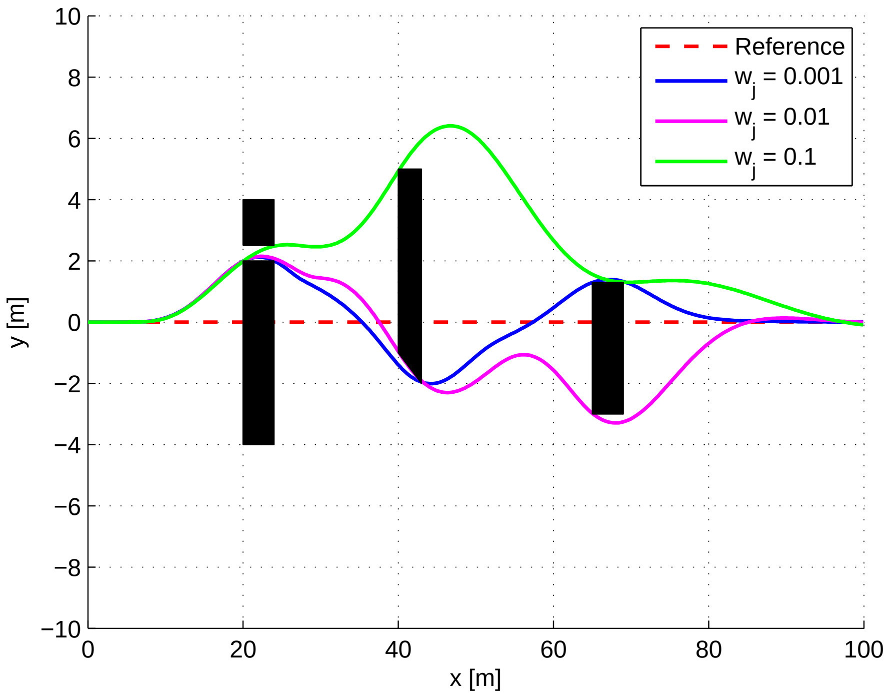

Figure 4.17: Multiple obstacles - followed paths for various jerk weighting terms w_j

values of w_j lead to smoother trajectories, since a variation of the acceleration is minimized by keeping the value of the control action as low as possible. It must not be forgotten, however, that the decision whether to pass the obstacle to the right or to the left is not taken by the optimization solver itself. Indeed it is taken every time a priori evaluating the shorter translation to be covered from the actual position in order to avoid the obstacle. Then, according to the choice made, lower bounds on the upper border or upper bounds on the lower border of the object are set. So, as illustrated in figure 4.17, the green trajectory passes the third obstacle to the left because, when detecting it, the quadcopter is closer to its left border. Note that after passing an obstacle the quadcopter starts again to approach its reference $y_T = 0$ until a new obstacle is detected by the prediction horizon. This behavior is well represented by the magenta trajectory in figure 4.17. Between the third and the last obstacle the quadcopter first tries to come closer to $y_T = 0$, since the last object is not included in its prediction horizon yet. As soon as it is closer than 15 meters it starts to change its trajectory in order to avoid it.

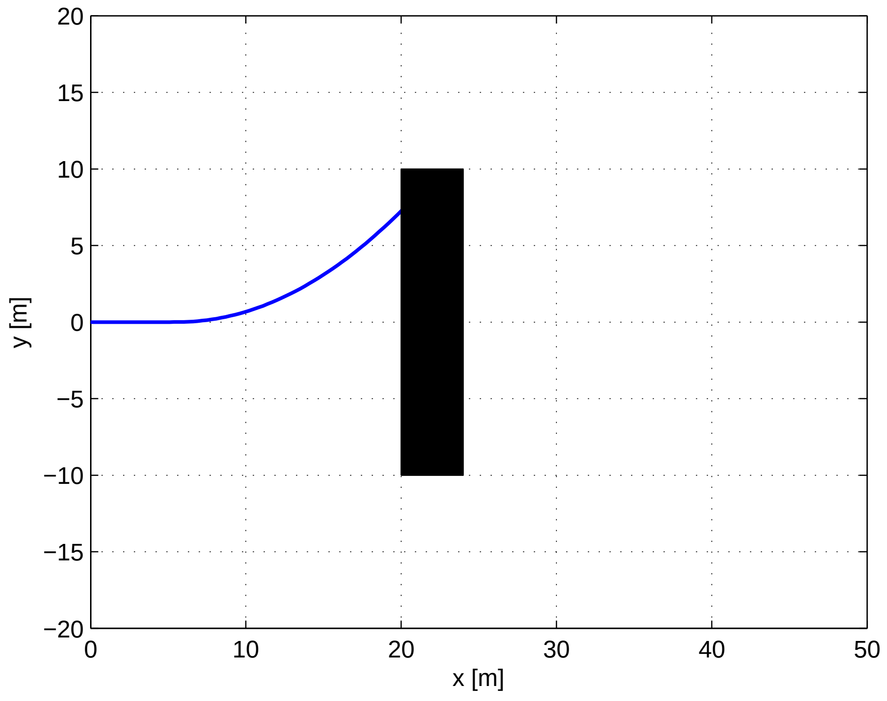

Figure 4.18: Wide obstacle

4.5 Problems

As specified in section 4.2 the obstacle is discretized in the time domain. This means that the bounds on the y-coordinate are set in the time and not in the space. It is worth to clarify this concept. It has been explained that bounding the y-coordinate as function of the x-coordinate brings to non-linear (due to the generic shape of the obstacle) and discontinuous (at the beginning and at the end of the object) inequalities that can't be considered using a linear MPC. It is therefore assumed, that the speed along the x-axis is constant during the predictive horizon and thus, simple bounds on the y-coordinate can be set for every time step. This is possible, because knowing a priori the speed of the quadcopter, its x-coordinate can be determined for each time step and hence the bounds to be set on the y-coordinate. This strategy implies two main problems. The first is related to the fact that considering a constant speed a strong hypothesis is made. In fact the longitudinal velocity v_x may vary during the flight. Thus the predicted trajectory could be different from the path that the quadcopter will follow. In reality, this issue is reduced by the fact that for every new trajectory the obstacle is re-discretized according

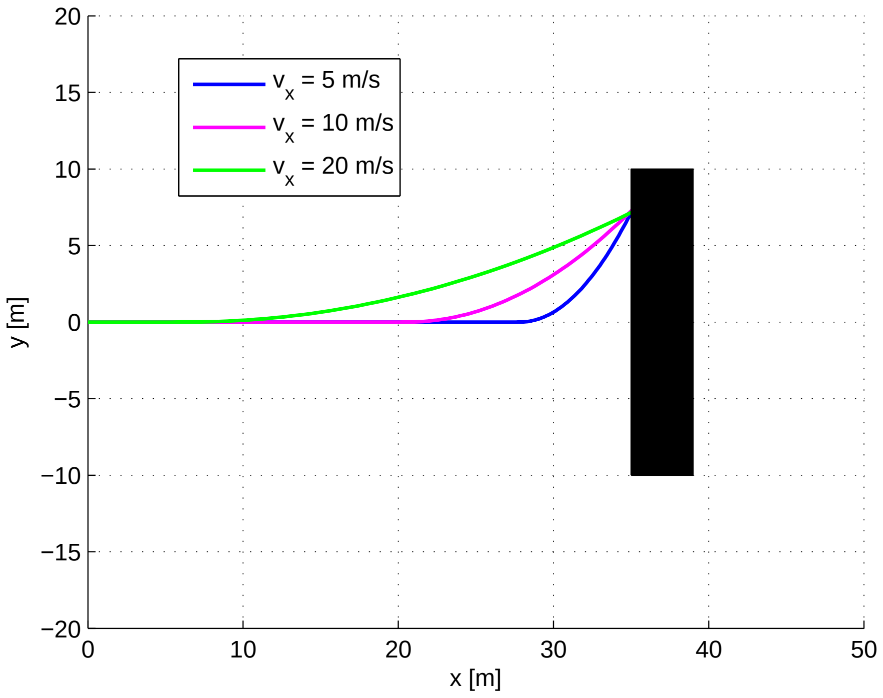

Figure 4.19: Wide obstacle - various v_x

to the new actual speed. The second main problem is that the trajectory along the x-axis is not affected from the presence of the obstacle at all. As previously stated, the trajectory generation task of x and y-axis can be decoupled, since the bounds on the two coordinates can't be related in a linear way. Hence only the MPC acting on the y-coordinate is carrying out the collision avoiding task. However there might be cases in which an action on the velocity along the x-coordinate is necessary in order to avoid the obstacle. For instance consider the object depicted in figure 4.18, which is 20 meters wide. As usual the MPC detects it 15 meters before approaching it. At this point the quadcopter has 1.5 seconds time to compute a translation of 10 meters about the y-axis in order to avoid the obstacle. This can't be achieved due to the physical limitations on jerk and acceleration seen in subsection 3.4.2. For example considering $t_s = 30ms$ and $N = 50$ and thus having a horizon of 1.5 seconds the maximal translation along an axis that can be achieved starting from rest is about 7.4 meters. Indeed, as depicted in figure 4.18 the quadcopter does everything possible to prevent a collision but doesn't manage to avoid it completely. Note that in this case, assumed to keep a constant speed, the ability to avoid a large obstacle doesn't depend on the velocity itself. In fact flying at different velocities

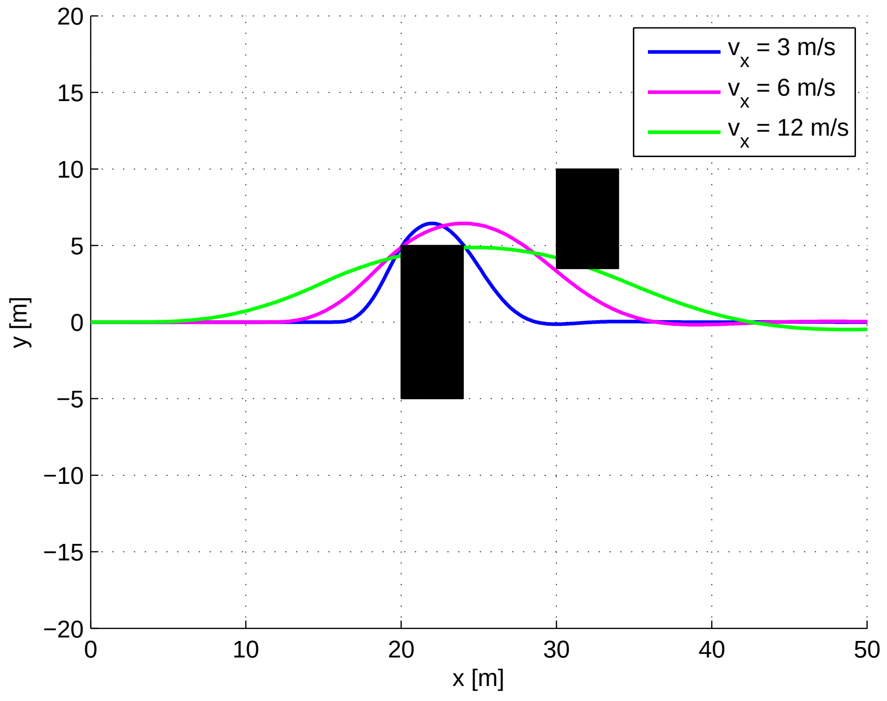

Figure 4.20: Two obstacles - various v_x

implies only that the obstacle will be seen at different distances, but the prediction horizon remains the same. This means that in terms of time the obstacle will be detected always 1.5 seconds in advance (keeping $t_s = 30ms$ and $N = 50$). Therefore the available time for the quadcopter to avoid the obstacle is always as long as T_H, independently from v_x. Figure 4.19 shows the results for different velocities v_x. As can be seen the maximal translation of 7.4 meters is reached for all three cases, so that though flying with a lower speed the collision can't be avoided.

The other situation the only control on the y-axis is not able to cope with, is when two or more obstacles are too close to each other and disposed as in figure 4.20. In this case the vehicle should move left to avoid the first obstacle and next it should reverse its direction in order to avoid the second one. Here the velocity v_x affects the ability to avoid a collision. In fact a lower speed brings the obstacles to be more distant from each other in terms of time. In other words, more time elapses from one obstacle to the other in case of lower speed. Thus the vehicle has more time to carry out its lateral maneuver. As illustrated in figure 4.20 with speeds up to $6\frac{m}{s}$ the obstacles can be correctly avoided. For $v_x = 12\frac{m}{s}$ the quadcopter crashes into the objects.

4.6 Control on the x-axis

4.6.1 Main strategy

In section 4.5 the problems that derive from controlling only the translations along the y-axis and keeping the velocity on the x-axis constant have been discussed. However, as mentioned before, the two coordinates cannot be managed within the same optimization process. This arises from the fact, that no linear relationship between the two coordinates can be found, that defines a correct boundary condition for the obstacle avoidance. Therefore the obstacles were discretized in the time domain assuming a constant velocity v_x during the predicted horizon. As explained, this will only affect the control on the y-axis leading to problems related to an infeasibilty of a valid trajectory as described in section 4.5.

The idea is to include a so-called "slack variable" that will soften the hard constraints on the y-coordinate in order to allow the optimization solver to keep on finding feasible solutions for the problem. In fact as hard constraints can't be satisfied (for example for too wide obstacles) the solver will return an infeasibility error and no solution is found. The "soft" constraints can be written as

$$y_k \leq ub_k + e \tag{4.20}$$

$$y_k \geq lb_k - e \tag{4.21}$$

where ub_k and lb_k are respectively the upper and lower bounds on the y-coordinate for each time step k. These bounds come from the discretization of the obstacle in the time domain. The slack variable e will be then strongly weighed in the cost function, therefore an increase of e will be heavily penalized. This means that wherever possible the slack variable will be kept around zero. As soon as the constraints on the y-coordinate cannot be satisfied e will increase its value. Hence, the slack variable becomes a manipulated variable of the optimization problem. The new objective function acting on the y-axis can be written as:

$$J = \frac{1}{2}\left(\sum_{k=1}^{N}(z_T - z_k)^T Q (z_T - z_k) + \sum_{k=0}^{N-1} w_j j_k^2 \quad + w_e e^2\right) \tag{4.22}$$

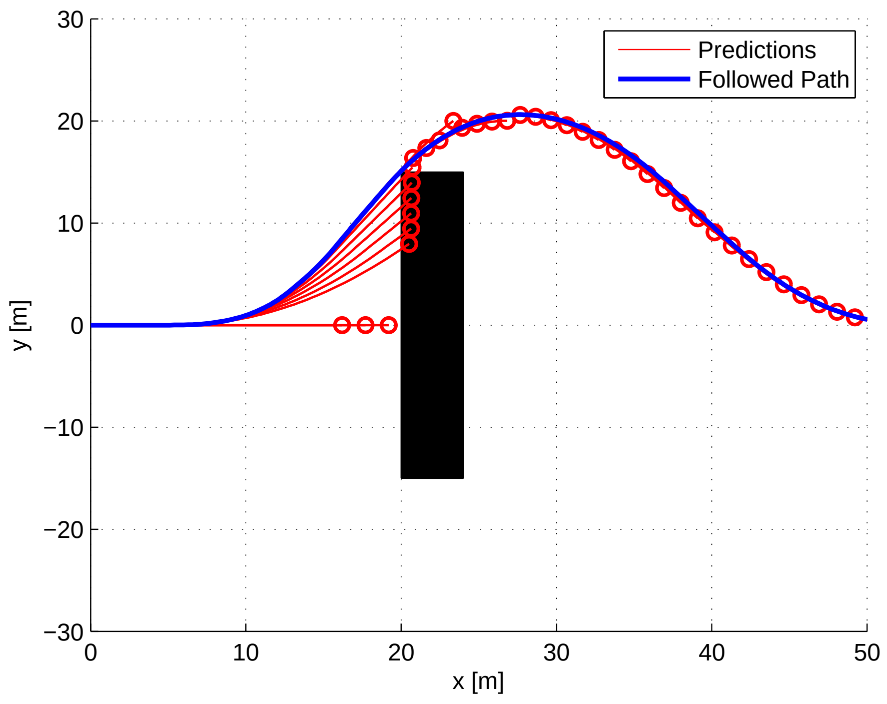

Figure 4.21: Wide Obstacle - trajectory and predictions - here also the longitudinal velocity is controlled

where the weighting term w_e assumes a very high value in order to soften the constraints only when strictly necessary. Note that using only one slack variable leads all of the constraints to become softer whenever even a single bound needs to be loosened. On the other hand the computational load won't be considerably affected, since only one variable is added to the optimization problem.

Having solved the problem of the trajectory infeasibility in case of obstacles that are too wide or too close to each other, the collision still needs to be avoided and this is only guaranteed if the velocity about the x-axis is decreased during flight. A way to do this is to exploit the information on the value of the slack variable in order to decelerate the quadcopter's longitudinal velocity v_x. In fact whenever a trajectory has been solved having $e > 0$, one or more constraints have been loosened. This means a trajectory about the y-axis has been found, which doesn't avoid the obstacle completely. So, an idea could be to reduce the velocity whenever a softening of the constraints is detected. In order to make sure that the collision is avoided we can think of decelerating the quadcopter with

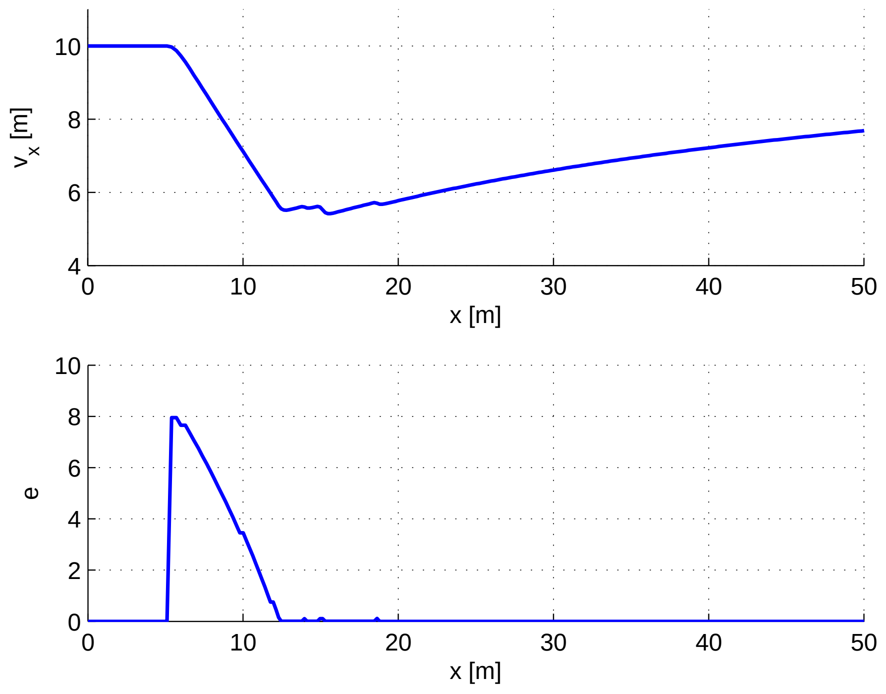

Figure 4.22: Wide Obstacle - the longitudinal velocity v_x depends on the value of the slack variable e

the maximum deceleration achievable, till a lateral trajectory that does not overstep the bounds is found. Note that the deceleration is meant to be along the x-axis.

Figure 4.21 represents the trial done with a 30 meters wide obstacle. In order to avoid it the quadcopter has to perform a translation of 15 meters along the y-axis. As known this is not possible within a time lapse of 1.5 seconds (prediction horizon). Hence the first trajectory evaluated when the obstacle is detected does not avoid the obstacle completely but only gains a translation of about 7 meters. This means the bounds on the last steps are overstepped by 8 meters, which is apparently the value of the slack variable (see figure 4.22). This information is sent to the longitudinal control which accordingly imposes a maximum deceleration. Since the model predictive controller is acting online as usual only the first control actions of the trajectories are applied. The control on the x-axis enforces a deceleration, while the control on the y-axis brings the quadcopter to raise its y-coordinate. After a time lapse equal to t_s a new trajectory is generated. The time discretization of the obstacle is done according to the new actual x-velocity. This leads the value of the slack variable e to decrease for every new generated trajectory, as the aircraft is increasing its y-coordinate while reducing its longitudinal speed (see figure 4.22). This brings the obstacle avoidance task to become more and more feasible, until an x-velocity is reached, for which the predicted lateral trajectory is capable of avoiding the obstacle completely. As can be seen in figure 4.22 as long as $e > 0$ the velocity v_x decreases with maximum deceleration.

4.6.2 Implementation in model predictive control

Now that the main principle of the longitudinal control has been explained, it is necessary to understand how the strategy is implemented into the model predictive controller. Remember that the control is carried out by two different MPCs. The one acting only on the y-coordinate and the other acting on the x-axis. The two controls communicate only through the information on the slack variable of the y-MPC.

The main requirement on the longitudinal control is to track a reference velocity (in this case $10\frac{m}{s}$) and to decelerate in case a loosening of the bounds on the lateral control is detected. This means that the errors on velocity and acceleration need to be weighed in the cost function in order to track the reference. The cost function of the longitudinal control can be written as:

$$J_x = \frac{1}{2}\left(\sum_{k=1}^{n}(z_R - z_k)^T Q (z_R - z_k) + \sum_{k=0}^{n-1} w_j j_k^2\right) \tag{4.23}$$

where the states-weighting matrix is given by:

$$Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & w_{\dot{x}} & 0 \\ 0 & 0 & w_{\ddot{x}} \end{bmatrix} \tag{4.24}$$

Note that the weighting on the position is not needed, since the x-axis is always directed towards a given target, which means given $v_x > 0$ the quadcopter is getting closer to it in any case. For what regards the reference state vector it is necessary to distinguish

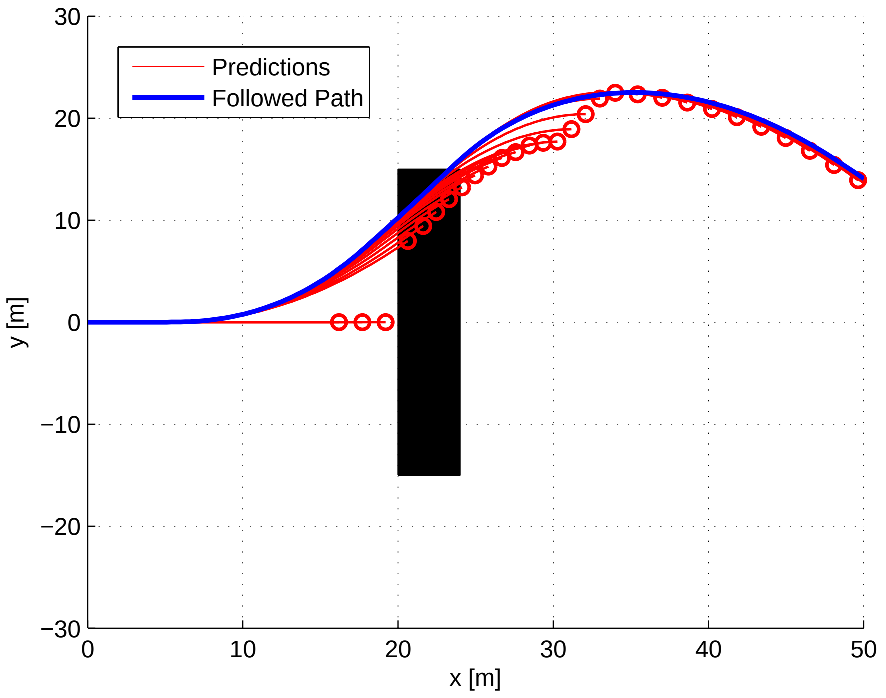

Figure 4.23: Wide Obstacle - the acceleration weighting term is chosen too low $w_{\ddot{x}} = 0.1$ and the trajcetory doesn't avoid the obstacle properly

between two cases:

$$z_R = \begin{bmatrix} 0 \\ 10\frac{m}{s} \\ 0\frac{m}{s^2} \end{bmatrix} \qquad for \quad e = 0$$

$$z_R = \begin{bmatrix} 0 \\ 10\frac{m}{s} \\ -7\frac{m}{s^2} \end{bmatrix} \qquad for \quad e > 0$$

This means that as long as the lateral control generates trajectories that don't overstep any boundaries, the longitudinal control will keep on referencing a $0\frac{m}{s^2}$ acceleration, maintaining therefore its reference speed. As soon as some bounds are overstepped ($e > 0$) a reference acceleration of $-7\frac{m}{s^2}$ is passed to the longitudinal optimization process in order to bring the quadcopter to decelerate. Note that e is the value of the slack variable of the optimization problem that has just been solved by the lateral control. Hence, seen

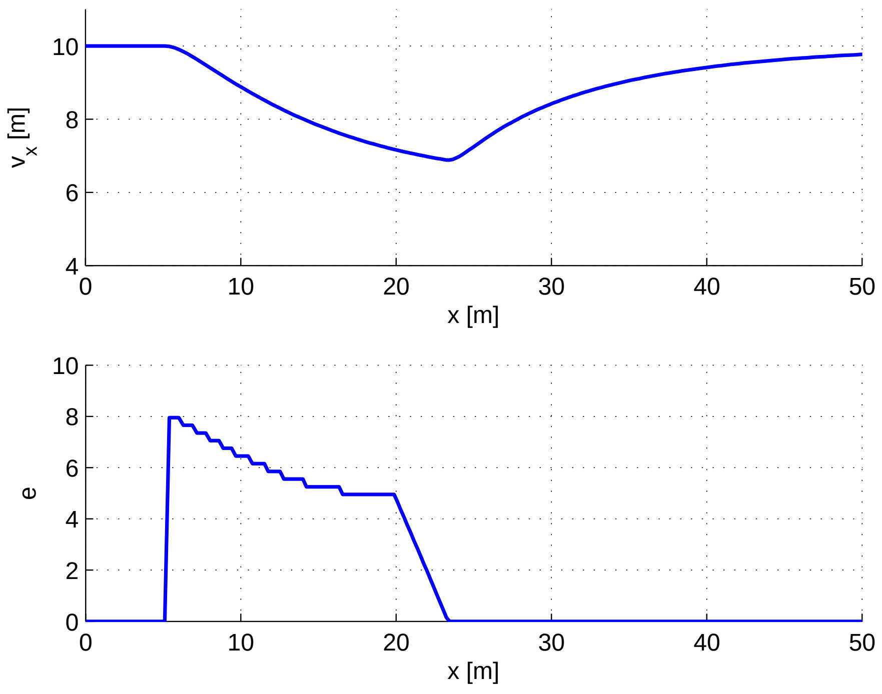

Figure 4.24: Wide Obstacle - $w_{\ddot{x}} = 0.1$ - longitudinal speed v_x and slack variable e

from the longitudinal control, it comes as an external parameter. Since different z_R can be imposed for every new optimization problem, they can depend on e. The acceleration weighting term $w_{\ddot{x}}$ has to be choosen greater than the velocity weighting term $w_{\dot{x}}$ since the deceleration task has a priority. Figure 4.23 and figure 4.24 show the results for a too low value of the acceleration weighting term with respect to the velocity weighting term. Here the choice was $w_{\ddot{x}} = 0.1$ and $w_{\dot{x}} = 0.1$. As can be seen the quadcopter doesn't decelerate properly, which leads to a collision. Good performances have been proved by choosing $w_{\ddot{x}} = 1$ and $w_{\dot{x}} = 0.1$.

To test the ability of the control to cope with narrow gaps a second obstacle was placed right after the first one as depicted in figure 4.25. First, the model predictive controller generates trajectories in order to avoid only the first object, since the second one doesn't appear in the prediction horizon yet. Here a longitudinal deceleration is necessary. As the y-MPC also detects the second obstacle it first tries to generate a trajectory that avoids it. The longitudinal velocity, however, is still too high, hence some boundaries will be overstepped. The information $e > 0$ is again passed to the longitudinal control, which reduces the speed a second time. Finally, also the second obstacle is correctly avoided.

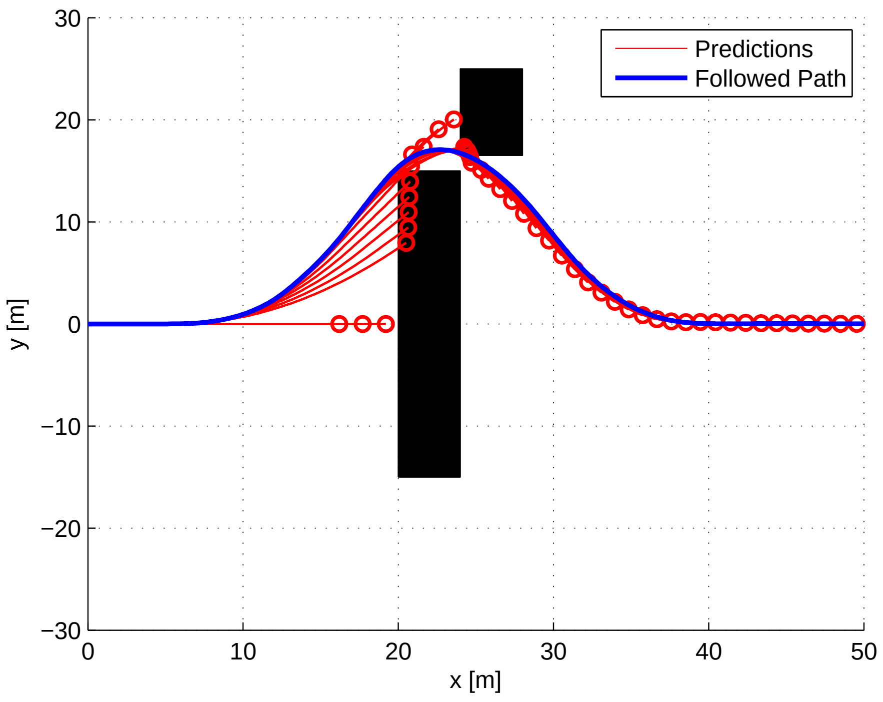

Figure 4.25: Two Obstacles - longitudinal velocity is controlled

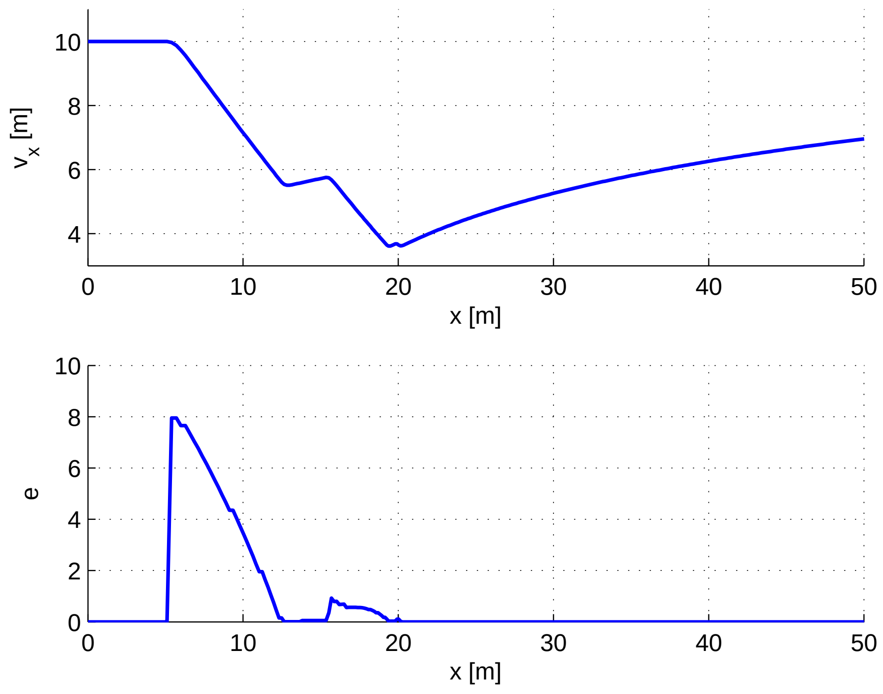

Figure 4.26: Two Obstacles - the longitudinal velocity depends on the slack variable

Some considerations have to be made on the proposed control strategy. As long as the quadcopter is moving with a constant speed the predictions done during the trajectory generation task are quite reliable. However, when an obstacle is detected for which a reduction of the longitudinal speed is necessary, the trajectories generated during the deceleration begin to be only suboptimal. In fact the lateral trajectories are solved relying on the assumption of constant longitudinal speed during the considered horizon, which does not reflect reality in case of deceleration along the x-axis. However, since the obstacles are re-discretized every time step according to the actual speed, course adjustments are made. Although this strong approximation seems to be only a drawback, it can be proved that it also implies great advantages. For instance, think of the 30 meters wide obstacle described in subsection 4.6.1. The control starts to consider the object when it is included in its time horizon, which, having $T_H = 1.5s$ and flying at a speed of $10\frac{m}{s}$, means detecting it 15 meters in advance. In reality, it can be shown that 2.3 seconds elapse from the moment the obstacle is seen to the moment it is reached, which is a longer time than the prediction horizon. This is caused by the fact that the quadcopter is decelerating, whereas the lateral control supposes to travel at constant speed and thus to reach the obstacle in a shorter time. This means the control acts as a sort of prediction time extender. Hence also collision-avoiding maneuvers that may last more than the prediction horizon can be easily performed.

Note that this is provided only because the discretization of the obstacle is done assuming to move at constant longitudinal speed. One could have actually thought of discretizing the obstacle according to the forecasted deceleration on the longitudinal axis. Although, at first sight, this could be interpreted as a better way to discretize the obstacle, since representing better the future longitudinal positions of the vehicle, it would have caused few problems. Considering again the 30 meters wide obstacle, as usual it is detected 1.5 seconds in advance. Since a deceleration is necessary in order to avoid it, the longitudinal control predicts a trajectory that tracks a maximum deceleration for the whole horizon. If the obstacle is descretized according to the forecasted deceleration, it is obvious that it is no more 1.5 seconds away from the vehicle, but much more. Indeed, if the speed is low enough to be able to stop the vehicle completely before the obstacle is reached, the object would be ∞ seconds away from the quadcopter in terms of time.

Hence, the obstacle would no longer be seen. Accordingly the vehicle would start again to accelerate till the obstacle is once again included in the prediction horizon. Also this time a deceleration would be needed, hence the obstacle would be again discretized according to a decreasing velocity and thus no more included in the horizon and so on. Therefore, it would cause an instable situation and a collision avoidance could not be guaranteed any more.

4.6.3 Maximum allowable speed

In this subsection some considerations are made on the maximum speed that should be allowed in order to avoid collisions. Two situations are considered. The first one is the case for which it is decided that the maximum velocity should be low enough to guarantee in every moment the possibility to stop the vehicle within the prediction horizon. This means that in every moment the vehicle should be able to impose a zero velocity for the last step of its prediction. Assuming a constant acceleration a the following equation can be written for the speed:

$$v = v_0 + at \tag{4.25}$$

It is assumed that in case of emergency the maximum achievable deceleration a_{min} is applied. Thus, assuming to travel at $v = v_{max}$, the velocity should be dropped down to zero within the time horizon T_H. The limits on the jerk are neglected, hence it is assumed that the maximum deceleration can be applied instantaneously. Imposing the speed to be zero after a time equal to T_H yields:

$$0 = v_{max} + a_{min}T_H \tag{4.26}$$

By rearranging the terms the maximum allowable speed can be found in relation to the prediction horizon:

$$v_{max} = -a_{min}T_H \tag{4.27}$$

For instance, choosing $N = 50$ and $t_s = 30ms$, thus having a prediction time of $T_H = 1.5s$, a maximum speed of $v_{max} = 10.5\frac{m}{s}$ is obtained, considered to have $a_{min} = -7\frac{m}{s^2}$

However this maximum velocity is set in a conservative way. Consider, for instance, the worse case in which an obstacle is detected, that can't be passed at all, since it is too large and there is no way out. Hence, the only way to avoid a collision is to decelerate for the whole distance until the obstacle is reached. As mentioned in previous sections the obstruction is detected by MPC as soon as it can be considered within the prediction horizon, hence it is can be only seen T_H seconds in advance. Assuming to travel at a certain speed v_0, one can calculate the distance x_{obst} at which the obstacle is first detected.

$$x_{obst} = v_0 T_H \tag{4.28}$$

The minimum time Δt needed for the vehicle to stop completely can be evaluated by:

$$\Delta t = -\frac{v_0}{a_{min}} \tag{4.29}$$

The distance covered during a deceleration is computed by:

$$x = v_0 \Delta t + \frac{1}{2} a_{min} \Delta t^2 \tag{4.30}$$

Substituting (4.29) in (4.30) the distance covered during a complete arrest of the vehicle traveling at an initial speed of v_0 is found. Imposing this distance to be equal to x_{obst}, namely the one that elapses between vehicle an obstacle in the moment when the latter is detected, yields:

$$v_0 T_H = v_0 \left(-\frac{v_0}{a_{min}} \right) + \frac{1}{2} a_{min} \left(-\frac{v_0}{a_{min}} \right)^2 \tag{4.31}$$

Rearranging the terms the maximum speed can be found, which guarantees a collision avoidance in the worse case in which the obstacle can't be passed at all.

$$v_{max} = -2 a_{min} T_H \tag{4.32}$$

Thus, having for example $T_H = 1.5$ and $a_{min} = -7\frac{m}{s^2}$, a maximum allowable speed of $21\frac{m}{s}$ is achieved.

4.7 Obstacle avoidance in three dimensional space

The proposed strategy for obstacle avoidance can be extended to the three dimensional case. A third MPC is added to control the translations about the vertical axis of the inertial frame. This can be controlled in the same way as the y-coordinate. The obstacle's upper and lower sides are discretized in the time domain with the assumption of constant longitudinal speed during the prediction horizon. Whether lower or upper bounds are imposed, which means whether the quadcopter should pass above or below the obstacle, is evaluated before each trajectory is generated. For this task one can employ the algorithm that best suits the needs. Note that in most cases the boundary conditions on the vertical coordinate will be lower bounds, since the quadcopter will have to pass over objects, for example trees or walls. It could however be necessary to also impose upper bounds, e.g. for passing under a bridge. There are many cases that could be considered. For example passing through a window requires the imposition of lower and upper bounds on the vertical coordinate as well as on the lateral coordinate, in order to obtain a rectangular feasible region.

An other important consideration that has to be done is that due to the same arguments discussed in the previous sections, it is quite difficult to linearly relate the y and z coordinate in order to describe the obstacle in a correct way. This is the reason why also y and z axis have been decoupled. Seen from a global point of view, the flyable regions will be rectangular frames (in y-z plane), one for each step of the horizon. This comes from the fact that y-MPC and z-MPC can both impose lower and upper bounds for each time-step of their prediction windows.

4.7.1 Implementation

In this subsection trials of the control acting in a three-dimensional environment are presented. In order to impose upper and lower bounds on the vertical and lateral optimization problems a simple algorithm was exploited. The latter evaluates the distances of the actual position with respect to the lower and upper vertices of the object along the respective axes. According to the shortest way that may be traveled in order to avoid the obstacle, the algorithm returns upper/lower bounds on the z-coordinate or upper/lower bounds on the y-coordinate. In other words if the obstacle is wider than it is tall the control imposes lower bounds on the z-coordinate and no bounds on the y-coordinate. Thus the quadcopter will pass above. If it is taller than it is wide it will pass to the left or to the right accoring to the shortest way, hence only lateral bounds will be set.

Note that the vertical control also has to track a reference that is evaluated in the

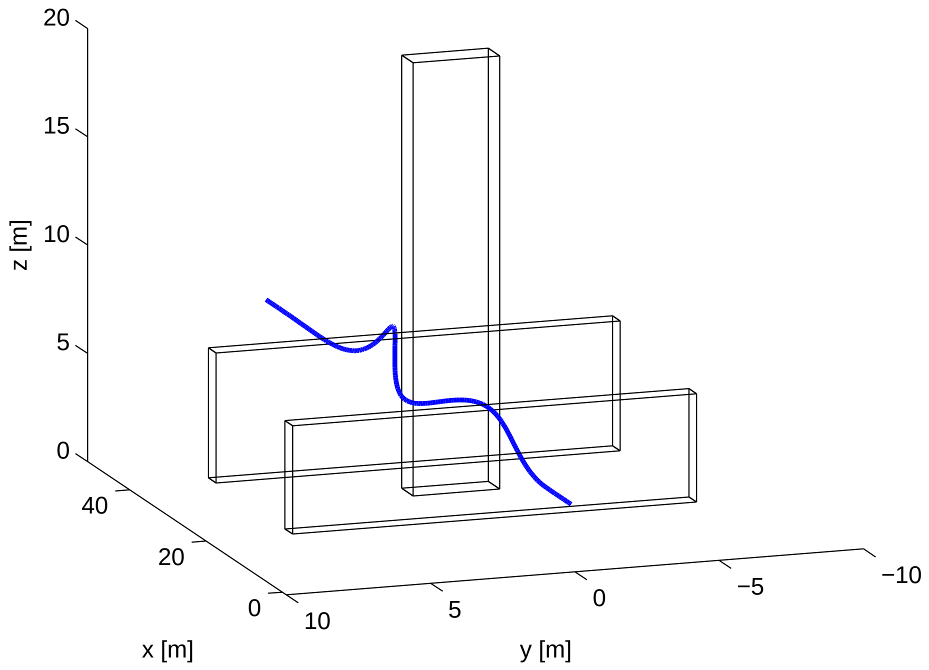

Figure 4.27: Three-dimensional environment - three obstacles

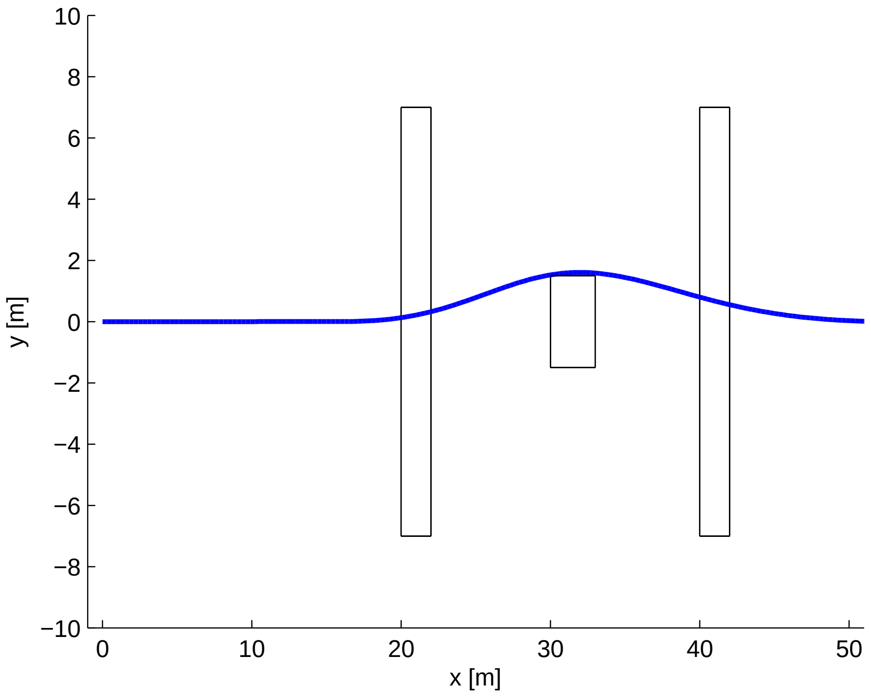

Figure 4.28: Three obstacles - three-dimensional environment - x-y plane

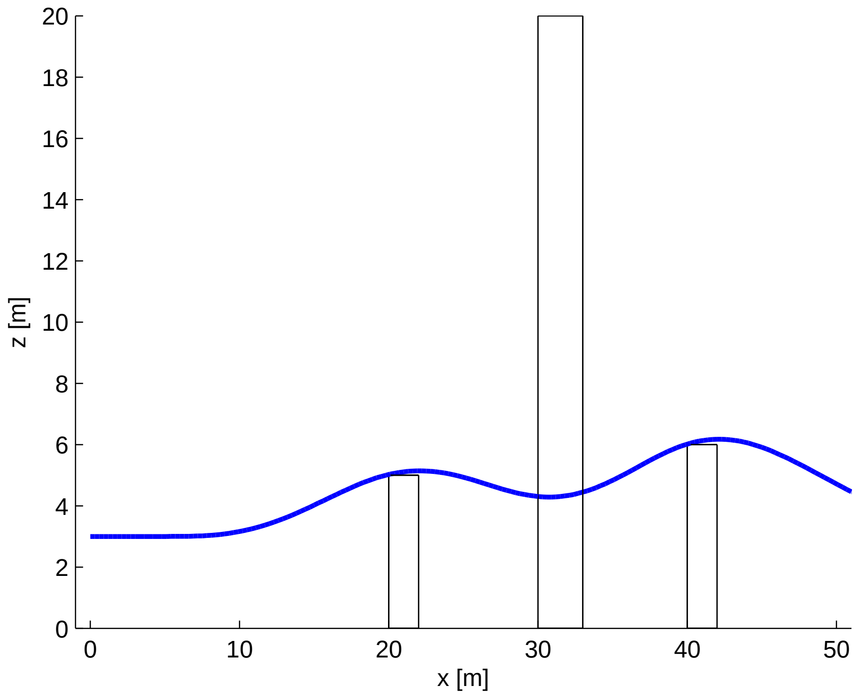

Figure 4.29: Three obstacles - three-dimensional environment - x-z plane

cost function. This will be the desired altitude at which the quadcopter has to fly. The simulation presented in figure 4.27 shows the avoidance of three consecutive obstacles in a three-dimensional environment. The quadcopter is flying 3 meters above the ground. The first obstacle is 14 meters wide, has a height of 5 meters and is 2 meters long. The second is 3 meters wide, 3 meters long and 20 meters tall, the third is similar to the first but 6 meters tall. In a real world situation the two wide obstacles could represent two walls, whereas the second one could be a column or a light pole. The quadcopter correctly avoids the objects. Figure 4.28 and figure 4.29 illustrate the trajectory respectively in the x-y and in the x-z plane.

The considerations made in section 4.6 can be exploited also for the three-dimensional case. This means it is still possible to control the longitudinal speed in case of predicted trajectories that overstep some boundary conditions and hence crash into obstacles. Here two different slack variables will be used. The first softens the position constraints of the lateral optimization problem, whereas the second is used by the MPC acting on the vertical axis. The MPC governing the longitudinal flight will decelerate the vehicle as soon as one slack variable assumes a value greater than zero. Remember that the slack variable

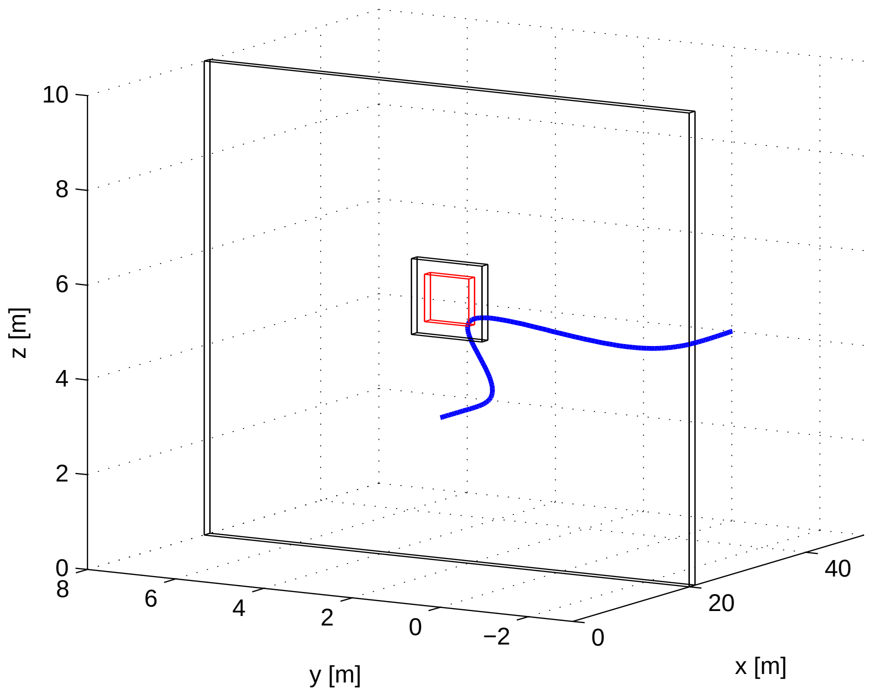

Figure 4.30: Window - quadcotper's trajectory passing through a window in a wall

performs two main tasks. The first task is to allow the optimization solver to find feasible solutions also in case an overstepping of the position-bounds is inevitable according to the actual speed v_x. The second task is to pass this information to the longitudinal control, which accrodingly decreases the velocity.

4.7.2 Quadcopter size

Until now the quadcopter was treated as a punctual mass. Hence the trajectories were allowed to touch the contours of the obstacles, since this was still interpreted as a feasible solution. In reality, this can't be permitted, since the quadcopter has its own size. This means that the dimensions of the quadcopter's frame and its propellers have to be considered when imposing boundary conditions on positions. The easiest way to cope with this problem is to simply reduce the flyable region by the quadcopters dimensions. For instance, the bounds on the y-coordinate should be reduced by half of the quadcopter's width, since the trajectory refers to the vehicle's midpoint (which in most cases corresponds to its center of gravity). Actually, another term should be added in order

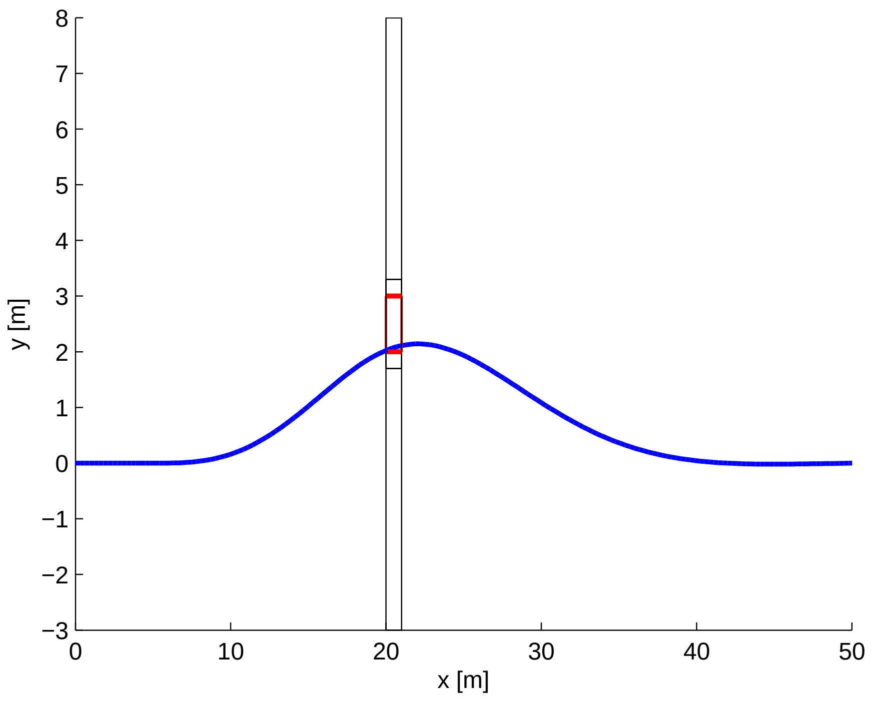

Figure 4.31: Quadcotper's trajectory passing through an aperture in a wall - x-y plane

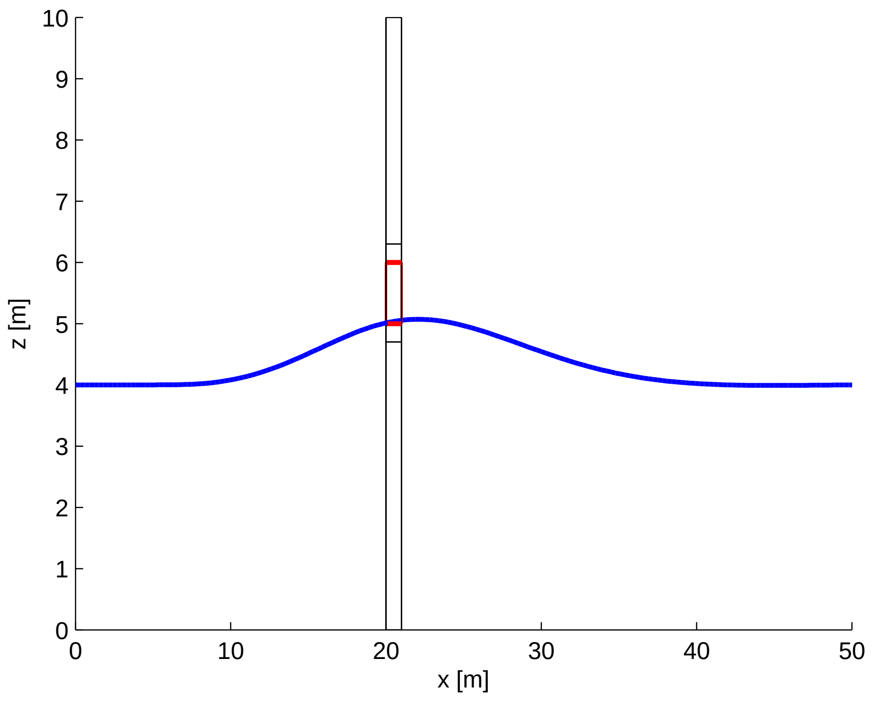

Figure 4.32: Quadcotper's trajectory passing through an aperture in a wall - x-z plane

to consider a safety margin. In fact, errors could arise for example from the position estimation made by the measurement systems (for the quadcopter's position as well as for the obstacles). The safety term should also consider the fact that, though the slack variable is hugely weighed into the cost function, it will always assume a very small value even in case there's no need for a relaxation of the boundary conditions. This could bring the trajectories to slightly cut the edges of the obstacles. Note that, since the obstacles are represented as bounds, the trajectories are allowed to pass very close to them. Hence, the safety term must also include a minimum distance to the obstacle that should be provided. Therefore the bounds on the y-coordinate can be written as:

$$y_k \leq ub_k + e - l_w - l_s \tag{4.33}$$

$$y_k \geq lb_k - e + l_w + l_s \tag{4.34}$$

where ub_k and lb_k are respectively upper and lower bounds deriving from time-discretization of the obstacle. e is the slack variable, l_w is the term considering the quadcopter's size and l_s is the safety term. The bounds on the z-coordinate can be written in the same way.

In figure 4.30 a wall with a rectangular window is depicted. The control has to generate trajectories in order to pass through the opening, which in a real world situation could represent a window. As soon as the aperture is detected lower and upper bounds on the y- and z-coordinate are imposed in correspondance to the horizon steps covered by the window itself (according to the actual speed). This time the quadcopter's size and the safety margin are taken into account and the boundary conditions are imposed as described in (4.33) - (4.34). Figure 4.30 shows the resulting path. The black frame represents the actual window, whereas the red one represents the bounds that are set on the positions. As shown in figure, the trajectory correctly passes through the red frame, thus avoiding a collision with the wall.

4.8 Avoidance of moving obstacles

In previous sections trajectories were planned assuming fixed obstacles. In reality, also moving objects, such as other vehicles or persons, may present on the way. For this reason it is necessary to develope a control capable of avoiding them.

4.8.1 Main strategy

Since MPC relies on future predictions made about the system behaviour, it is obvious that in order to set bounds at future time steps a forecast of the obstacle's movements is needed. Though it may seem like a difficult request, it is normal to make assumptions about future behaviours of other agents. For example vehicles are generally expected to travel at constant speed. Moreover, model predictive control generates trajectories every few milliseconds, which means for every optimization problem a new prediction about the obstacle's movements can be made. This can adjust the course of the quadcopter in case of unexpected changes in direction and velocity of the moving obstacle.

For the first trial a rectangular obstacle is chosen. It is 7 meters long and 5 meters wide. It is assumed that it is traveling at a constant speed of $v_{x_{obst}} = 3\frac{m}{s}$, $v_{y_{obst}} = 2\frac{m}{s}$, with $v_{x_{obst}}$ longitudinal velocity and $v_{y_{obst}}$ lateral velocity. The quadcopter is flying at $10\frac{m}{s}$ and the horizon parameters are set to $t_s = 30ms$ and $N = 50$. Figures 4.33 and 4.34 show the resulting trajectory of the aircraft captured at various time steps. Note that the graphical representation of the moving obstacle and the path followed by the quadcopter can be deceptive. For instance looking at figure 4.33 the trajectory seems to pass across the object. It is actually a misleading optical illusion caused by the attempt to plot a dynamical environment through static frames. It has to be taken into account that the plotted trajectory is representing past positions of the vehicle. Thus, it doesn't matter if the obstacle crosses the path that has just been traveled by the quadcopter. Only the actual position of the vehicle (represented by the small circle) with respect to the current position of the obstacle has to be observed in order to verify whether the quadcopter is avoding the object. Moreover it has to be rememberd that the vehicle is considered as a punctual mass. Hence, though the figures depict it as a circle, it only has to be verified that the center of the circle is not passing through the object.

As mentioned in subsection 4.7.2, for real situations the bounds will be increased in order to take into account the dimensions of the quadcopter together with a safety term that sets a minimum distance to stay away from the obstacle.

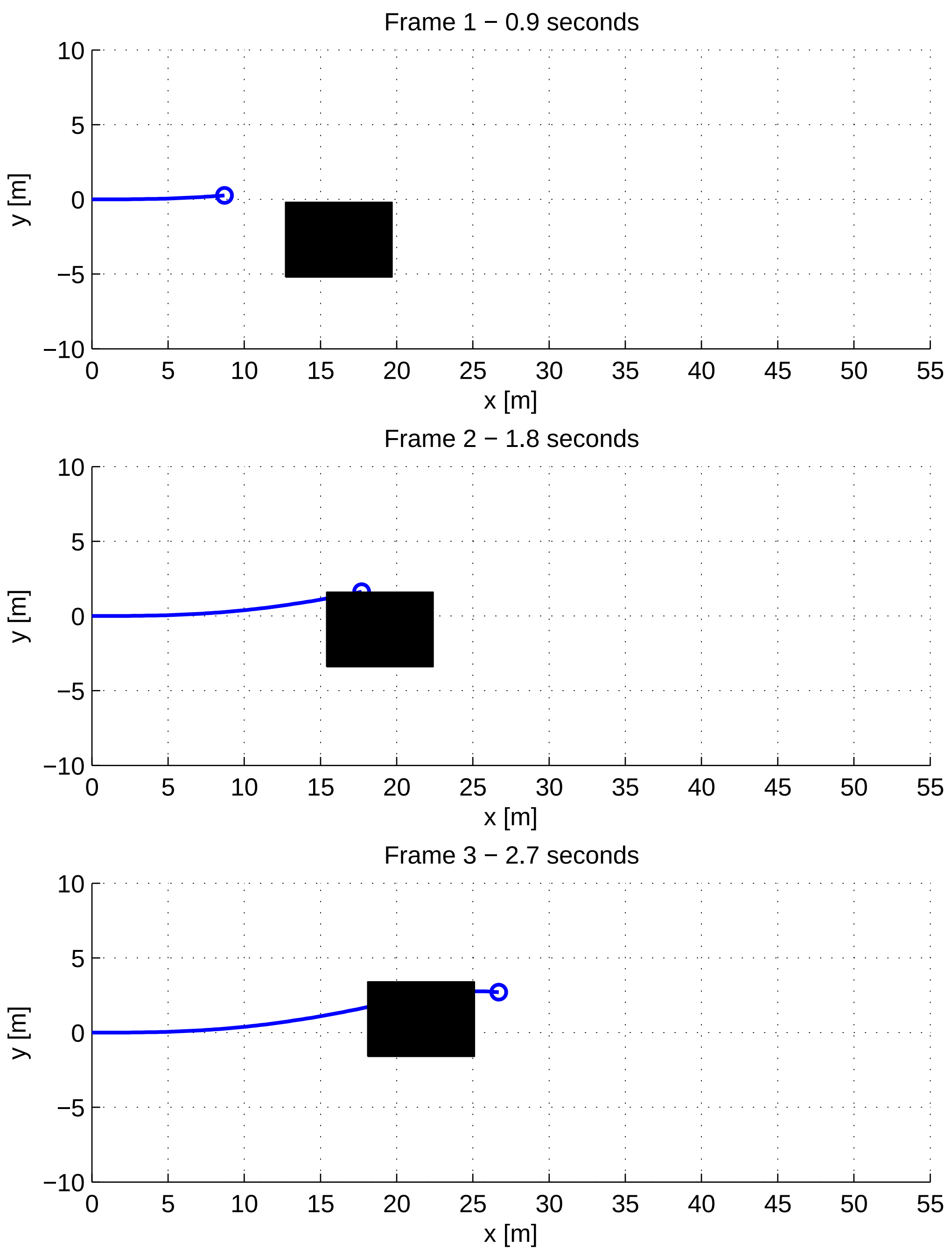

Figure 4.33: Trajectory avoiding moving obstacle - Frames 1-3

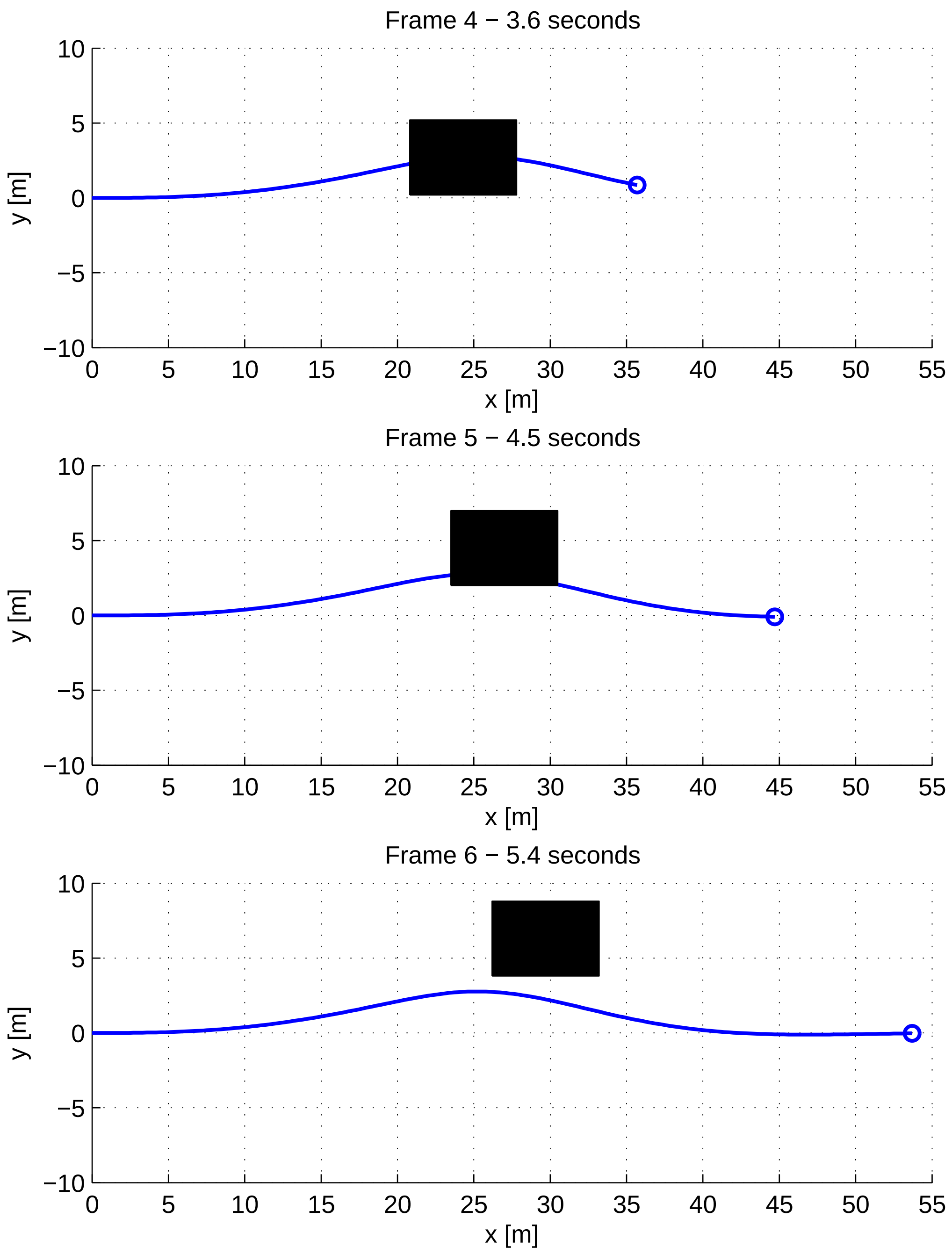

Figure 4.34: Trajectory avoiding moving obstacle - Frames 4-6

4.8.2 Time-discretization of the moving obstacle

As usual, the obstacle is discretized in the time domain in order to be able to set boundary conditions on the y-coordinate for each time step. This time, setting bounds for the obstructing object requires a more challenging consideration. While the discretization of a fixed obstacle is only depending on the quadcopter's velocity, the discretization of a moving one is determined also by the lateral and longitudinal speed of the obstacle itself. For instance, if the moving object was in front of the quadcopter, traveling with a higher longitudinal velocity, no bounds would be required at all, since the aircraft is never reaching the obstacle.

Given a generic obstacle its upper and lower sides can be described by two generic functions that define its shape.

$$y_u = f_u(x) \tag{4.35}$$

$$y_l = f_l(x) \tag{4.36}$$

where y_u and y_l are respectively y-coordinate of upper and lower side of the object described as functions of the x-coordinate. In order to be able to use f_u and f_l to define bounds they should assume the following generic form:

$$f_u(x) = \begin{cases} -\infty, & if \quad x < 0 \\ f_u^*, & if \quad 0 \le x \le l \\ -\infty, & if \quad x > l \end{cases} \tag{4.37}$$

$$f_l(x) = \begin{cases} +\infty, & if \quad x < 0 \\ f_l^*, & if \quad 0 \le x \le l \\ +\infty, & if \quad x > l \end{cases} \tag{4.38}$$

where f_u^* and f_l^* define the shapes of upper and lower side and l is the longitudinal length of the object.

Since the obstacle is moving, its position and thus the functions defining its upper and lower shapes will also depend on time in the following way. For simplicity, the equations are written only once, since the same considerations are valid for both upper and lower sides of the moving object.

$$y_{u/l} = f_{u/l}\left(x - x0_{obst} - v_{x_{obst}}t\right) + y0_{obst} + v_{y_{obst}}t \tag{4.39}$$

where $v_{x_{obst}}$ and $v_{y_{obst}}$ are respectively the longitudinal and lateral components of the obstacle's velocity, which is assumed to be constant over time. $x0_{obst}$ and $y0_{obst}$ define the initial position of the object (for $t = t_0$). The functions described in (4.39) represent a translation of the functions defined respectively in (4.35) and (4.36) that depends on time t.

In order to set bounds on the y-coordinate of the quadcopter it will be again assumed that it is moving with constant speed. Thus, its longitudinal position can be written as:

$$x = x_0 + v_x t \tag{4.40}$$

with x_0 initial position of the quadcopter and v_x longitudinal speed. By substituting (4.40) in (4.39) and rearranging the terms the following equation is obtained:

$$y_{u/l} = f_{u/l}\left(x_0 - x0_{obst} + \left(v_x - v_{x_{obst}}\right)t\right) + y0_{obst} + v_{y_{obst}}t \tag{4.41}$$

The two equations define the shape of the obstacle in the time domain, from the quadcopter's point of view. They can actually be used for defining the bounds on the y-coordinate, since they only depend on the time. What still needs to be done is to discretize the function with stepsize equal to t_s, since the bounds have to be imposed for each time step k.

$$t = t_0 + k \cdot t_s \tag{4.42}$$

By substituting (4.42) in (4.41) one can evaluate the bounds according to the considered time step k.

$$y_{u/l}\{k\} = f_{u/l}\left(x_0 - x0_{obst} + \left(v_x - v_{x_{obst}}\right)kt_s\right) + y0_{obst} + v_{y_{obst}}kt_s \tag{4.43}$$

For every new trajectory the actual time is set to be $t_0 = 0$. This means x_0, y_0, $x0_{obst}$ and $y0_{obst}$ are the coordinates of quadcopter and obstacle measured at the time when the trajectory is solved.

Again an algorithm can be exploited, which decides whether to impose upper or lower bounds according to the shortest y translation that may be performed in order to avoid the obstacle. Thus:

$$y_k \leq y_l\{k\} \qquad if \quad y_0 - \min(y_l) \leq \max(y_u) - y_0 \tag{4.44}$$

$$y_k \geq y_u\{k\} \qquad if \quad y_0 - \min(y_l) > \max(y_u) - y_0 \tag{4.45}$$

Note that by rearranging (4.40) it is possible to evaluate the time for which the quadcopter is reaching a certain x-coordinate.

$$t = \frac{x - x_0}{v_x} \tag{4.46}$$

Substituting (4.46) in (4.41) upper and lower bounds on the y-coordinate can be represented in the spatial domain.

$$y_{u/l} = f_{u/l}\left(x\left(1 - \frac{v_{x_{obst}}}{v_x}\right) - x_{0_{obst}} + \frac{v_{x_{obst}}}{v_x}x_0\right) + y_{0_{obst}} + \frac{v_{y_{obst}}}{v_x}(x - x_0) \tag{4.47}$$

This means that from the quadcopter's point of view, traveling with constant longitudinal speed, the moving obstacle can be seen as a virtual fixed object. If both vehicle and obstacle are traveling at constant velocities, the boundary conditions on the vehicle's y-coordinate can be set a priori for every future time step. Hence, assuming that the longitudinal speeds do not vary over time, the bounds are fixed in time and thus in space. Therefore, every moving obstacle involves a fictive fixed object representing the bounds that has to be avoided. The latter can be treated exactly in the same way as for fixed obstacles.

4.8.3 Implementation of the moving obstacle

Figures 4.35 and 4.36 show the same trial as the one depicted in subsection 4.8.1. The red dashed frame represents the bounds on the position seen from the quadcopter's point of view as described in (4.47). As visualised, the trajectory correctly avoids the moving obstacle only if it also avoids the dashed frame.

Various considerations can be made. First the case $v_x > v_{x_{obst}}$ is considered. The obstacle is moving longitudinally slower than the aircraft, which means it can only be seen if it is in front of the quadcopter. Looking at equation (4.47), the term $\left(1 - \frac{v_{x_{obst}}}{v_x}\right)$ in the

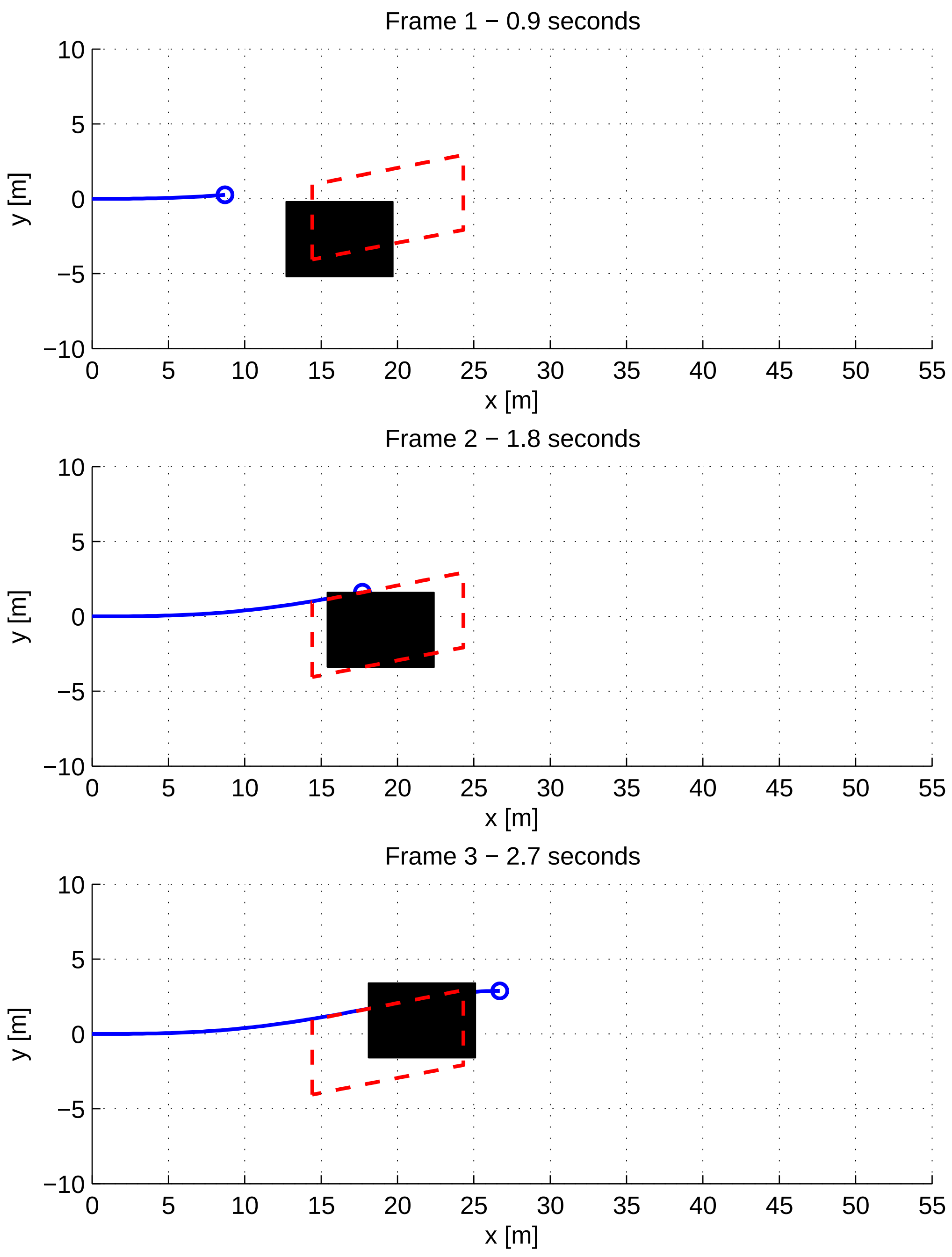

Figure 4.35: Moving obstacle - Frames 1-3 - The red dashed frame represents the fictive fixed object that has to be avoided in order to prevent a collision with the moving obstacle

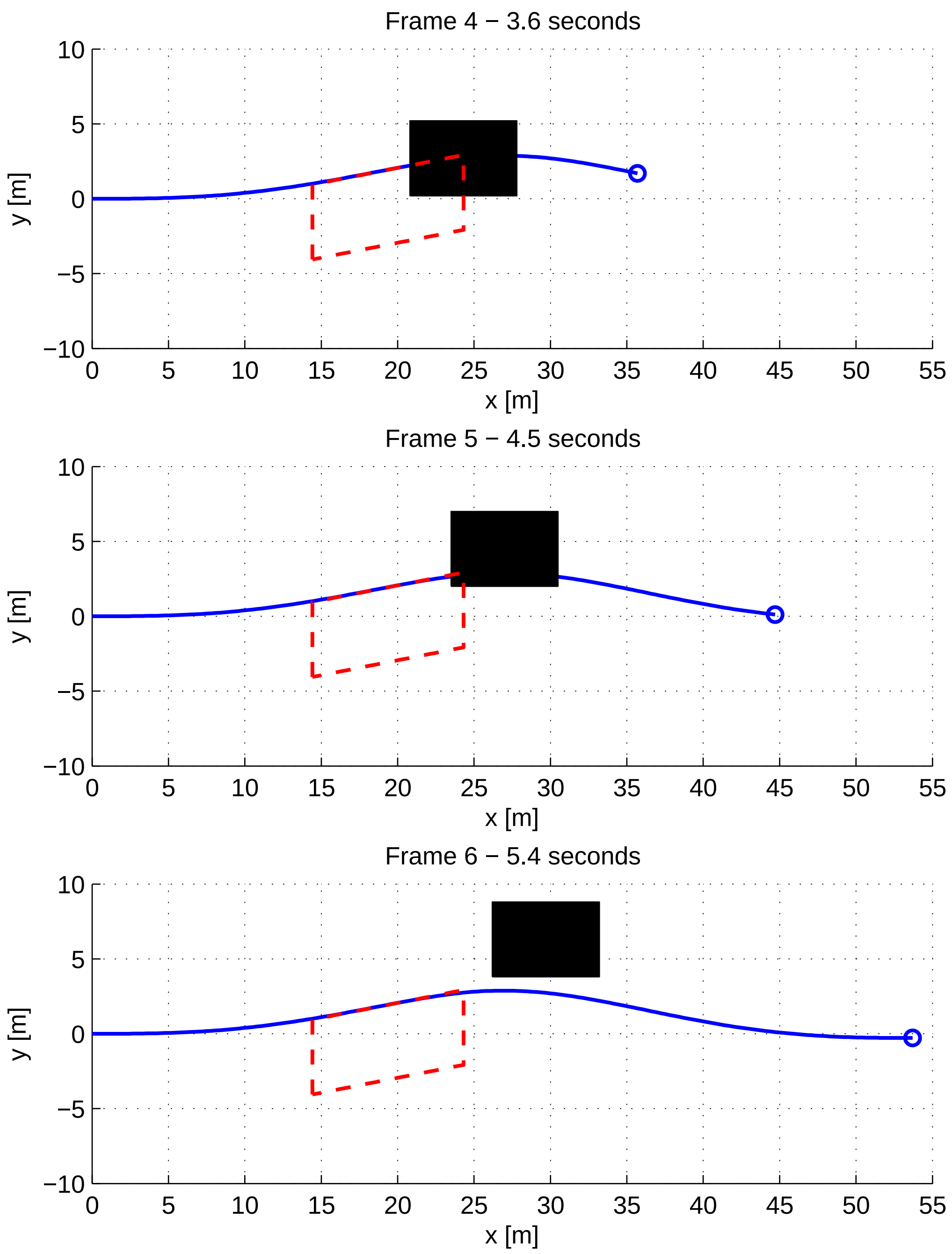

Figure 4.36: Moving obstacle - Frames 4-6 - The red dashed frame represents the fictive fixed object that has to be avoided in order to prevent a collision with the moving obstacle

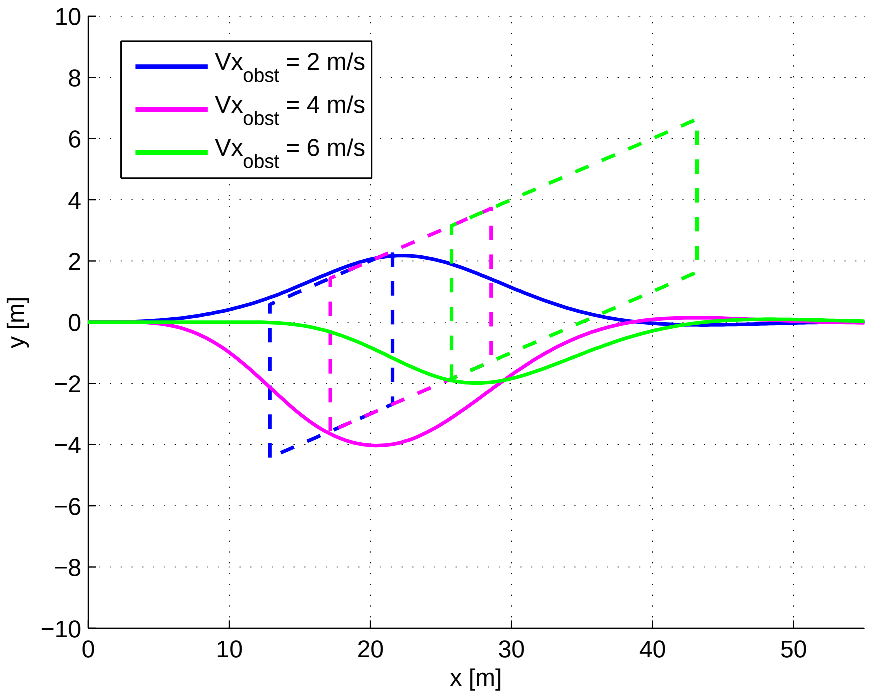

Figure 4.37: Moving obstacle - Bounds corresponding to various longitudinal velocities of the obstacle

function's argument causes a stretching of the bounds. The more similar the two speeds are, the more the shape gets stretched. This comes due to the fact that if the relative velocity is low, the quadcopter takes longer to pass the object. The term $(-x_{0_{obst}} + \alpha x_0)$, with $\alpha = \frac{v_{x_{obst}}}{v_x}$ is responsible for a longitudinal translation. The greater α the more the bounds are shifted to the right, since the quadcopter takes longer to reach the obstacle. Figure 4.37 shows the effect of $v_{x_{obst}}$ on the bounds shape. For clearness, the obstacle has been left out and only the bounds (dashed) together with the resulting trajectories were plotted. As can be deduced from equation (4.47), $v_{y_{obst}}$ brings to a shear transformation of the original shape. The higher the lateral speed of the obstacle, the more distorted is the representation of the bounds in the spacial domain. Figure 4.38 shows the results for different values of the obstacle's lateral speed.

For $v_x \leq v_{x_{obst}}$ the situation is slightly different. The obstacle will be approached only if its initial position is located behind the quadcopter. It first reaches the aircraft with its front side. Figures 4.39 and 4.40 show the results for an obstacle flying at $14\frac{m}{s}$. As can be seen, the quadcopter runs along its side with a negative relative velocity. In

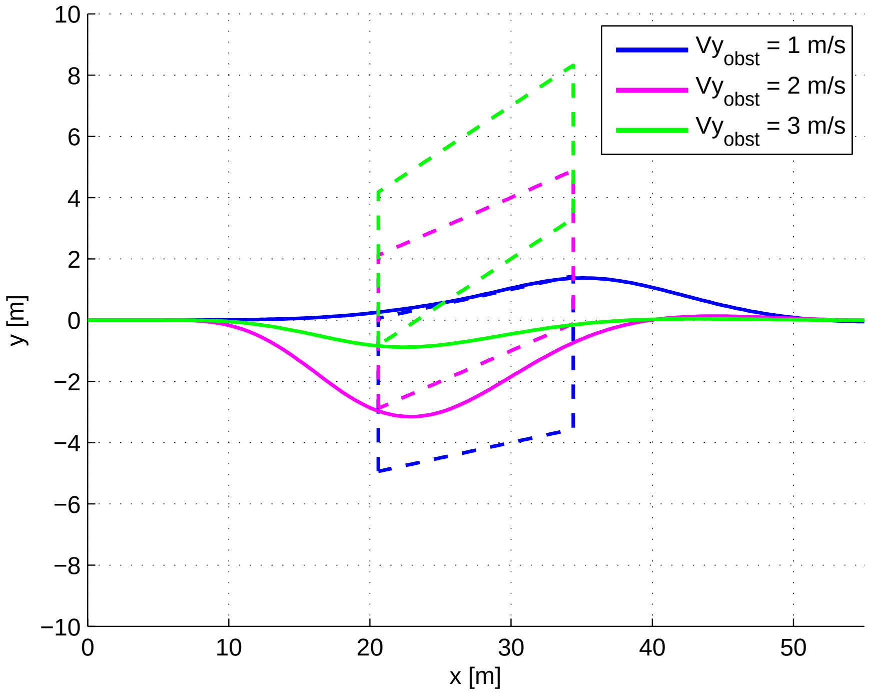

Figure 4.38: Moving obstacle - Bounds corresponding to various lateral velocities of the obstacle

order to pass from the obstacle's shape to the spatial representation of the bounds (red dashed frame), the form-function is flipped, since the quadcopter first approaches the front side of the object. This can be mathematically proved by looking at equation (4.47) and noting that the term $\left(1 - \frac{v_{x_{obst}}}{v_x}\right)$ is negative due to $v_x \leq v_{x_{obst}}$.

The method is capable of dealing with obstacles of various forms, since no limitation is set on the shape function. Figure 4.41 shows frames of the path followed by the aircraft while avoiding a hexagonal object. The object is traveling with $v_{x_{obst}} = 6\frac{m}{s}$ and $v_{y_{obst}} = 2\frac{m}{s}$. Figure 4.42 shows the results for a triangular obstacle moving with $v_{x_{obst}} = 14\frac{m}{s}$ and $v_{y_{obst}} = 5\frac{m}{s}$. Note that, due to $v_x < v_{x_{obst}}$, the dashed frame representing the fixed bounds on the y- positions is flipped with respect to the obstalce's shape. Note that the receding strategy of MPC is capable of adjust the course in case the obstacle changes speed or direction. In fact, the time-discretization is done according to the current forecast of the object's movements, which can actually change over time. In this case the shape of the frame representing the bounds in the spacial domain will change according to the new predictions made.

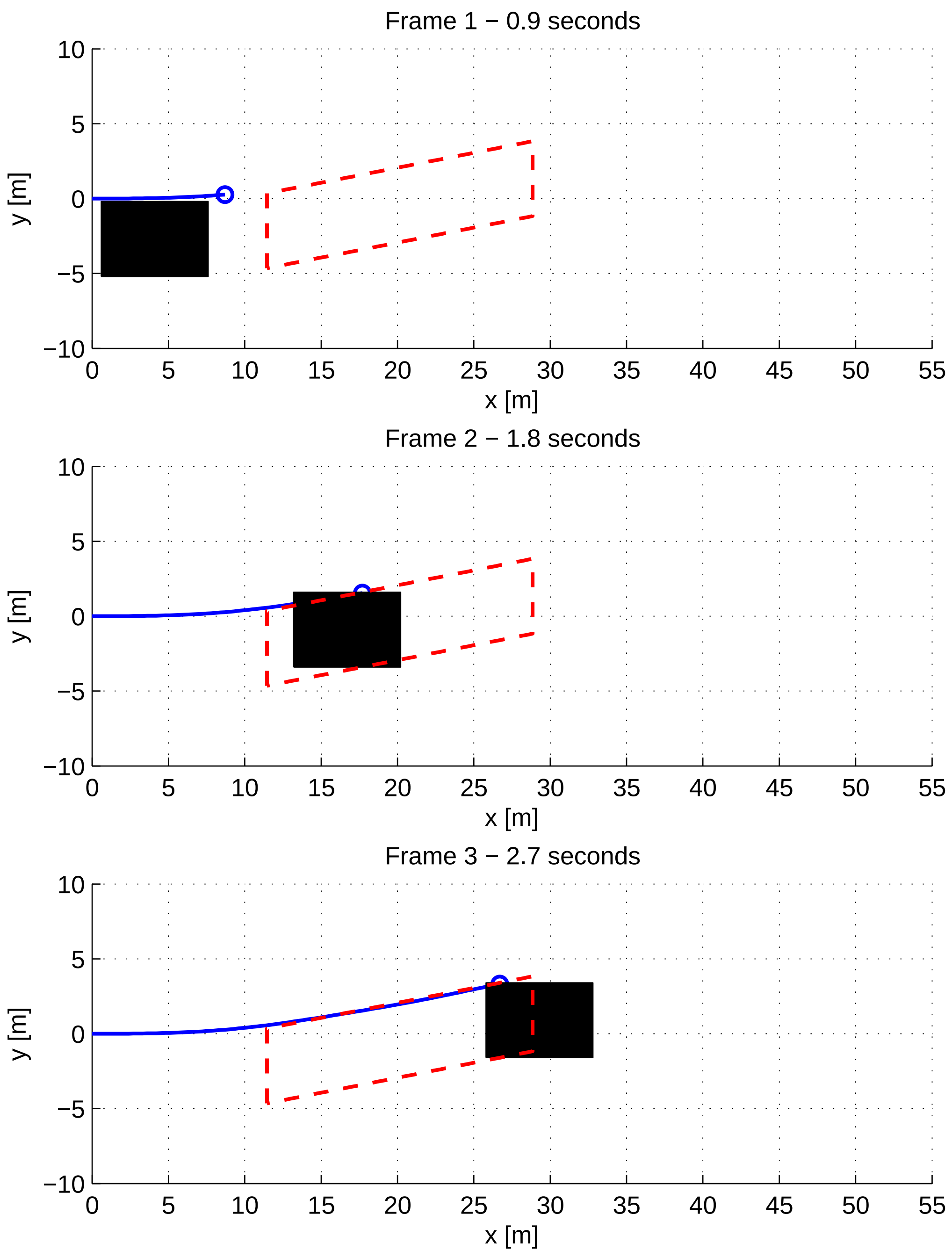

Figure 4.39: Trajectory avoiding an obstacle that is travelling faster than the vehicle - Frames 1-3

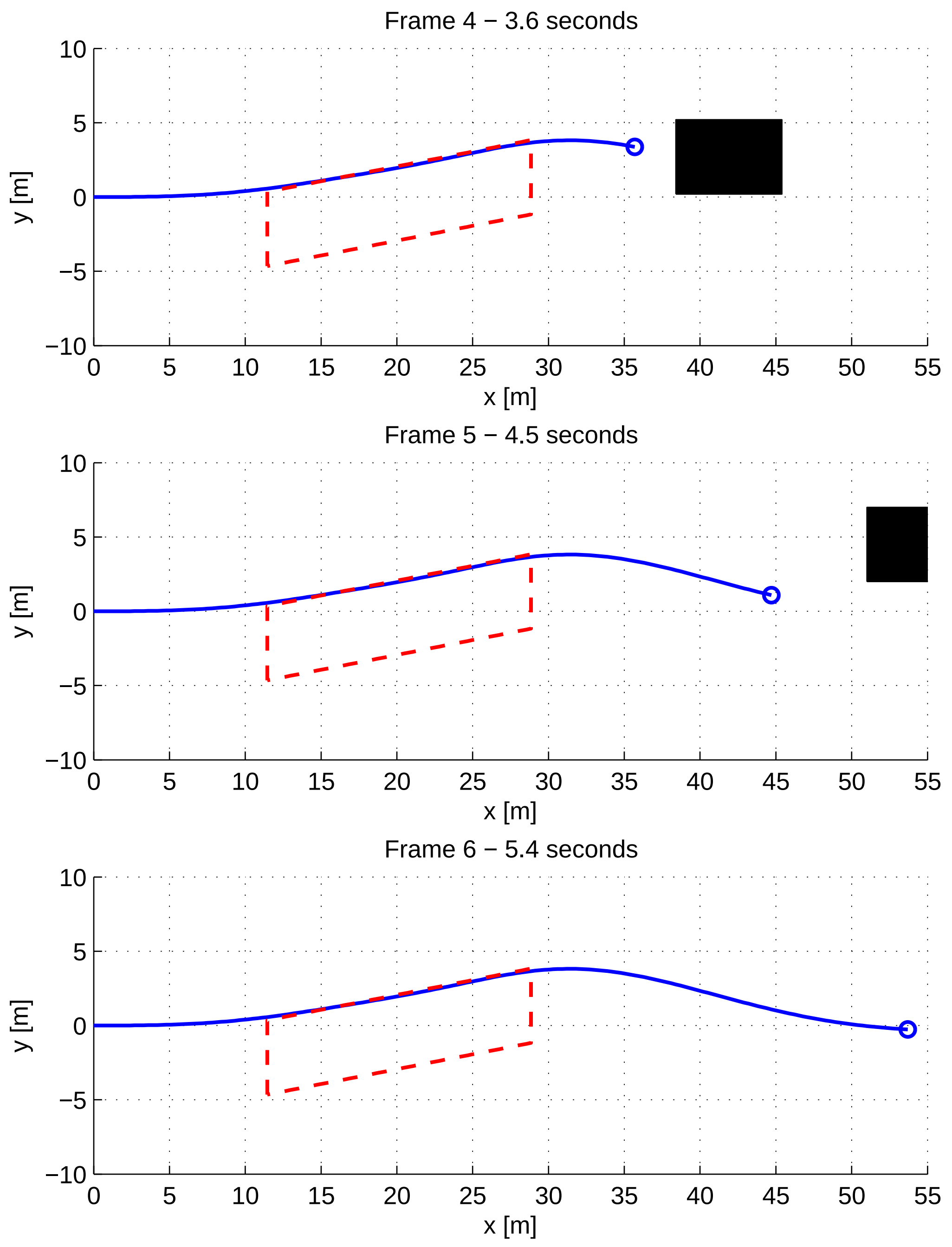

Figure 4.40: Trajectory avoiding an obstacle that is travelling faster than the vehicle - Frames 4-6

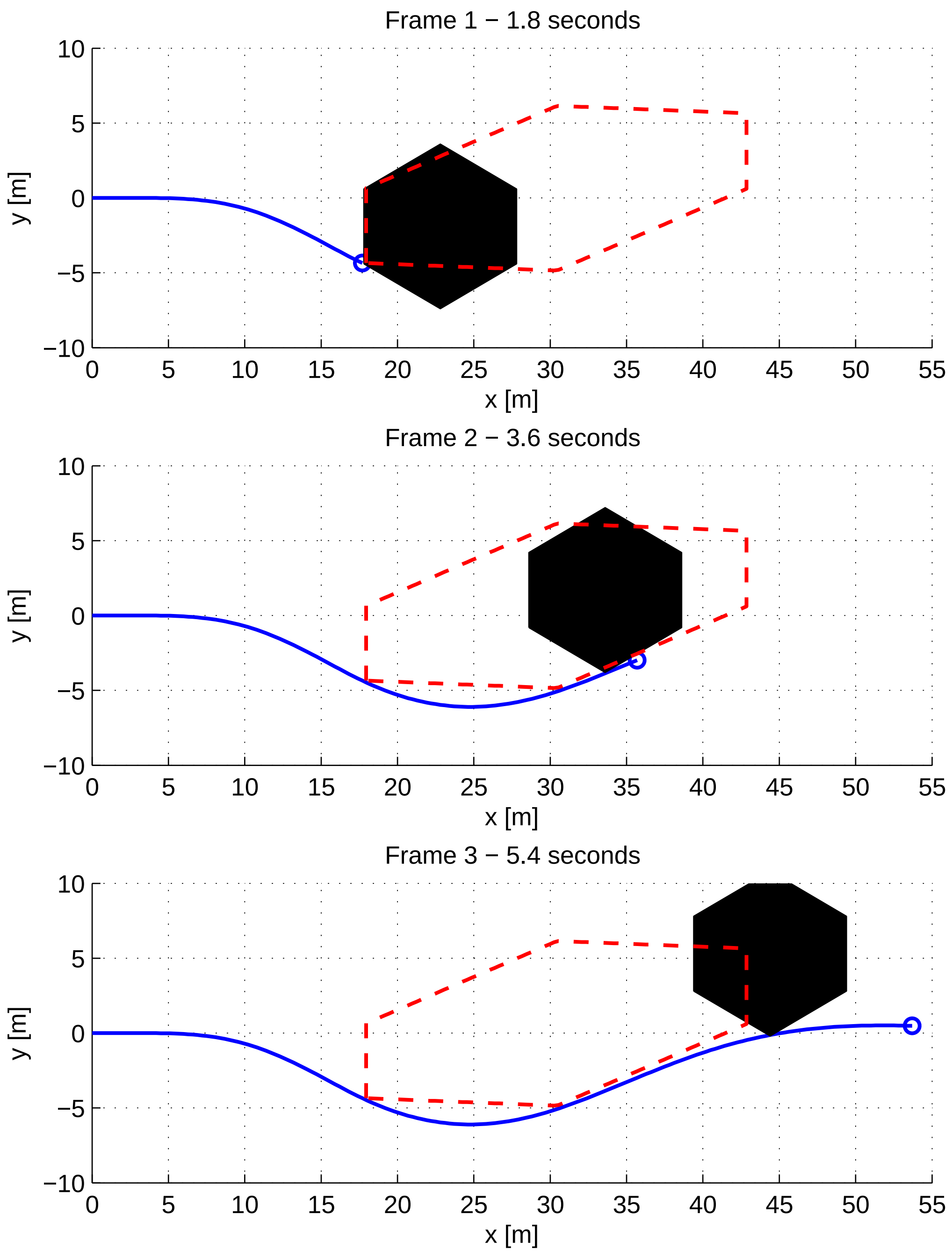

Figure 4.41: Moving obstacle - Hexagon

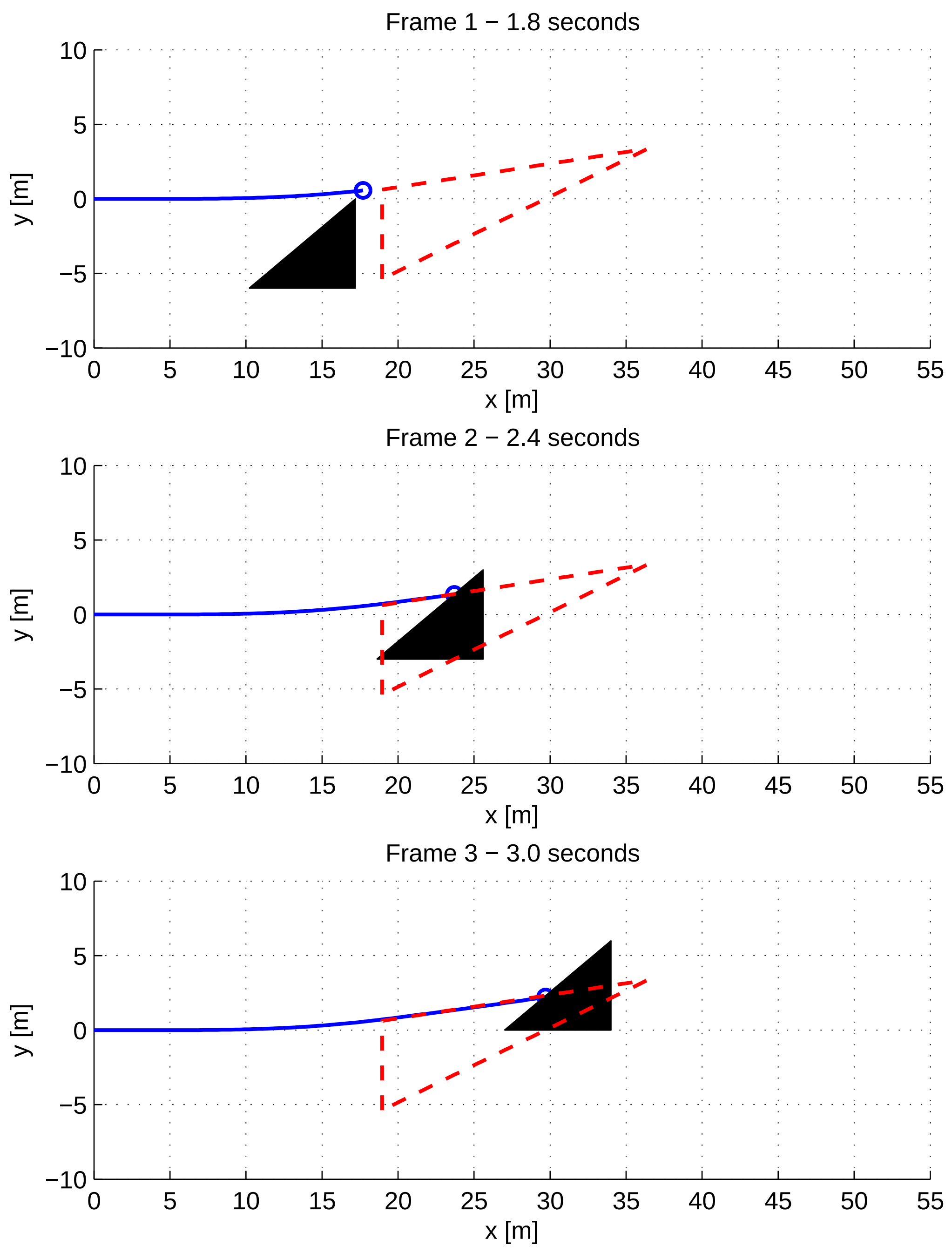

Figure 4.42: Moving obstacle - Triangle - frames 1-3

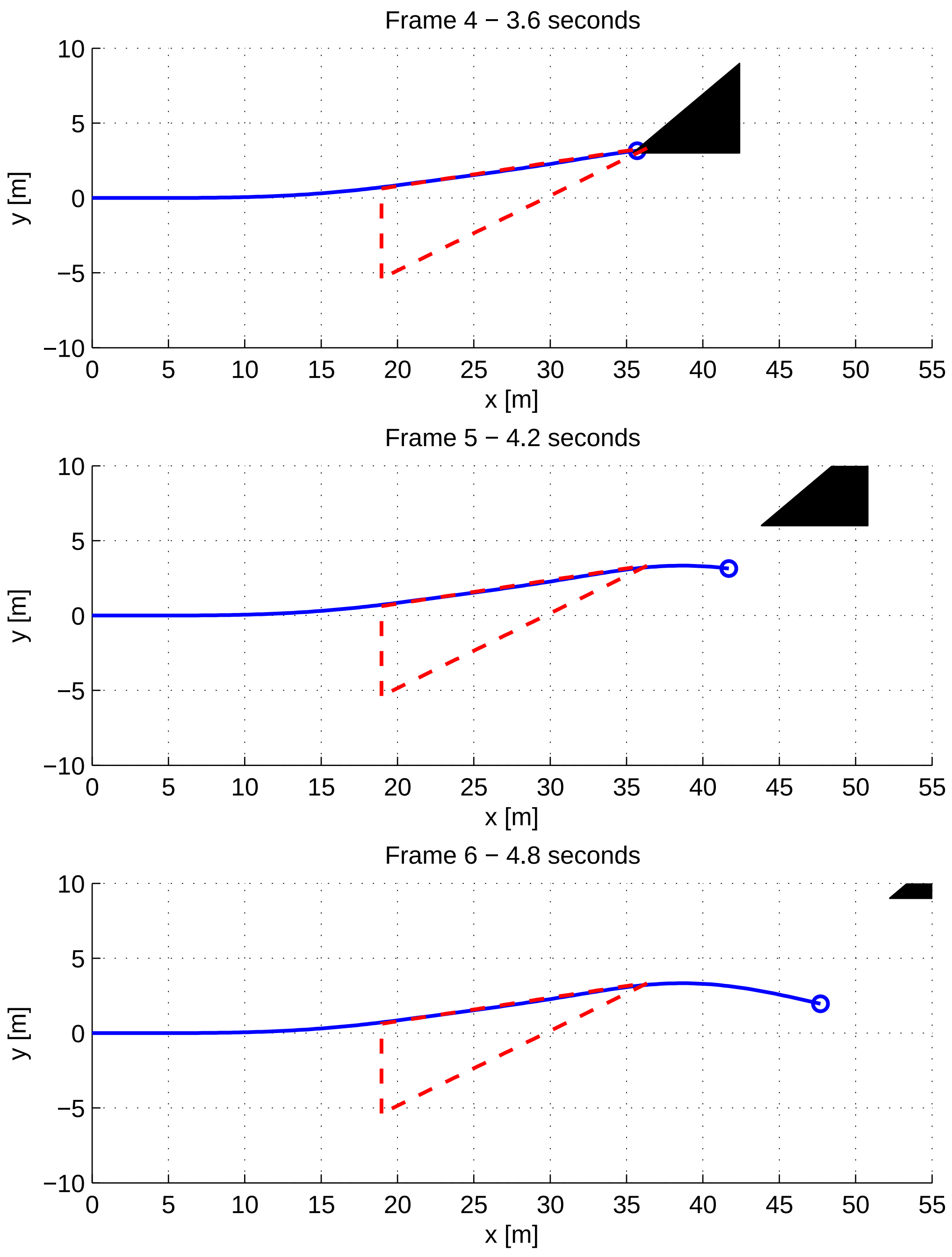

Figure 4.43: Moving obstacle - Triangle - frames 4-6

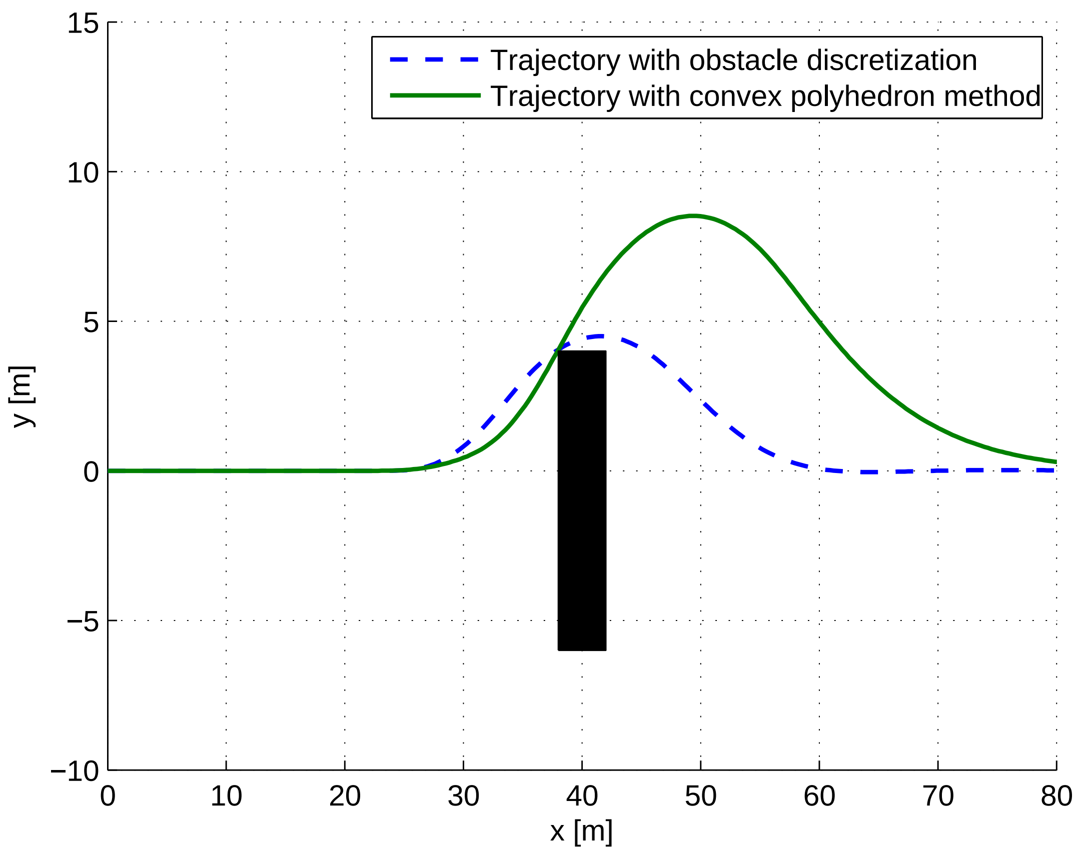

Figure 4.44: Comparison between the two methods - single obstacle

4.9 Comparison between the two strategies

More trials were conducted in order to make a comparison between the obstacle avoidance strategy proposed in [8] and the one developed in this work. The first method approximates the feasible area with a convex polyhedron, which is recomputed for every new optimization according to the current position of the vehicle with respect to the obstacles. Then it imposes that all positions of the solution lie within the convex area. The second one discretizes the obstacle in the time-domain, assuming to travel at constant speed, and accordingly imposes upper and lower bounds on the y-coordinate. If a trajectory is found that does not avoid the obstacle, the longitudinal control reacts decreasing the velocity until a valid trajectory is found. Some relevant trials are presented in this section.

First the two methods are compared in avoiding a single rectangular obstacle. The horizon parameters are chosen equally for both strategies as $t_s = 30ms$ and $N =$

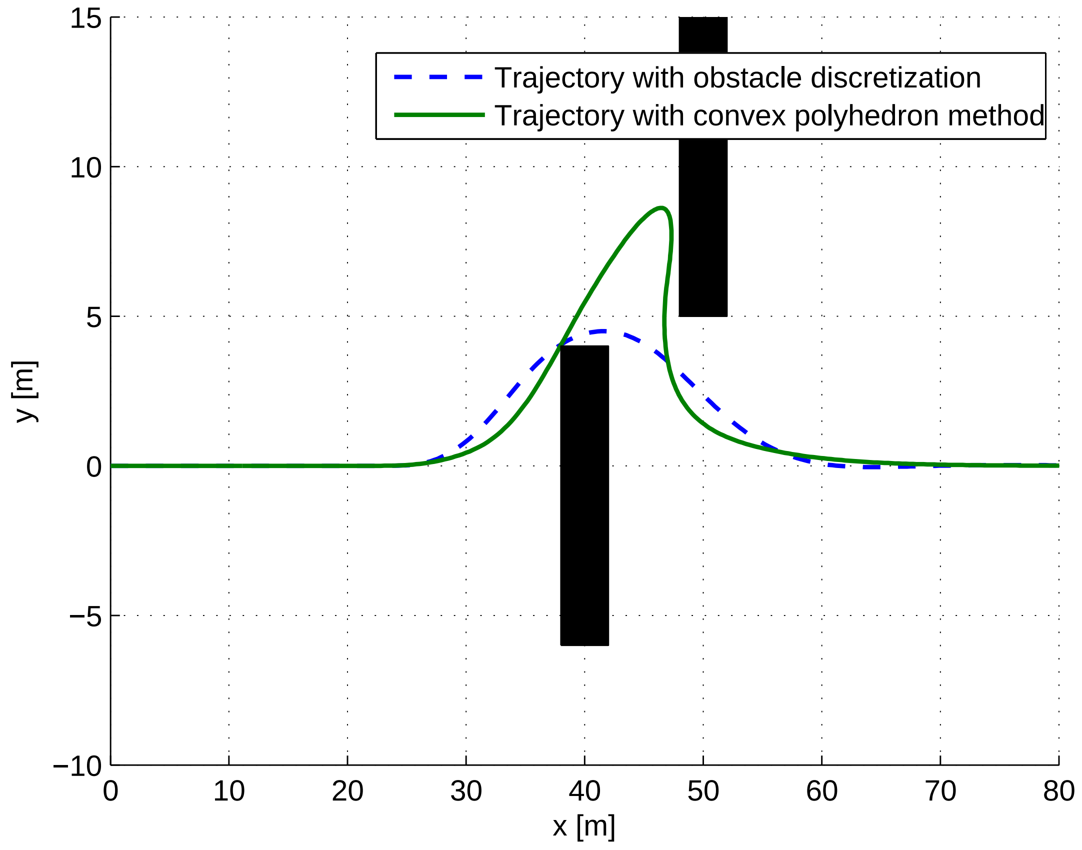

Figure 4.45: Comparison between the two methods - two obstacles

50 in order to obtain a meaningful comparison. The obstacle is 10 meters wide and located in front of the vehicle. Its center is positioned slightly offset with respect to the quadcopter's y-coordinate so that the deadlock situation mentioned in subsection 4.1.4 for the convex polyhedron strategy is avoided. The velocity is limited in both cases to $10\frac{m}{s}$. Figure 4.44 plots the results for the two methods. As depicted, the convex approximation strategy has a much higher overshoot of the y-coordinate. Moreover, while the obstacle discretization method manages to keep a constant longitudinal speed, the other strategy needs to decelerate a lot in order to avoid a collision.

Figure 4.45 shows the results for two subsequent obstacles. As illustrated, the convex polyhedron method has difficulties in finding a valid path. This is caused by the fact that the polyhedra conduct the vehicle towards improper directions. The quadcopter has to shortly reverse its longitudinal velocity, however it gets correctly to the target positioned at 80 meters. The high longitudinal deceleration brings the vehicle to employ

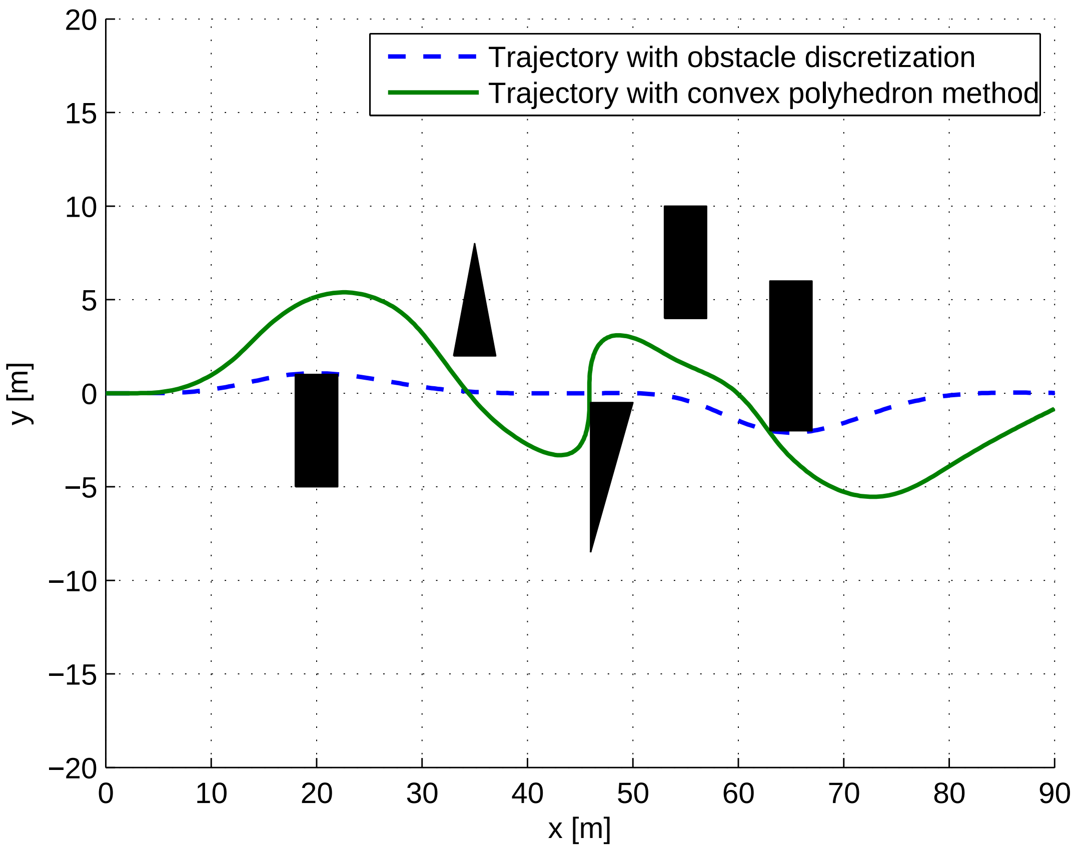

Figure 4.46: Comparison between two methods - various obstacles

more time to reach the final position. Indeed, the convex polyhedron method takes 10.2 seconds to get to the target, whereas the obstacle discretization strategy employs only 8 seconds, since the vehicle travels at a constant speed of $10\frac{m}{s}$.

Figure 4.46 illustrates a scenario consisting of five obstacles. The obstacle discretization method behaves clearly better with respect to the convex polyhedron strategy. The followed trajectory is much smoother and lateral translations occurr only when strictly necessary. The convex polyhedron method, instead, brings the vehicle to heaviliy divert its course. This is caused by the considerable reduction of the feasible area due to the approximation with a convex polyhedron.

In order to quantify the computational load needed to solve each optimization, the CPU time was measured for both the strategies. The simulations were performed using Matlab and the solver qpOASES was compiled into a MEX-function in order to be able to use it directly within the Matlab environment. The calculations were done on a

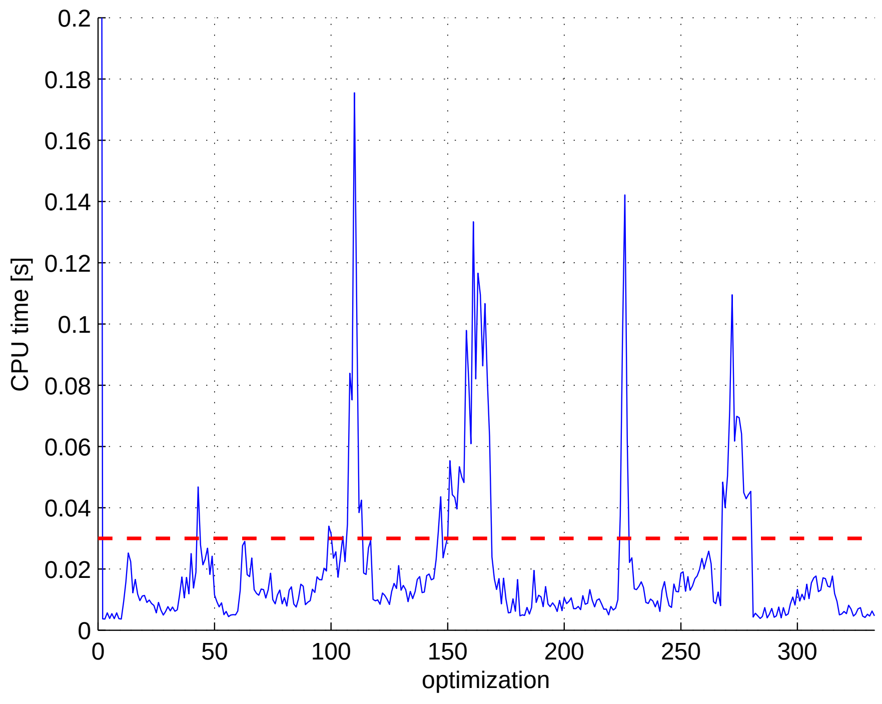

Figure 4.47: Convex polyhedron strategy - CPU time for each optimization

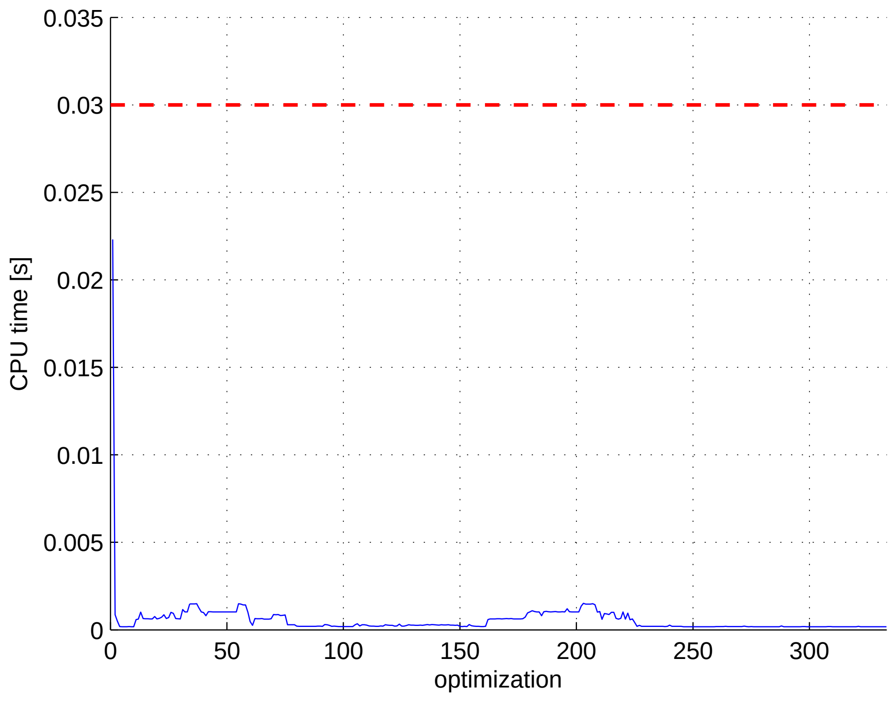

Figure 4.48: Obstacle discretization strategy - CPU time for each optimization

PC running Windows 7, with an Intel Core i7-3610QM at 2.30 GHz, with 8GB RAM. Figure 4.47 plots the results for the convex polyhedron method, whereas figure 4.48 shows the CPU times for the obstacle discretization method. The data refer to the scenario depicted in figure 4.46. The figures present the time needed by the CPU to solve each optimization problem. For real-time applications it is obvious that this time has to be kept lower than the sampling time, since a control action has to be provided every t_s seconds. Note that the initial peak can be neglected, since it represents the time needed to solve the first optimization. This is always much higher than the other values due to the "cold" start of the optimizer. Indeed, for the first minimization problem, it has to "guess" the optimal solution in order to start its algorithm. All further optimizations are "hot-started", which means they are done exploiting the solutions of the previous ones, so that the minimization process will be much faster.

As shown in the figures, there is a great difference between the CPU times needed for the two methods. The values differ by two orders of magnitude. Neglecting the first value, the convex polyhedron method presents a peak of almost 180 milliseconds, which is apparently off limit for a real-time implementation. The situation for the obstacle discretization method is very different. Here, always neglecting the first peak, the maximum value lies at 1.6 milliseconds, which is a very good result. This ensures the requirements for a real-time implementation of this strategy. Note that the computational load depicted in the figures refers only to the optimization processes. In order to evaluate the overall computatinal load it is necessary to consider also routine algorithms as the ones needed to build the bounds for collision avoidance according to the position of the vehicle with respect to the obstacles. However, it can be shown that this operations take much less time, leading the optimization processes to be the main cause of the computational load.

There are several reasons for such a great difference between the computational time needed by the two strategies. The first one lies in the fact that the convex polyhedron method considers the two axes within the same minimization problem, wheareas the other method solves two distinct optimizations, one for each axis. As mentioned in previous chapters, solving a single problem with $2 \cdot N$ variables takes longer than solving two problems each of N variables. Hence, the three-dimensional case is even worse, since $3 \cdot N$ have to be optimized within the same problem. The second main reason is that the

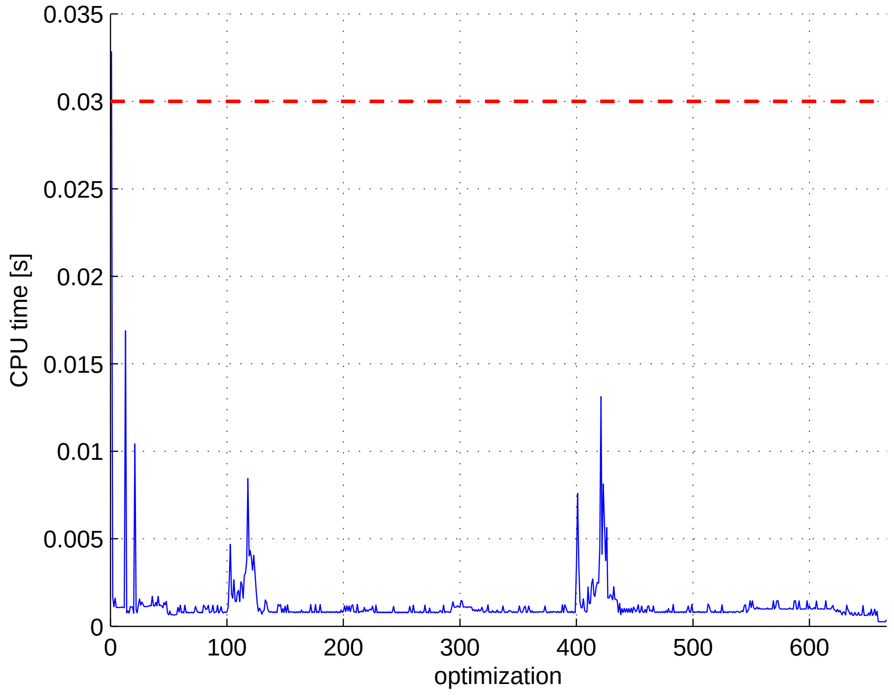

Figure 4.49: Convex polyhedron strategy with $N = 50$ - CPU time for each optimization

obstacle discretization method always imposes at most two bounds on the y-coordinate for each time step, hence in the case of $N = 50$, a maximum of 100 boundary conditions. The convex polyhedron method, instead, imposes, for each time step, a boundary condition for every half-space to be cut out, hence one for each obstacle. This means that for the case of five obstacles and an horizon of $N = 50$, 250 boundary conditions are set. Apparently the time needed to solve an optimization is also related to the number of bounds. An other reason lies in the fact that, for the convex polyhedron strategy, the matrix A_c needed for the boundary conditions changes every new optimization, since it has to be recalculated according to the current position of the vehicle with respect to the obstacles (see eq. 4.1). Hence, the control strategy becomes a linear time varying MPC (LTV-MPC), which in general needs more time to solve the optimizations, since a new matrix A_c has to be passed to the solver every time.

In order to decrease the computational load of the convex polyhedron method, the only solution is to reduce the number of variables considered by the optimization problem, hence to reduce the horizon N. Figure 4.49 shows the results for a simulation done with the same scenario as before, but reducing the prediction horizon by half of its length, thus setting $N = 25$. The computatinal load decreases significantly, such that a real-time implementation could be considered also for this strategy. However, the maximum allowed vehicle speed must also be reduced. In fact the risk of having a short prediction time window is to not react in time to a change of the bounds due to obstacle avoidance.

5 Simulation Results

5.1 Simulation model

It is important to underline that, since dealing with model predictive control, two different system models are needed. The first one is the model that MPC uses in order to make its predictions. This one has to be reliable and has to bring out the main dynamics of the system. On the other hand it has to be kept as simple as possible in order to be efficient in making predictions in short times. Thus, it is not fundamental for it to be exact since, thanks to the receding strategy, the control is capable of coping with model variances. The second model is the one needed to actually simulate the real vehicle. The latter has to be kept as accurate as possible, in order to be able to verify the effectiveness of the control. This model has to represent reality in the most detailed way.

In previous chapters the same system model is used for both MPC and simulation, namely the one described in section 3.4 by equations (3.33)-(3.35), which consists of a set of triple integrators, one for each axis, and takes the respective jerks as inputs of the system. In this chapter the controls are implemented into an exhaustive model of

property	value	unit
mass m	0.58	$[kg]$
inertia tensor J	$\begin{bmatrix} 6.4 & 0 & 0 \\ 0 & 6.4 & 0 \\ 0 & 0 & 12.5 \end{bmatrix} \times 10^{-3}$	$[kgm^2]$
lever arm l	0.17	$[m]$

Table 5.1: model properties

	x-MPC	y-MPC	z-MPC
Q	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
R	0	0.01	0.001

Table 5.2: Weighting matrices for each axis

the quadcopter in order to verify the effectiveness and applicability of the overall control structure. Hence also the low level control, which tracks the body rates evaluated by the MPC, is implemented. This time also the propeller and motor dynamics together with their own speed controllers are taken into account. All this results in a complex model able to represent reality quite well. The quadcopter model relies on an AscTec® Hummingbird with the properties reported in table 5.1. Simulations were done using Simulink®, while the model predictive controller uses an embedded Simulink-interface of the software qpOASES [14] in order to solve the optimization problems.

Here the attitude control is simply provided by a proportional controller as described below.

$$M_1 = kp_1 \left(\omega_{1r} - \omega_1\right) \tag{5.1}$$

$$M_2 = kp_2 \left(\omega_{2r} - \omega_2\right) \tag{5.2}$$

$$M_3 = kp_3 \left(\omega_{3r} - \omega_3\right) \tag{5.3}$$

where ω_{1r} and ω_{2r} are the reference body rates evaluated by the MPC, whereas the yaw speed is referenced as $\omega_{3r} = 0$. The values of the body rates ω_1,ω_2 and ω_3 are provided by onboard sensors. M_1, M_2 and M_3 are the torques about the body fixed axes, which are then mixed together with the evaluated total thrust f to individual motor thrusts exploting equation 2.7 as depicted in section 2.3. Note that the three rotations can be decoupled, since the inertia tensor J is diagonal.

5.2 Simulations

In the trial presented in this section the quadcopter has to avoid three subsequent obstacles while flying at a longitudinal speed of $10\frac{m}{s}$. The obstructions are respectively 3 meters, 7 meters and 14 meters wide. The control on the vertical axis is supposed to maintain a ground clearance of 2 meters, while the longitudinal control has to keep a constant speed and to provide a deceleration only in case the lateral control is not able to find a valid trajectory that avoids a collision. Hence, the lateral MPC, acting on the y-coordinate, is responsible for the obtsacle avoidance task.

Given the tasks that each control has to carry out, the parameters that define the performances of the model predictive controller have to be chosen. Table 5.2 reports the weighting matrices for states and jerk, respectively Q and R, that have been set for each axis and that provide a quite good overall performance. As shown in the table, the MPC acting on the longitudinal coordinate only weighs velocity and acceleration. This is due to the fact that no position reference is given to the x-coordinate, since the vehicle is intrinsically flying towards a given target. The coordinate system is indeed placed with its origin at the starting position of the vehicle and directed with its x-axis towards the target. This means that, given $v_x > 0$ and assumed that the lateral control is correctly following its reference on the y-coordinate, the vehicle will reach the final position. The state-weighting matrix for the lateral MPC is chosen in order to track the reference ($y_T = 0$). Here also the jerk is slightly weighed so that smoother translations are obtained. Weighing the jerk avoids high peaks on the body rates ensuring gradual movements. The vertical control weighs the altitude error with respect to the reference ground clearance of 2 meters. Additionally also the velocity is weighed in order to have a damping action.

The slack variable for the boundary conditions needed for obstacle avoidance is weighed with $w_e = 9 \cdot 10^6$. So as to ensure that the vehicle is not flying too close to the ground or too high, lower and upper bounds on the vertical coordinate were added. These are chosen to $lb_z = 1m$ and $ub_z = 3m$. The boundary conditions that derive from the imposition of limitations on the vertical position are "relaxed" by a slack variable in order to avoid problems associated with an infeasibility of the optimization solution. This slack

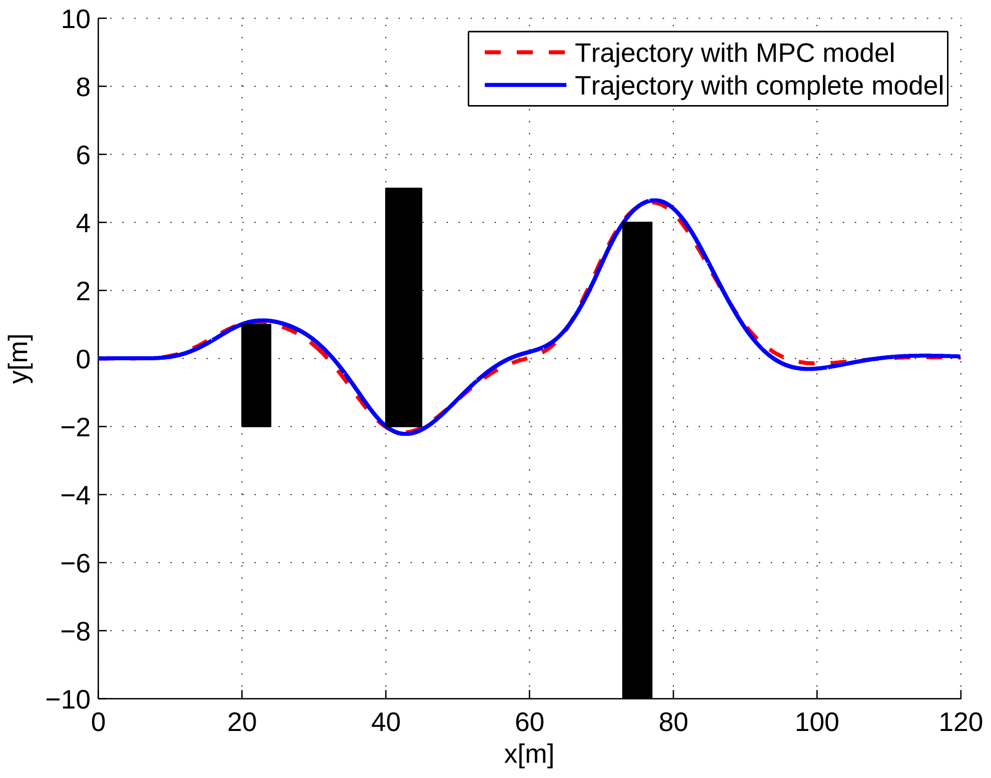

Figure 5.1: Comparison between MPC model and complete model

variable is again weighed with $w_e = 9 \cdot 10^6$. Hence, the task of remaining within a feasible altitude of 1-3 meters, is treated in the same way as for the obstacle avoidance task.

The parameters setting the prediction horizon are chosen equally for all three axes to $t_s = 3ms$ and $N = 50$. Note that the sampling time has to be the same for all axes, since the body rates and the total thrust are computed mixing together the optimal jerks and accelerations evaluated by the three MPCs. Therefore the values of the three jerks have to be provided simultaneously.

Figure 5.1 illustrates the resulting trajectory followed by the quadcopter in order to avoid a collision with the three obstacles. To make a comparison, the same trial was implemented simulating the model exploited by the MPC to make its predictions, which consists only of a set of a triple integrators as described in equations (3.33)-(3.35). As mentioned in previous chapters this model does not consider neither the rotational dynamics, nor the motor and propeller dynamics nor any aerodynamic effect. The red dashed line depicts the trajectory resulting by simulating the latter. All parameters were

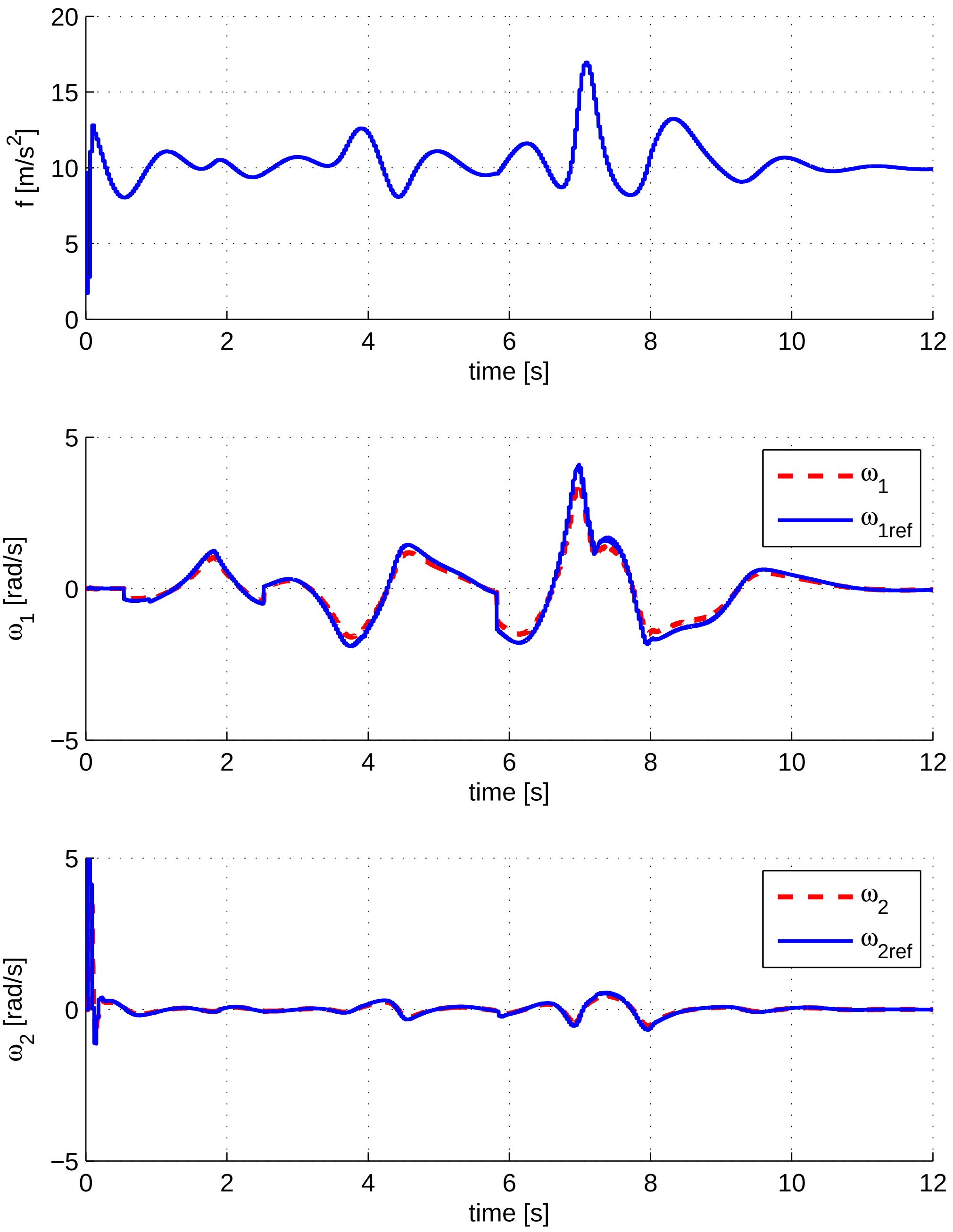

Figure 5.2: inputs - total thrust, normalized by the vehicle mass, and two body rates commanded by model predictive control - the dashed lines illustrate the real body rates as tracked by the attitude control

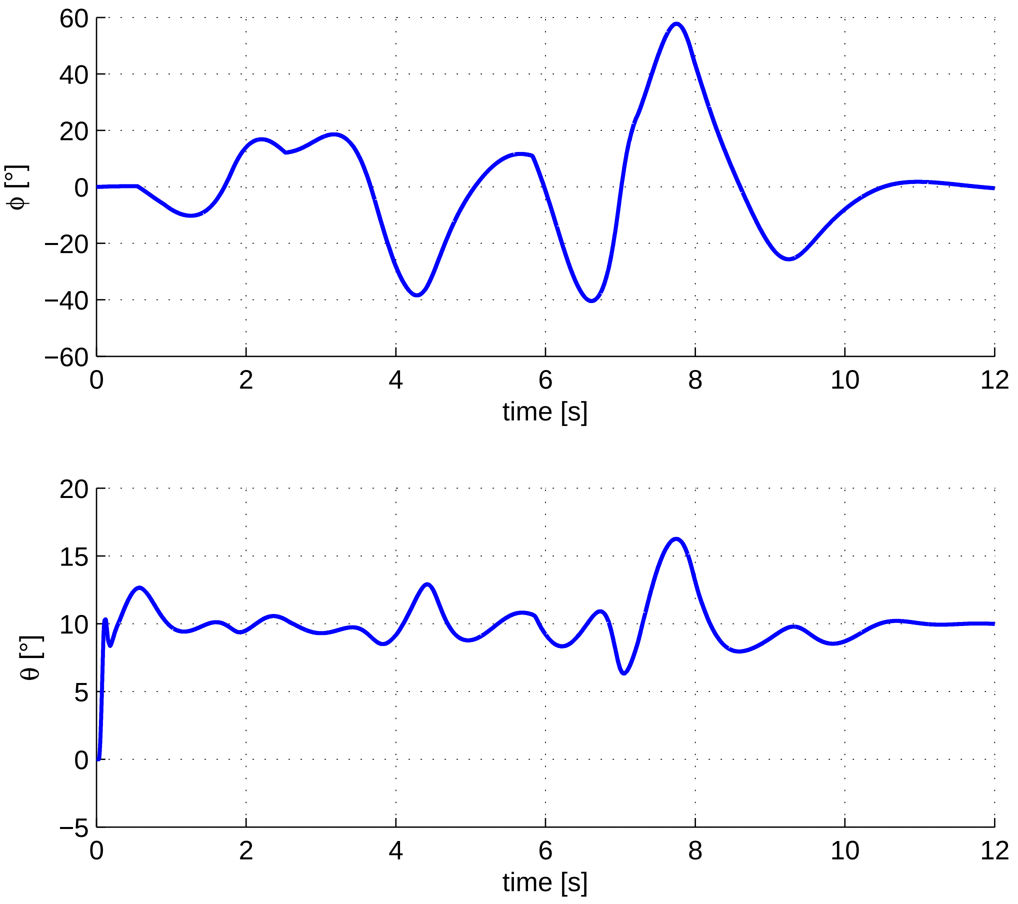

Figure 5.3: Euler angles of the body frame with respect to the inertial frame

chosen the same way for both simulations in order to be able to make a meaningful comparison. As depicted in figure, the trajectory achieved by simulating the comprehensive model differs very little from the one resulting from implementing the control on the model used by MPC. The good results justify the choice made by passing a very simple system model to the MPC. As mentioned in chapter 3, the great advantage of feeding the model predictive controller with a trivial model of the system, lies in the fact of being able to solve optimization problems very quickly. The receding strategy is able to cope with all simplifications made by approximating the complex system with a very simple one. Thus, the effectiveness of the control has been proven, which means MPC is able to manage the deviation between the real behavior of the system and the predictions made.

Figure 5.2 illustrates the inputs commanded by the model predictive controller after mixing together the three optimal jerks and solving equations (2.11) and (2.15). Note that for computing the desired inputs, namely the total thrust f normalized by the vehicle mass, and the two body rates ω_{1ref} and ω_{2ref}, also the optimal acceleration is needed. The latter is simply evaluated by integrating the jerk trajectory. The rotation matrix R of the body frame with respect to the inertial frame, which is also needed to compute the inputs, is calculated exploiting the estimation of the attitude provided by onboard sensors. The red dashed lines depict the real body rates ω_1 and ω_2 resulting from the overall control action. As can be seen, the attitude control manages very well to track the reference body rates provided by MPC.

Note that body rate ω_1, namely the rotational rate about the body-fixed x-axis, is responsible for lateral translations, since it leads the total thrust to have a lateral component. Thus, it is mainly associated with the obstacle avoidance task. This explains the high peaks depicted in figure 5.2.

Body rate ω_2 is associated with longitudinal movements. The fluctuations that can be seen are due to the control action that tries to keep the longitudinal speed as close as possible to $10\frac{m}{s}$. A peak is noticeable shortly after $t = 0$. This is caused by the control that has to slightly rotate the body frame in order to counteract the aerodynamic forces. In fact, though the simulation starts with the vehicle already having a speed of $10\frac{m}{s}$, the body frame is initially flat. Thus, a small angle around the body-fixed y-axis has to be provided in order to achieve a longitudinal force that compensates for aerodynamic effects.

Figure 5.3 depicts the Euler angles ϕ and θ, respectively about the x-axis and y-axis. As can be seen, θ fluctuates around a steady state value of approximately $\theta_s = 10°$. This is the angle of the frame needed to bend the vector of the total thrust forward in order to counteract the aerodynamic force associated to a speed of $10\frac{m}{s}$.

Figure 5.4 shows the velocities for all three axes. The dashed lines represent the simulations done directly on the model used by MPC to make its predictions. The longitudinal control is able to track the $10\frac{m}{s}$ speed with very small deviations. As demonstrated, MPC is able to cope with modeling errors very well. For instance it manages to maintain a constant speed despite the influence of external forces that have not been modeled.

The lateral velocity v_y does not differ much from the one resulting from the simu-

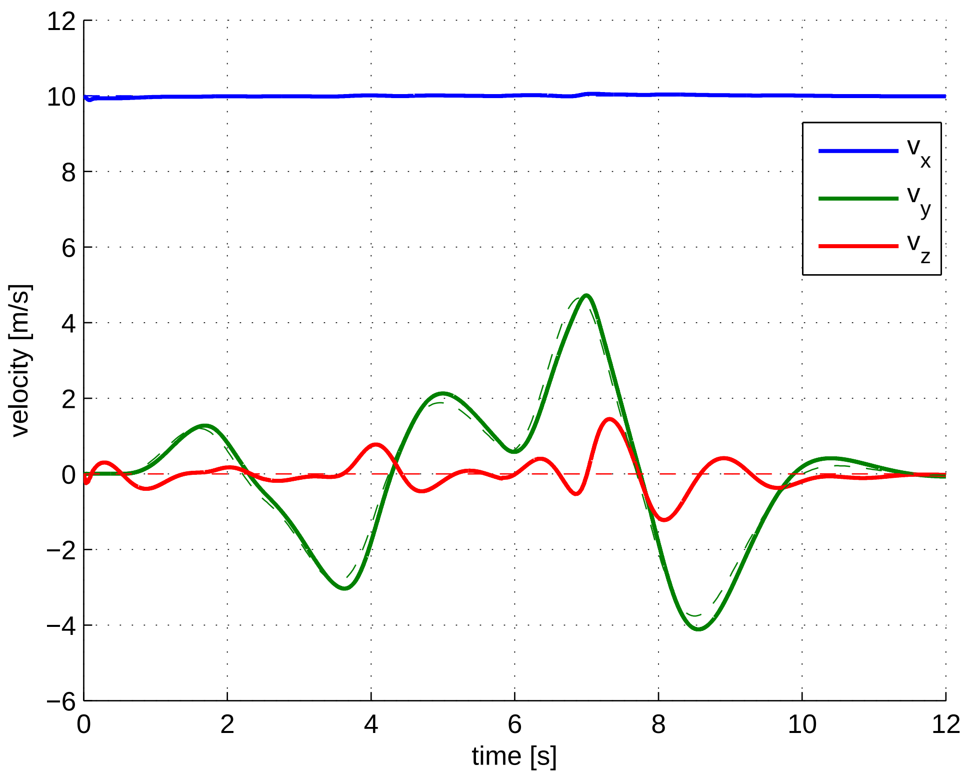

Figure 5.4: velocities - the dashed lines represent the results of the simulation made on the same model used by MPC

lations done on the simple model. As said before, lateral movements are responsible for avoiding the obstacles and tracking the reference $y_T = 0$.

Larger differences between simulations on complete model and results with simple model are notable for the velocities about the z-axis. Here the fluctuations are more relevant, with a peak of $1.45\frac{m}{s}$. These are caused by strong aerodynamic effects arising at higher speeds. As can be seen, the greatest fluctuations arise always shortly after a peak of the lateral speed v_y, hence corresponding with high values of the magnitude of the overall velocity.

Figure 5.5 plots the trajectory of the z-coordinate. As depicted, the altitude decreases up to 1.8 meters and has a maximum overshoot of ca. 0.62 meters caused by the mentioned aerodynamic effects. Since the ground clearance represents a priority for safety reasons one can assume to put stronger lower bounds on the z-coordinate. For this reason a trial was made setting $lb_z = 1.99m$ and $ub_z = 2.4m$. Also the upper limits on

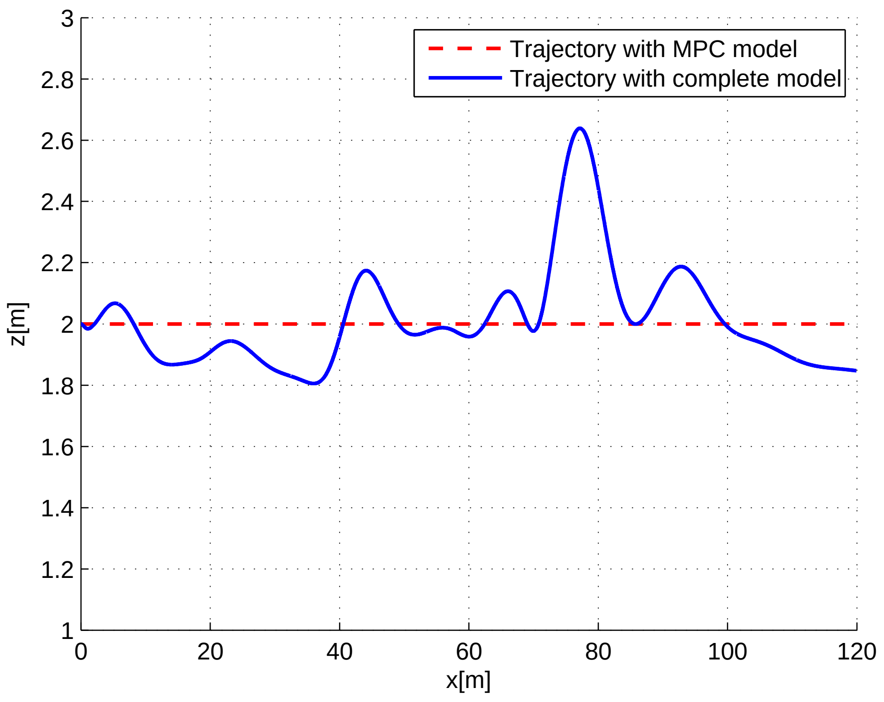

Figure 5.5: Trajectory for z-axis - bounds are set to $lb_z = 1m$ and $ub_z = 3m$

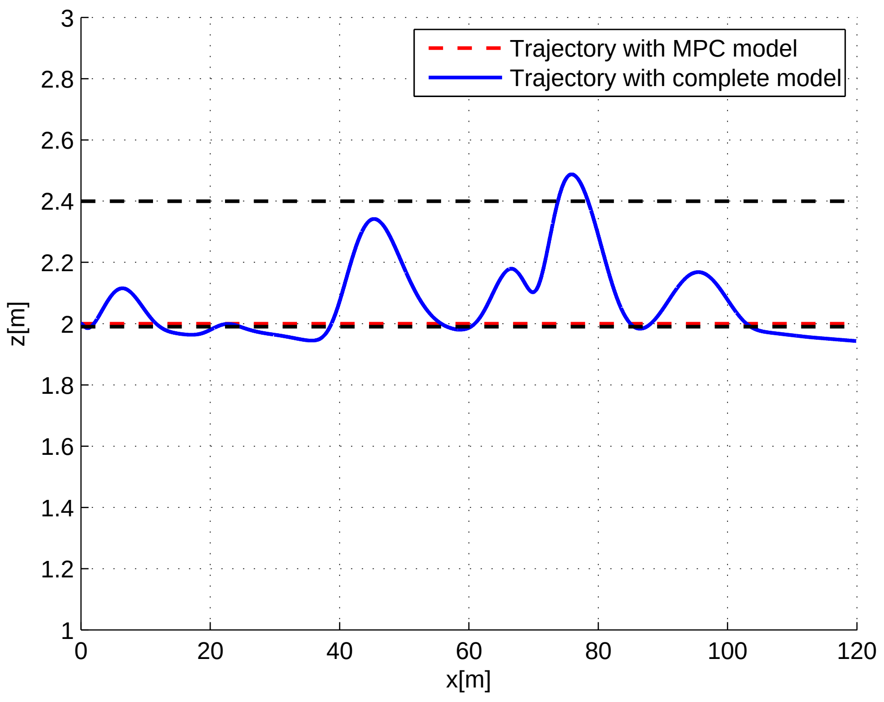

Figure 5.6: Trajectory for z-axis - bounds are set to $lb_z = 1.99m$ and $ub_z = 2.4m$

the altitude were set stronger in order to try to reduce the overshoots of the z-coordinate. However, experiment have shown that the slack variable has to be weighed in a milder way. Otherwise the optimazation solver will have difficulties in finding a valid solution and will return an infeasibility error. Hence, though the new bounds limit the altitude to lie within a smaller region ($1.99m$-$2.4m$), the corresponding boundary conditions are set in a "softer" way, since the slack variable is weighed less into the cost function, that is with $w_e = 1 \cdot 10^4$. This means, an overstepping of the bounds will be less penalized.

Figure 5.6 shows the results. As can be seen the situation has improved. The altitude drops with a maximum value of $0.057m$ and has a peak of $2.49m$. Hence the bounds are slightly overstepped due to the slack variable. The z-coordinate however is kept within a range of $0.5m$, which is an acceptable result. A further reduction of the feasible range for the altitude has shown not to bring any improvements. This means that, though bringing the upper and lower limits closer to each other, overshoots of approximately 0.4 meters still occurred. Moreover, limiting the solution for the altitude to lie in a too small region leads to infeasibility problems of the optimization.

5.3 Real-time feasibility

One of the purposes of this work is to develop a model predictive controller suitable for real-time implementations. This means it has to be able to control the system online, managing to solve its algorithms fast enough to not cause any losses in performances and stability. In the special case of MPC, real-time applicability translates in its ability to solve all the necessary calculations within a time equal to the chosen sampling time t_s. In fact, after this time lapse a new control input has to be evaluated in order to perform a feedback action for the system. As known, one of the main drawbacks of MPC is the time needed for the solver to find a solution for the optimization problem, which underlies the control strategy. For this reason all considerations on the control parameters as well as on the choice of the system model that is handed to the MPC have to be made in order to keep the computational load as low as possible. This brought to choose a very simple model of the vehicle as well as a not too long prediction horizon. On the other hand the control has still to be able to deal with all dynamics of the system in order to not cause stability problems.

In section 4.9 some results regarding the computational load needed for each optimization to be solved are already shown. Those simulations are done using Matlab. The solver qpOASES is used directly within the Matlab environment. It is compiled into a so-called MEX function, hence still providing very fast performances. However, the control needs to perform other routine algorithms like the evaluation of the bounds for obstacle avoidance as well as the recalculation of the vectors to be passed to the optimization problem. In fact, looking at equations (3.27),(3.28),(3.29), vector g as well as the overall lower and upper bounds, lb_A and ub_A, depend on the actual state vector x_0, which apparently changes for every new optimization, since it is evaluated according to the informations provided by the sensor unit. As known, though very versatile, Matlab is not a fast programming language. This means that solving all auxiliary calculations needed for the control using Matlab is not the most efficient way. Hence, though the performances were already very satisfactory, the overall model predictive controller, including optimization solver and routine algorithms, was coded in programming language C using Simulink Coder™. The compiled controller was then passed to a Real-Time Target computer and simulated in order to obtain reliable values of the computational load. Various trials were performed so as to to verify the effectiveness of the control together with its real-time capability. In order to get closer to the performances provided by current microcontrollers, trials were performed simulating various values of the processor's speed.

The horizon parameters were set to $t_s = 30ms$ and $N = 50$. Note that the first parameter, namely the sampling time t_s, sets time restrictions for the optimization. Indeed it represents the maximum time allowed to find a valid solution. The horizon length N instead determines the complexity of the optimization problem. In fact it sets the number of variables the solver has to cope with, which apparently increases the numerical calculations that have to be performed. Note that only the controller is simulated with the real-time computer. The actual system response is not simulated, since it would only charge the processor with unnecessary calculations. In fact, in real applications, the controller only has to evaluate the control action according to the informations on the current vehicle state.

Note that the first optimization needs a longer time to be solved. This is due

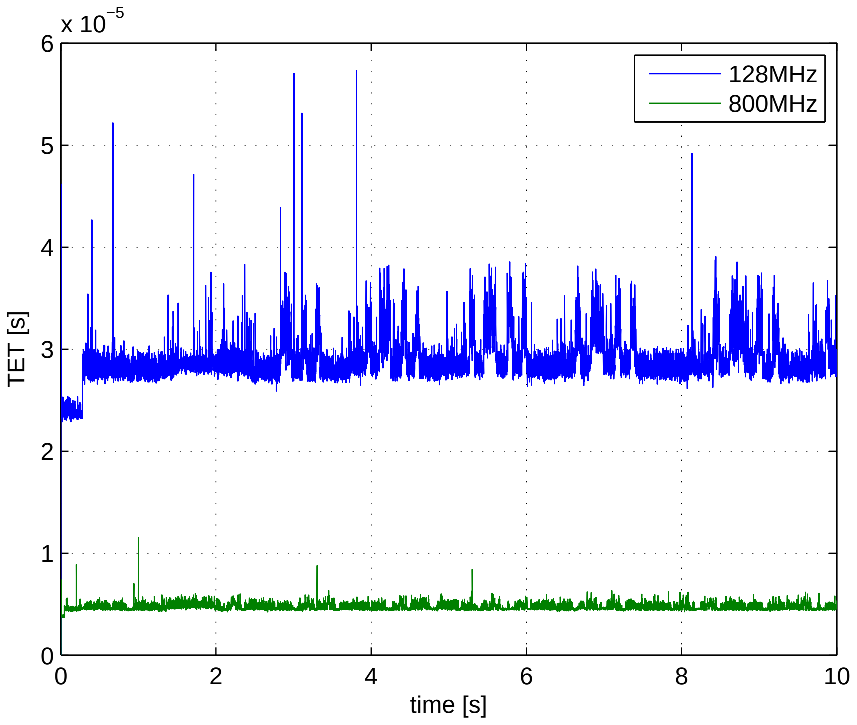

Figure 5.7: Task execution times at different CPU rates

to the online active set algorithm exploited by qpOASES, which needs to start in a "cold" way. This means, that for its first optimization it has to make guesses about the possible solution, which apparently costs computational load. For this reason the real-time machine was allowed to take more time for the very first optimization problem. For all successive optimizations the algorithm takes the solution of the previous problem and uses it as an initial guess. This procedure is called "hot start", and brings the subsequent solutions to be found within a shorter time.

The sampling time at which the overall control unit processes its tasks is set to $\Delta t = 1ms$. All auxiliary and routine algorithms are processed within Δt. In order to allow the optimization processes to last within $t_s = 30ms$, the minimization algorithm is splitted in subtasks that are computed at a sampling rate $\frac{1}{\Delta t}$. Thus, the simulation can be performed at a single sampling rate.

Real time feasibility was tested for the obstacle discretization strategy. Figure 5.7 shows the results for a simulation done considering a single obstacle. The trial was repeated simulating two different CPU rates: $800MHz$ and $128MHz$. As can be seen, the processor manages in both cases to compute the tasks within a lower time with respect to the primitive sampling time $\Delta t = 1ms$.

6 Conclusions and outlook

In this work the design of a model predictive control for online trajectory generation including fixed and moving obstacle avoidance is presented. The design relies on a control structure originally developed for state interception manuevers and described in [9]. The choice to use that control strategy was done due to the very simple model that model predictive control has to manage. Furhtermore no linearization is needed, since the model only considers translational movements and thus can be described through a set of triple integrators. The rotational movements can be evaluated analytically given the solutions provided by model predictive control. Basing on this results, the trajectory planning for distant targets together with the collision avoidance task have been added. Initially only the two-dimensional case, hence the case in which the vehicle is moving in the x-y plane, was considered. First, the strategy presented in [8], which relies on the approximation of the flyable area with convex polyhedra, was implemented. Several problems associated with this method have been detected. These issues lead to considerable restrictions on the real-time feasibility as well as on the maximum speeds that may be achieved in order to guarantee a collision avoidance. All this brought to choose a complete different strategy for avoiding obstacles. Still using linear constraints, that can be managed much easier and with very low computational loads, a new method was implemented, which relies on the time-discretization of the obstacle. Indeed, assuming to fly at a constant speed, the bounds on lateral positions, which in the spatial representation are function of the longitudinal coordinate, start depending directly on time. Hence, the boundary conditions can be directly applied to the lateral movements. This, together with the holonomic nature of the quadcopter, implies an other great advantage, that is the possibility to completely decouple the axes. Thanks to it the problem can be splitted in two different optimization problems, which leads to great improvements in terms of computational load. The obstacle avoidance task is only provided by the lateral control, since the bounds are imposed on the y-coordinate. However there might be cases in which the longitudinal speed needs to be dropped in order to avoid an obstacle. For this reason the use of a slack variable was added. The latter has two main functions. The first is to "soften" the hard constraints on the lateral positions in order to prevent that the optimization solver returns an infeasibility error in case an obstacle can't be avoided (due to physical limitations). The second function of the slack variable is to send the information to the longitudinal control, in order to decrease the speed in case a trajectory is found, that does not avoid a collision. The only way that has been proved to be efficient was the one that reduces the speed with maximum deceleration until the lateral control again finds a valid trajectory that correctly avoids all objects on the way. The strategy has shown great results in avoiding multiple obstacles simultaneously as well as objects of different shapes and dimensions. The same method was then applied to three dimensional environments again showing good performances. At the end, also a method to avoid moving obstacles was proposed. The latter relies on the assumption that the obstacles move with constant speed during the prediction horizon. Thanks to the receding strategy of model predictive control a new forecast of the obstacle's velocity can be made for every new optimization, thus the course can be adjusted in case of unpredictable changes of speed and direction of the obstacle. The strategy assures, that given the mentioned assumptions, the bounds on the y-coordinate deriving from the collision avoidance task, remain fixed in the space and can be therefore considered as a fictive fixed object. The method was able to manage obstacles moving both slower and faster than the vehicle. At the end of the chapter a comparison was made between the convex polyhedron strategy dexcribed in [8] and the obstacle time-discretization method described in this work. The strategies were compared in the ability of avoiding obstructions as well as in the computational load needed for the optimizations to be solved. The obstacle discretization method has shown great improvements with respect to the other method. The trajectories are much smoother and reliable. Furthermore, the strategy can manage much faster speeds, always ensuring that the obstacles will be avoided. Also from a computational point of view the advantages are evident. Keeping the same horizon length for both strategies, the obstacle discretization method employs almost hundred times less in terms of time for solving the optimization processes. The overall control was then implemented in a complex model of the quadcopter, that simulates the system dynamics in a very detailed way, thus providing a reliable representation of the system response. The real system behavior differs very little from the predictions made with the model exploited by the MPC, which proves the ability of the control to cope with external forces that have not been modeled, as for example aerodynamic effects. The greatest discrepancies occur for the vertical axis. Indeed, the control showed some difficulties in keeping a reference altitude at high speeds. This may be caused by strong aerodynamic effects. Finally the control unit was simulated on a real-time target computer. Trials were made simulating various clock rates. Real-time feasibility was proved also for low CPU frequencies, thus providing the requirements for a future implementation of the controller directly on board the quadcopter.

Further research based on the presented work can be made. The good results achieved in simulations make the controller ready to be tested on real quadcopters. First, experiments can be done providing informations on positions and velocities of the vehicle through an external motion capture system. Also position (and eventually velocity) of the obstacles could be passed by an external source. The next step can be to develop a unit capable of detecting obstacles directly onboard the quadcopter. This could be provided for example by exploiting cameras or laser scanners, so that the vehicle becomes completely autonomous. Improvements can also be done on the controller itself. Till now, the longitudinal control decreases the speed with maximum deceleration in case the lateral trajectory does not manage to completely avoid a collision. Seen from a global point of view this is not the optimal solution. The obstacle avoidance task, however, can not be performed correctly within the same optimization problem for all three axes, since linear contraints are considered. Thus, more detailed studies can be made on how to formulate the longitudinal optimization problem in order to achieve smoother trajectories along the x-axis. For instance, the optimal longitudinal speed at which to move within a given environment could be estimated a priori given the number, dimension and density of the obstacles along the way.

Bibliography

[1] Lee T., Leok M., McClamroch N. H. "Geometric Tracking Control of a Quadrotor UAV on SE(3)", in *IEEE Conference on Decision and Control*, pp. 1383-1389, 2010.

[2] Mellinger D., Michael N., Kumar V. "Trajectory generation and control for precise aggressive maneuvers with quadrotors" in *The International Journal of Robotics Research*, pp. 664-674, 2012.

[3] Mellinger D., Kumar V. "Minimum Snap Trajectory Generation and Control for Quadrotors" in *IEEE International Conference on Robotics and Automation*, pp. 2520-2525, 2011.

[4] Raffo G.V., Ortega M.G., Rubio F.R. "MPC with Nonlinear H∞ Control for Path Tracking of a Quad-Rotor Helicopter" in *Proceedings of the 17th World Congress The International Federation of Automatic Control*, pp. 8564-8569, 2008.

[5] Raffo G.V., Ortega M.G., Rubio F.R. " An integral predictive/nonlinear $H\infty$ control structure for a quadrotor helicopter" in *Automatica*, pp. 29-39, 2010.

[6] Bemporad A.,Pascucci C.A., Rocchi C. "Hierarchical and Hybrid Model Predictive Control of Quadcopter Air Vehicles", in *3rd IFAC Conference on Analysis and Design of Hybrid Systems*, pp. 14-19, 2009.

[7] Alexis K., Nikolakopoulos G., Tzes A. "Model predictive quadrotor control: attitude, altitude and position experimental studies" in *IET Control Theory and Applications*, pp. 1812-1827, 2012.

[8] Bemporad A., Rocchi C. "Decentralized Linear Time-Varying Model Predictive Control of a Formation of Unmanned Aerial Vehicles" in *50th IEEE Conference on De-*

cision and Control and European Control Conference (CDC-ECC), pp. 7488-7493, 2011.

[9] Mueller M.W., D'Andrea R. "A model predictive controller for quadrocopter state interception" in *European Control Conference (ECC)*, pp. 1383-1389, 2013.

[10] Hehn M., D'Andrea R. "Quadrocopter Trajectory Generation and Control" in *Preprints of the 18th IFAC World Congress*, pp. 1485-1491, 2011.

[11] Alexis K., Nikolakopoulos G., Tzes A. "Switching Model Predictive Attitude Control for a Quadrotor Helicopter subject to Atmospheric Disturbances", in *Control Engineering Practice*, pp. 1195-1207, 2011.

[12] Richter C., Bry A., Roy N. "Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments", in *Proceedings of the International Symposium of Robotics Research*, 2013.

[13] Kim J.H., Shim D.H., Sastry S. "Nonlinear Model Predictive Tracking Control for Rotorcraft-based Unmanned Aerial Vehicles" in *Proceedings of the American Control Conference*, pp. 3576-3581, 2002.

[14] Ferreau H.J., qpOASES User's Manual. http://www.qpOASES.org, 2007-2011.

[15] Bouffard P. *On-board Model Predictive Control of a Quadrotor Helicopter: Design, Implementation, and Experiments*, 2012.

[16] Chen M. *Formation and Flight Control of Affordable Quad-rotor Unmanned Air Vehicles*, 2003.

[17] De Val N., Fuso A. *Model Predictive Control for an Autonomous Vehicle*, 2013.

[18] Huyck B., Callebaut L., Logist F., Ferreau H. J., Diehl M., De Brabanter J., Van Impe J., De Moor B. "Implementation and Experimental Validation of Classic MPC on Programmable Logic Controllers" in *20th Mediterranean Conference on Control and Automation*, pp. 679-684, 2012.

[19] Lapp T., Singh L. "Model Predictive Control Based Trajectory Optimization for Nap-of-the-Earth (NOE) Flight Including Obstacle Avoidance" in *Proceeding of the 2004 American Control Conference Boston*, pp. 891-896, 2004.

[20] Wang L. *Model Predictive Control System Design and Implementation Using MATLAB*, Springer-Verlag London, 2009.

[21] Camacho E. F., Bordons C., *Model Predictive Control*, Springer-Verlag London, 2007.