Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Institut für Informatik
Lehrstuhl VII
Robotik und Telematik
Prof. Dr. Klaus Schilling

Model Predictive Spacecraft Formation Control with Nonlinear Lyapunov Stability Constraints

Bachelorarbeit im Fach Luft- und Raumfahrtinformatik
vorgelegt von

Lisa Sophia Spahn Lundgren

Model Predictive Spacecraft Formation Control with Nonlinear Lyapunov Stability Constraints

Bachelorarbeit im Fach Luft- und Raumfahrtinformatik
vorgelegt von

Lisa Sophia Spahn Lundgren
geboren am 18.01.1997 in Seligenstadt

Angefertigt am
Lehrstuhl für Robotik und Telematik
Bayerische Julius–Maximilians–Universität Würzburg

Betreuer:
Prof. Dr. rer. nat. K. Schilling
Dipl.-Inform. F. Kempf

Abgabe der Arbeit:
24.10.2019

Erklärung

Ich versichere, die vorliegende Bachelorarbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt zu haben.

Würzburg, den 24.10.2019

(Lisa Sophia Spahn Lundgren)

Abstract

A control strategy for spacecraft formations using a **M**odel **P**redictive **C**ontrol (MPC) approach is presented. It enables controlling the relative position and velocity of a spacecraft in a formation. The relative motion dynamics are described using the linear **C**lohessy-**W**iltshire (CW) model, thus the controller is suited for formations flying in nearly circular **L**ow **E**arth **O**rbits (LEO). Constraints based on the Lyapunov stability theorem are imposed on the MPC to guarantee that the formation controller is stable. The control scheme is implemented using the open-source framework GRAMPC in "MATLAB" and "C" language code to facilitate application in embedded systems. Simulations for two different reconfiguration scenarios are performed and the performance and stability of the controller evaluated.

Contents

Chapter 1

Introduction

Over the last two decades, the interest and research in spacecraft formation flying has grown immensely and new fields of application such as earth observation, deep-space observation or communication are constantly developed [**?**].

Recently areas such as networked satellite formations and fractionated spacecrafts have been researched. For instance the missions NetSat [**?**] and YETE [**?**], conducted at the Zentrum für Telematik e.V. and the University of Würzburg, investigate these topics. The motivation behind these missions is to increase the stability and robustness of spacecraft formation missions. This is achieved by providing the possibility of a control-handover in case the on-board data handling (OBDH) of one spacecraft is disrupted. Thereby the mission objective is still pursued, despite the corrupted spacecraft.

That is only possible due to the networked control system (NCS) used on fractionated spacecrafts. On a fractionated spacecraft, the original monolithic system is divided into subsystems which interact wirelessly [**?**]. These subsystems can be mounted on one satellite bus, but they can also be farther apart from each other, as long as the wireless connection is maintained. On software level, each subsystem can be represented by one or more nodes such as sensor nodes, computing nodes for attitude control or data processing nodes. The benefit of this architecture is, that the spacecrafts are able to withstand partial system failure. If one computing node malfunctions, another node that has enough computing capacity could carry out the necessary calculations and send them to the respective receiver. In a networked spacecraft formation, this concept can be extended to one spacecraft remotely controlling another one, whose central OBDH is corrupted [**?**]. However the disadvantages that come with this architecture are the danger of package loss or connection failure and thereby data loss.

A controller that handles the relative motion of the formation members has to cope with these problems as well. As the input and output signals of the controller are shared as data packages through a wireless communication network, it is possible that for instance a sensor node is unable to transmit current measurement data. Hence the control law must keep an alternative data source in reserve. Therefore a Model Predictive Control (MPC) strategy suits this application, as its characteristics naturally provide solutions for the data loss issue of networked spacecraft formations.

1.1 Problem Definition

Motivated by these factors, this thesis aims to propose a control strategy based on the Model Predictive Control approach. This controller shall be able to handle the relative motion of a spacecraft formation. The work is summarized in the following tasks:

- Formulation of the relative motion dynamics
- Formulation of an optimal control problem using the MPC strategy
- Derivation of a Lyapunov controller and application of Lyapunov stability constraints on the MPC
- Implementation in software
- Simulation and evaluation of the Lyapunov based Model Predictive Controller

1.2 Thesis Outline

This section briefly describes the structure and contents of the following chapters of this thesis.

Chapter 2 contains literature research and all necessary background knowledge for this thesis.

Chapter 3 gives the dynamics of the model that is used to describe the relative motion of the formation. Furthermore the mathematical formulation of the MPC problem is given and the state and Lyapunov constraints are derived. The implementation of the proposed controller is described as well.

Chapter 4 presents simulations that were performed to test and assess the established formation controller. The simulation results are analysed and evaluated and unresolved weaknesses of the control law are addressed.

Chapter 5 summarizes the work that was contributed within the scope of this thesis and gives a conclusion and insights on the results. Areas that potentially need further research are suggested.

Chapter 2

State of the Art

In this chapter, a quick state of the art to all topics associated with this thesis including all necessary background information to understand the content of this thesis is given.

2.1 Spacecraft Formation Flying

The term "spacecraft formation flying" (SFF) has been established in the year 2000, when ESA launched their cluster quartet mission of four satellites [**?**] and NASA launched a tandem flying mission with the two satellites Landsat-7 and EO-1 [**?**]. Since then, many more formation missions have been launched or are currently being planned. The objectives of these missions range from earth observation to communication to studies of the environment and more, as these new types of missions provide new functionalities, such as capturing one specific area at one time point from several angles or covering more areas at the same time.

However, additional challenges go along with these new possibilities. For instance if the mission requires two spacecrafts to fly close to each other, this means that a very high control accuracy is necessary to prevent collisions. Furthermore in a formation, the trajectory of each formation member has to be planned with respect to the other spacecrafts. According to a definition by [**?**], spacecraft formation flying is the case when the dynamics of two or more spacecrafts are connected via a shared control law. This means that a spacecraft of the formation tracks a reference point that is relative to another member and hence the control law depends on this member's state as well. The dynamics of a formation can therefore be described by relative motion models, which amongst others have been researched for docking and rendezvous manoeuvres already. The Clohessy Wiltshire Model (CW equations [**?**]) is very popular and commonly used, as the CW equations produce linearised relative motion equations,

assuming that the relative distance between the spacecrafts is small compared to the distance of the spacecrafts to the earth's centre. This is beneficial, as more control schemes for linear systems have been proposed compared to nonlinear systems, which will be discussed further in the next section.

2.1.1 Spacecraft Formation Control

The most studied control approach is the Linear-Quadratic Regulator (LQR) and variations of this technique [**?**]. As indicated by the term "linear", this approach on optimal control can only handle linear dynamics and thereby restricts the choice of which dynamical model is used to describe the motion of the spacecrafts. For non-linear dynamical models, it is possible to apply feedback linearisation or to use state-dependent Riccati equations (SDRE). With SDRE, the nonlinear dynamics equations are factorized in such a way that they are brought into a linear-like form, parameterized by state-dependent matrices which in turn express the nonlinearities of the system [**?**]. H_∞ methods have also been researched, some variations achieve an H_∞ controller by using SDRE equations, but this is just one of several possibilities. However, a drawback of H_∞ control is that the resulting controller usually is unreliable at handling nonlinear constraints.

Adaptive control, sliding mode control and model predictive control enable the use of nonlinear dynamics, yet these approaches are currently not studied as much in the context of spacecraft formation flying as LQR. Although in recent years, more research has been invested in these less common control techniques due to the advantages they have compared to the older conventional control methods [**?**]. For instance, MPC controllers offer a great degree of flexibility in terms of how many inputs and outputs are possible as well as efficient constraint handling of linear and nonlinear constraints. Hence this is the control technique used in this thesis and will be explained further in section 2.2. [**?**]

2.1.2 Formation Dynamics

The motion of the spacecrafts is described using a "Leader and Follower" principle, which is the most researched architecture for spacecraft formations [**?**]. One spacecraft is defined as the leader and follows a defined trajectory. The other spacecrafts are called followers and follow a trajectory that is specified relative to the leader. As a result, the motion control of the follower spacecrafts is simplified to a tracking problem [**?**]. In other literature the same principle is described using different titles,

for example chief and deputy, master and slave or target and chase. The terms chief and deputy are used in this thesis. A benefit of this approach are the intuitive and comprehensible dynamics equations for the relative position and velocity, which are necessary to control the motion of the spacecrafts relative to each other. However, a disadvantage is that if the chief is not able to steadily keep its trajectory at all times, the control of entire formation is corrupted as well. An alternative to prevent this problem is to describe the formation dynamics using a decentralized approach, for instance by using orbital elements [**?**] to specify the trajectories each spacecraft has to follow. With these decentralized approaches on the other hand, keeping up the formation as a whole is not guaranteed, as the members of the formation do not explicitly keep specified relative distances and velocities to each other. [**?**]

2.1.2.1 Reference Frame

The coordinate frame, that is used to express and calculate the formation dynamics in the chief-deputy architecture, is the local-vertical-local-horizontal (LVLH) frame. Figure 2.1 shows a visualization of the LVLH frame and the arrangement of the unit vectors $[\hat{x}\ \hat{y}\ \hat{z}]^T$. The origin is in the centre of the chief spacecraft. The relative distance $\vec{d}$ is defined as the vector pointing from the chief spacecraft in the origin of the LVLH frame to a deputy spacecraft.

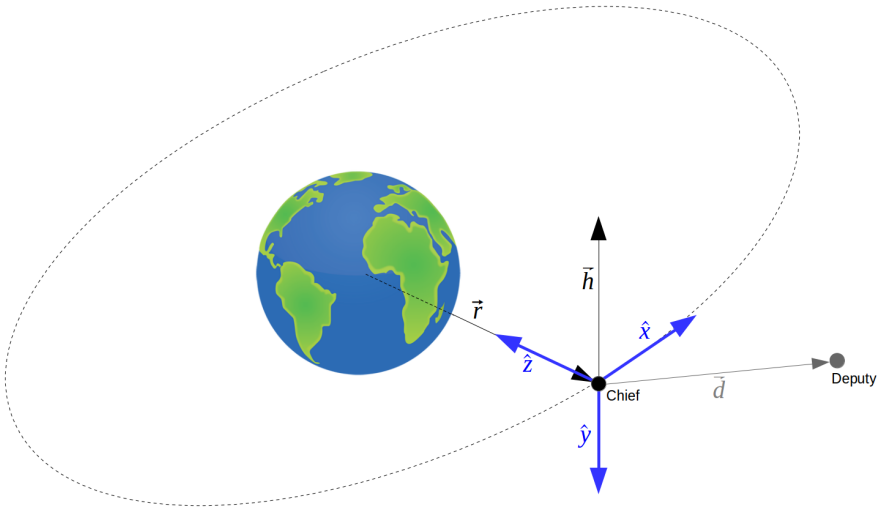

Fig. 2.1: LVLH frame

The unit vector $\hat{x}$ is the tangent to the orbit trajectory and describes the along-track motion. Unit vector $\hat{y}$ is perpendicular to the orbital plane and anti-parallel to the angular momentum vector $\vec{h}$, describing the cross-track motion. The last unit

vector $\hat{z}$ completes the right-handed coordinate system and is pointing nadir, meaning it goes radially inwards from the chief spacecraft to earth's centre. The equations are given by

$$\hat{x} = \hat{y} \times \hat{z} \tag{2.1}$$

$$\hat{y} = -\frac{\vec{h}}{||\vec{h}||} \tag{2.2}$$

$$\hat{z} = -\frac{\vec{r}}{||\vec{r}||}, \tag{2.3}$$

where $\vec{r}$ is the position vector from earth's centre to the chief's centre, $\vec{h}$ is the angular momentum vector and $||\vec{r}||$ and $||\vec{h}||$ are the amounts of the corresponding vector. [**?**]

2.1.2.2 Dynamics Equations

For the dynamical model, the popular Clohessy-Wiltshire model based on the Hill's equations [**?**] is used. Using the LVLH reference frame, the dynamical equations for the relative position $\vec{d} = [x\ y\ z]^T$ of the deputy spacecraft can be expressed as follows

$$\ddot{x} = 2\omega\dot{z} \tag{2.4}$$

$$\ddot{y} = -\omega^2 y \tag{2.5}$$

$$\ddot{z} = 3\omega^2 z - 2\omega\dot{x}, \tag{2.6}$$

where $\dot{\vec{d}} = [\dot{x}\ \dot{y}\ \dot{z}]^T$ is the relative velocity and $\omega = \sqrt{\frac{\mu}{r^3}}$ is the mean motion of the deputy with $\mu = G * M_E$ being the product of the gravitational constant G and the earth's mass M_E.

This relative dynamics model is linearised and therefore well comprehensible. However, it has to be noted that these dynamical equations are simplified, assuming that the chief orbit is circular and that the earth has the shape of a perfect sphere. Furthermore, the model only considers linearised gravitational perturbations, no other disturbances are taken into account. [**?**]

Using the LVLH frame, numerous other models of relative spacecraft dynamics have been published besides the CW model. A notable linear model is amongst others the Tschauner-Hempel model, which is an extension of the CW model to allow for eccentric orbits while still keeping linearised dynamics [**?**]. A very accurate nonlinear model is the Xu-Wang model, describing the relative position and motion

on an eccentric orbit and taking the effect of the oblateness of earth (J_2-perturbation) into account [?].

2.2 Model Predictive Control

Model Predictive Control (MPC) is an advanced control technique that has been studied for more than three decades [?]. It is able to control the dynamics of various multi-variable systems (linear and nonlinear systems) with respect to defined control objectives, while also observing soft or hard constraints if desired. According to [?], MPC is the only control method that has no restrictions regarding process model, cost function and constraints and which also provides a systematic way of describing and implementing the controller's constraint handling. Due to this high degree of flexibility it is possible to control basically any system, which makes MPC very popular and widely used.

However, as the required computational power and time of these online calculations can be relatively high, MPC is often used for industrial process control, ranging from petrochemical plants to the cement industry or oil refineries [?], where the processes develop relatively slow and thereby have a bigger sample time. In recent years, the high-performance processors today have become so powerful, that MPC can be applied on a lot more systems with higher control frequencies, such as robots or servos [?] or even in the aerospace sector and automotive fields [?].

2.2.1 Explanation of the General MPC Method

To describe the basic principle of MPC, it is important to understand the difference to other, more conventional control approaches. In general, the task of a controller is to calculate control inputs which will keep the spacecraft on the trajectory that is followed, in order to get it to a desired position and keep this target state. As it can be seen in e.g. PID-controllers, it is a common approach to plan one step at a time based on the current state, current deviations from the setpoint and control inputs. This means, that the controller always calculates the first step of a trajectory, executes the control action and then repeats this cycle of calculating a "good first step" towards the setpoint. Yet the complete trajectory from initial state to desired state is not taken into account and the trajectory pursued by the "one step at a time" controller is most probably not optimal, as it cannot consider future development in the planning.

MPC on the other hand has a freely definable horizon (so-called prediction horizon

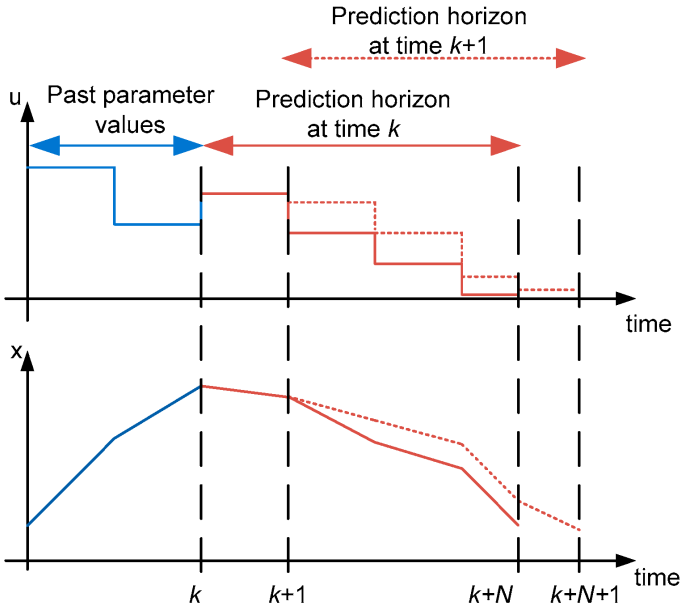

Fig. 2.2: Receding horizon principle of MPC [**?**]

N). For the duration of this prediction horizon, the development of the system is predicted and included in the planning. A sequence of control inputs for the same horizon is calculated, that lets the spacecraft follow a more optimal trajectory to the target state. The first control element of this sequence is then applied on the system and the optimization is repeated with updated measurements and inputs at the next time step, as deviations from the former "optimal trajectory" due to unforeseen disturbances or similar are unavoidable. Hence MPC is often called "receding horizon control". At each time step, the controller predicts the state for the next N time steps and then calculates an array of N_c control actions, where N_c is the so-called control horizon with $N \geq N_c$. These control inputs represent the first N_c steps of an optimal trajectory to the target state. Then the first control action is implemented and at the next time step, this cycle is repeated by shifting the horizon by the duration of one time step and updating the control variables. In figure 2.2, this process is visualized. This method can be transferred to the manner in which humans make a plan to accomplish a certain goal. First, it is roughly estimated how to achieve a objective and what steps are necessary. Then one starts with the first actions but updates the plan every couple of steps, when more knowledge is gathered or if something unexpected happens and disrupts the original plan. This analogy displays the logic and intuitiveness of the MPC approach. [**?**]

In an algorithmic manner, the steps that are followed by a model predictive controller at the time point $t = t_k$ with sample time dt can be described by the following:

1 Prediction: Estimate the state $\tilde{x}(t)$ based on current sensor data and past outputs and control inputs for $t \in [t_k\ , t_k + Ndt)$.

2 Optimization: The MPC calculates an array of optimal control inputs $u(t)$ for $t \in [t_k\ , t_k + N_c dt)$ with respect to the prediction $\tilde{x}$ and a reference setpoint by minimizing a cost function $J(x,u) = \int_{t_k}^{t_k+Ndt} l(x(\tau), u(\tau), \tau) d\tau$.

3 The controller sends the computed control inputs $u(t)$ to the actuators.

4 The actuators execute only the first control input command.

5 Increment k, meaning that the horizon is shifted by one time step dt and the current time changes from t_k to t_{k+1}. Then go to step **1**.

This structure is also shown in figure 2.3, to help visualize and understand the fundamental principle of MPC even better.

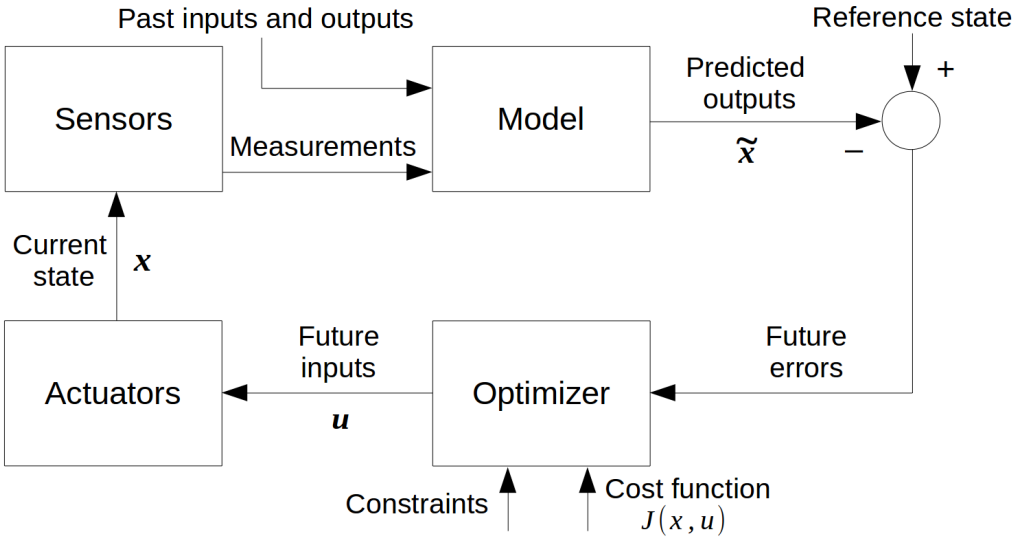

Fig. 2.3: MPC Structure [**?**]

Step **1** corresponds to the output $\tilde{x}$ of the plant model. This is followed by step **2**, where the deviation from the desired state is computed and with these errors as input to the cost function, an optimal input trajectory u is calculated by minimizing this cost function. Step **3** and **4** then represent the data transfer of u to the actuators as well as the execution of the control inputs. Finally step **5** initiates the repetition of this cycle by going to the next time step and thereby providing new sensor measurements to the system model, to compute a new prediction of the state.

2.2.2 MPC – Pro and Contra

The advantages and disadvantages of MPC in general as well as of the implementation of the controller are displayed in table 2.1. [**?**]

Advantages	**Disadvantages**
Can be applied to a big variety of plants: Systems with linear and non-linear dynamics, as well as single input and single output (SISO) and multiple inputs and multiple outputs (MIMO) systems	Prediction model and observer are required (full state estimation), meaning that a good knowledge of the plant is necessary
Very flexible, as it can be personalized to respect limitations on e.g. the state, control acceleration or convergence time by adding constraints	Requires a relatively high amount of computational power and time (fast hardware or "slow" plants are beneficial)
There are several frameworks such as MATLAB's Model Predictive Control Toolbox™ [**?**] or GRAMPC [**?**] (see section 2.2.4.1) that offer a general outline for a basic MPC controller as a starting point	These frameworks facilitate implementing a standard MPC controller, however they can be restrictive, as certain constraints or extensions might not be supported, which make the implementation more complicated

Tab. 2.1: MPC benefits (left) and deficits (right) in tabular form

The main advantage is that in terms of system model and controller design one is practically unrestricted, since linear as well as nonlinear dynamical models and constraints are possible. While this benefit applies to any system that is to be controlled, there is one more great advantage of an MPC controller when applied to the scenario as described in section 1 in the motivation: networked control of a spacecraft formation. In particular of fractionated spacecrafts, meaning that the On-Board Data Handling (OBDH) is distributed into several computing modules, instead of all instruments being connected on a physical bus and data being transferred via physical wires.

One of the biggest challenges of this networked control concept is the possibility of data loss, as the wireless transmission paths between the modules are not completely reliable [**?**]. If for instance the connection between the computing modules and the sensors or actuators is lost during the information exchange, current packages which include sensor data or calculated control actions can not be transmitted and

are therefore not available.

A solution to this problem is to use MPC as a control scheme, as the characteristics of MPC controllers offer alternative data sources in the event of package loss. As a state observer is necessary for MPC, in case of missing sensor data the array of estimated states could be used for the MPC calculations to at least obtain approximated control inputs, until new measurements of the state are available. This also applies to the control actions sent to the actuators, as the MPC always calculates an array of control inputs. If new data between the OBDH and the actuators cannot be transmitted, the next entries of the control input array could be implemented. This ensures, that instead of not executing any control actions at all, it is at least attempted to keep the spacecraft on a trajectory, that does not deviate too much from the optimal trajectory.

2.2.3 Mathematical Formulation of MPC

A generic MPC formulation of a nonlinear optimal control problem according to [**?**] can be described as follows

$$\begin{aligned} \min_{u} \quad & J(x,u) = \int_{t_k}^{t_{k+N}} l\Big(\tau, x(\tau), u(\tau)\Big) d\tau \; + \; F\Big(x(t_{k+N})\Big) \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), u(t)), \\ & u(t) \in U, \; \forall \; t \in [t_k, t_{k+N}], \end{aligned} \tag{2.7}$$

where the nonlinear state $x \in \mathbb{R}^{N_x}$ is considered as a function of the control input $u \in \mathbb{R}^{N_u}$ with N_x and N_u denoting the dimensions of the state and the control variable respectively. A vector U of control inputs is determined as a solution to the optimal control problem so that it minimizes a cost function J, subject to state x and input u over the duration of horizon $N * dt$. The term l is called stage cost and is time-varying, whereas F is a terminal penalty term which shall ensure that the object reaches its goal within the specified horizon. As described in the step-by-step explanation of MPC in 2.2.1, only the first element of U will be carried out by the actuators, then the time is incremented by one time step dt, the horizon is shifted and the calculations are repeated due to the receding horizon principle.

This formulation can be understood as a "basic building block", that has to be adjusted in order to meet certain specifications depending on the plant that is to be controlled. To name a few possibilities, constraints on the state or the control input can be added, for instance by defining inequality constraints $h(x(t), u(t), t) \leq \alpha$ such

as minimum or maximum values of the control acceleration. It is common to choose the stage cost $l(x(t), u(t), t)$ as a quadratic term with tuning parameters, for example by defining two weight matrices W_x and W_u and multiplying these with the respective variable, resulting in the quadratic cost $l(x(t), u(t), t) = x(t)^T W_x x(t) + u(t)^T W_u u(t)$. The terminal cost $F(x(t_{k+N}))$ is an optional term, that can be added for stability. The horizon N can be set to a fixed value, but it is also possible to define it as $N \in [N_{min}, N_{max}]$, meaning that the optimal horizon out of a specified range is chosen at each time step depending on the current state. It is also popular to define a control horizon N_c, that can be smaller as the prediction horizon N as it was explained in 2.2.1. In doing so, computing time and power can be saved when solving the optimal control problem and calculating the control input vector U, because less control actions have to be computed due to the shorter array.

These are only a few possibilities in order to show how flexible MPC is, there are many more ways of varying and adapting the formulation of MPC to fit many different scenarios. Even the choice of iterative method and solving algorithm for the MPC problem strongly influences the resulting controller.[**?**]

2.2.4 Solving MPC

The previous sections addressed all of the characteristics and advantages of MPC, as well as the basic idea of this advanced control technique and how the optimization works in theory. But these features which set MPC apart from other control schemes make solving the optimization problem more challenging. The difficulty lies in the real-time solution of the optimal control problem, as MPC is an online control method [**?**]. In particular the algorithm that is used by the solver and the computational power that is necessary for the numerical calculations can be a great challenge, depending on the system. As mentioned in 2.2, typical application areas in the past were for example slow evolving plants in the chemical industry, because the online requirement could be met due to the lower sampling frequency. However, since the computing power of modern processors has improved immensely during the last years [**?**] and keeps progressing exponentially [**?**], this processing speed and power difficulty of MPC poses less of a problem compared to the implementation of the solver itself.

The most common MPC solving techniques can be divided into linear and nonlinear solvers, meaning that the solving algorithm is formulated either for linear or nonlinear dynamics. In this section, an overview of relevant work and papers for ev-

ery solving method will be given, yet the techniques themselves will not be described in greater detail, as this is not the main topic of this thesis and therefore would go beyond the scope of this subtopic. For that reason, plenty of references and literature will be linked.

For linear MPC problems the optimization problem can be transformed to a quadratic problem (QP), for which many algorithms and even software toolboxes already exist. The easiest solution is to pre-compute the optimal control inputs with respect to all possible linear states, which is an offline optimization technique. Using this explicit MPC approach as a base and extending it by a simple look-up table that is used in the real-time application, the online calculations are kept to a minimum [**?**]. However, this limits the controller to a very small number of states and control inputs, due to memory restrictions as the number of possible scenarios would eventually become too big. Because of that, online MPC solving approaches are generally preferred. Linear online optimization solvers can be split into two groups: first-order approaches and second-order approaches.

First-order algorithms are gradient methods and splitting methods, whereof Nesterov's fast gradient method [**?**] or accelerated gradient methods, such as [**?**] that is based on the Nesterov approach, are especially popular. An advantage is that they are very fast and that a maximum number of iterations necessary to obtain a solution can be set. For systems with hard real-time constraints this is particularly beneficial, as a worst case longest computing time can be formulated with the maximum iterations. Due to the simpler mathematics necessary by this method, implementing and understanding the software is very intuitive. A popular software framework for this technique is for instance FiOrdOs [**?**]. A drawback for the accelerated gradient methods is that they easily become unstable, if the problem is formulated badly, for example if the number of conditions is too large [**?**].

In terms of second-order methods, there are active-set approaches which use the knowledge that at each time step, only a limited set of constraints is active whereas others are not due to the receding horizon principle of MPC. Examples for this method are [**?**] and [**?**]. The more popular second-order approach is the interior-point method, such as the solver described by [**?**]. This technique typically provides very good solutions after only a few iterations. Established software frameworks are for example the CVXGEN framework for fast embedded applications [**?**] or the FORCES implementation by [**?**]. Disadvantages are that usually more algebraic operations are used and hence more processing power and time is required. Furthermore

a maximum number of iterations can not be guaranteed, meaning that the slowest sampling time is difficult to estimate. This makes these second-order methods less suited for really fast changing system or for application in space, as the processors used on spacecrafts are relatively slow.

Now a few options for nonlinear system dynamics are named. A well established algorithm is the continuation/GMRES method by [?] that solves the optimization prblem using a generalized minimum residual method. Very popular is even the ACADO toolkit by [?], that extends the active-set method that was explained above so that it is able to handle nonlinear states as well. Another alternative is the VIATOC toolkit [?], whose optimization algorithm is based on the gradient projection method, however the toolkit is limited to solving discrete-time MPC problems only. The last method given in this section is the recently presented software framework GRAMPC [?], which uses an augmented Lagrangian formulation in combination with a real-time projected gradient method and thereby enables the control of highly nonlinear and dynamical high-frequency systems.

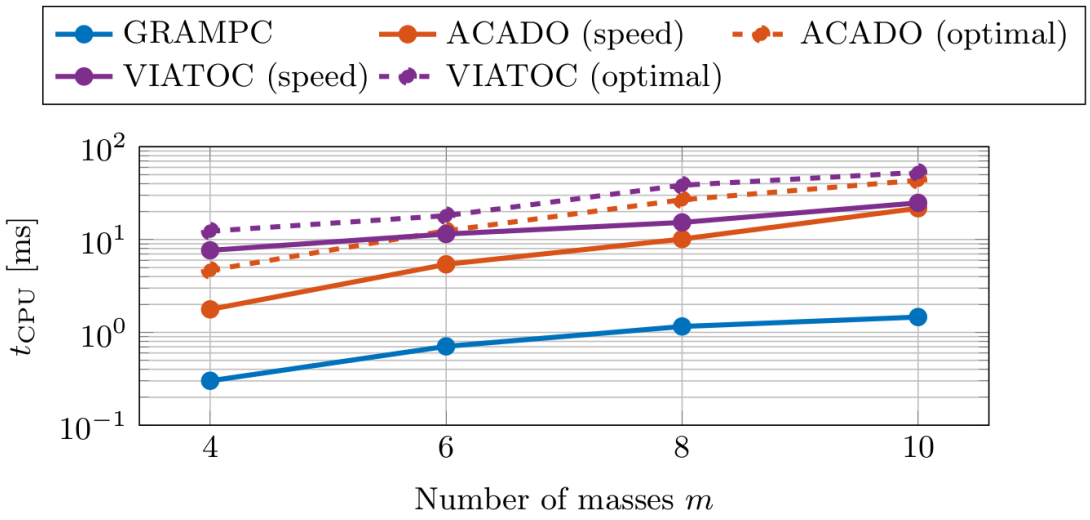

Fig. 2.4: Comparison of the computation time for the same scenario between the frameworks GRAMPC, ACADO and VIATOC [?]

Figure 2.4 shows a benchmark problem of a nonlinear chain with scalable masses as described by [?], which was implemented in the three MPC frameworks ACADO, VIATOC and GRAMPC that were described above. Thereby a comparison of these methods in terms of accuracy and efficiency is possible. One can see that with GRAMPC, the smallest computational times could be achieved at a similar level of accuracy compared to the other nonlinear MPC solvers. Hence GRAMPC is the

framework of choice for this thesis.

2.2.4.1 GRAMPC

The software framework GRAMPC (**GRA**dient based **MPC**) offers functions and templates for nonlinear MPC, providing a MATLAB and Simulink interface while compiling everything in "C"-language code [**?**]. It is open-source and comes with all necessary libraries, which makes it easy to use on any operating system and hence it is suited for embedded systems as well.

MPC problems that can be handled by GRAMPC are formulated as follows

$$\min_{u,p,T} \quad J(u,p,T;x_0) = V(x(T),p,T) + \int_0^T l\Big(x(t),u(t),p,t\Big)dt \tag{2.8}$$

$$\text{s.t.} \quad M\dot{x}(t) = f(x(t),u(t),p,t), \quad x(0) = x_0 \tag{2.9}$$

$$g(x(t),u(t),p,t) = 0, \quad g_T(x(T),p,T) = 0 \tag{2.10}$$

$$h(x(t),u(t),p,t) \leq 0, \quad h_T(x(T),p,T) \leq 0 \tag{2.11}$$

$$u(t) \in [u_{min},u_{max}],\ p \in [p_{min},p_{max}],\ T \in [T_{min},T_{max}]. \tag{2.12}$$

Any problem that is implemented with GRAMPC has to be a subproblem of this formulation, meaning it has to be expressed fully by equations defined in 2.8. Then the problem can be solved using a gradient-based augmented Lagrangian approach.

This means that firstly, the dual problem of the original "primal" optimization problem has to be constructed by reformulating the cost function using "dual variables" or so called "Lagrange multipliers" $\bar{\mu}$ and $\bar{c}$. These multipliers are obtained from the equality and inequality constraints ($\bar{\mu}$) as well as penalty parameters ($\bar{c}$) of the original problem. The motivation for this Lagrangian approach is that generally it is easier to solve the dual problem, which is a convex optimization problem, than the typically non-convex original optimization problem [**?**]. A best lower bound to the primal problem can be calculated by maximizing the augmented Lagrangian optimization problem over the Lagrange multipliers. This results in the following max-min-problem

$$\max_{\bar{\mu}} \min_{u,p,T} \quad \bar{J}(u,p,T,\bar{\mu},\bar{c};x_0) \tag{2.13}$$

$$\text{s.t.} \quad M\dot{x}(t) = f(x(t),u(t),p,t), \quad x(0) = x_0, \quad t \in [0,T] \tag{2.14}$$

$$u(t) \in [u_{min},u_{max}],\ p \in [p_{min},p_{max}],\ T \in [T_{min},T_{max}], \tag{2.15}$$

where the bar symbol denotes variables and functions associated with the augmented

Lagrangian formulation. The outer maximization problem is solved using a steepest ascent approach, whereas the inner minimization problem is solved via a projected gradient method which is based on the gradient methods mentioned in the previous section 2.2.4. In the course of this thesis, these two cascaded solvers are referred to as outer loop and inner loop respectively. For further details and information on the solving approach used by GRAMPC the reader is kindly referred to [?].

2.2.5 Stability of MPC

The last aspect of the theoretical preliminaries that needs to be addressed is the stability of the MPC scheme. While the solution of infinite horizon optimal control schemes guarantees closed-loop stability for unconstrained linear states [?], the systems on which this strategy can be applied are very limited. MPC overcomes these restrictions, however this complicates ensuring stability. MPC uses a finite horizon strategy, because the cost function is minimized over a fixed time span defined by the prediction horizon N. This has the consequence that no explicit control law function can be calculated, like it is done in for instance LQR, meaning that the controller on its own cannot assure stability.

In order to compensate this problem, it is a popular technique to define a terminal cost that penalizes the final state as it was done in equation 2.7 or to add a terminal constraint set, forcing the final state to be in a defined region, for instance the state and therefore also the control action to be zero at the end of the prediction horizon. A combination of these two formulations is also possible, a more extensive overview can be found in [?]. On the downside, these terminal conditions increase the computational effort and time and moreover constrain the possible operating region of the plant. Thereby the feasibility of the problem tends to decrease. Hence other approaches that exclude terminal cost or constraints have been researched, for example by applying constraints on the stage cost, which ensure that cost function decreases monotonically and thereby will drive the state towards zero. Lyapunov functions are commonly used to achieve and prove stability of the controller, this will be explained further in the next section. [?]

2.3 Lyapunov Stability

As mentioned in the previous paragraph, it is necessary to prove convergence of the state towards the setpoint when designing a controller. Furthermore, it is favourable if

the control law manages to keep the state after reaching the desired value, even when it is affected by disturbances. These can be described by the well-known Lyapunov stability theorem, named after the russian mathematician and physicist Aleksandr Mikhailovich Lyapunov.

2.3.1 Principle of Lyapunov Stability

In order to understand the idea behind Lyapunov stability, consider the system

$$\dot{x} = f(x), \tag{2.16}$$

that has an equilibrium point in $x = 0 = x_0$ and a domain $\Omega \subset \mathbb{R}^n$ that contains the equilibrium x_0. If it is possible to find a continuously differentiable function $V : \Omega \to \mathbb{R}$, which satisfies the following conditions

$$V(x) = 0 \text{ for } x = 0 \textbf{ and } V(x) > 0 \text{ for } x \in \Omega \setminus \{0\}, \tag{2.17}$$

$$\dot{V}(x) = 0 \text{ for } x = 0 \textbf{ and } \dot{V}(x) \leq 0 \text{ for } x \in \Omega \setminus \{0\}, \tag{2.18}$$

then the system is stable in Ω and $V(x)$ is a so-called Lyapunov function. [**?**]

One can think of the Lyapunov function as a function, that is only zero at the origin, everywhere else it is positive, which means it is positive definite, as expressed in condition 2.17. More specifically, it is shaped in a way that at any given point in $V(x)$, it will draw x towards the origin. This is formulated in condition 2.18, which says that the derivative of the Lyapunov function has to be negative definite, meaning that for every x, the gradient of $V(x)$ has to point towards the equilibrium. In the origin itself, x can only diverge from the origin if an external force disturbed it, but due to the second Lyapunov stability condition, x would eventually converge towards the origin again. To visualize it further and help understanding this theorem, an intuitive analogy is to imagine a bowl that represents the Lyapunov function and x could be expressed by a marble. No matter where one places the marble, it will always roll towards the deepest point of the bowl, which would be the origin of $V(x)$.

2.3.2 Applications of Lyapunov Stability

The above mentioned Lyapunov principle, that is formulated by the conditions 2.17 and 2.18, can be included in the control law of a MPC to guarantee stability.

As described in section 2.2.5, there are two different ways on how Lyapunov stability

can be implemented in a MPC control law:

1 By choosing the terminal cost and terminal constraint set accordingly, the cost function $J(x, u)$ can be interpreted as a Lyapunov function and therewith the control law guarantees a stable system. In [**?**] this approach is described in greater detail, as well as how the terminal cost and constraints are chosen, how the cost functional then can be rewritten as a Lyapunov function and how the stability of the control can be proven mathematically.

2 The second option is to implement a Lyapunov function in the form of state-dependent inequality constraints instead of using terminal penalties, as described by [**?**]. This is achieved by calculating a feedback control law from a Lyapunov function that uses the same state as the MPC and then at every time step the controller tries to be "better" than the Lyapunov feedback controller. This means, that the controller tries to get the states that are predicted by the MPC control law closer to the setpoint than what the Lyapunov-based control law would achieve. Thereby the MPC will always be able to control the system to the equilibrium, as the Lyapunov controller is proven to be stable, which guarantees stability of the MPC as well.

The resulting control scheme is often called "Lyapunov-based MPC" or "Lyapunov MPC" (LMPC). As controllers utilizing the first technique with a terminal cost and set of constraints typically require more computational power compared to the second approach [**?**], in this thesis the latter LMPC scheme will be implemented, since processing power is a limited resource on satellites. The exact formulation of the Lyapunov function and the LMPC control law will be given in the next chapter, that describes the concept and implementation of the controller.

Chapter 3

Concept and Implementation

In this chapter the concept and implementation of the control law of this thesis is presented, showing the mathematical representation of the problem as well as the translation into the software framework GRAMPC, which was described in the previous chapter.

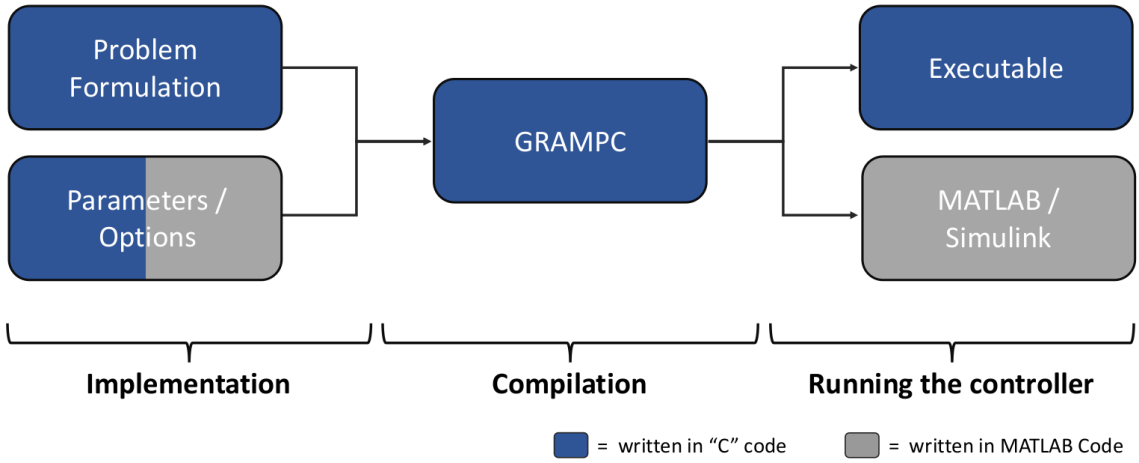

Fig. 3.1: Simplified representation of the structure of a controller realized in GRAMPC [**?**]

Figure 3.1 gives an overview of the overall structure of the model predictive controller. The focus of this chapter is on the two leftmost blocks. The compiling and linking of the code is already provided by GRAMPC and will therefore not be discussed in greater detail. However, it is interesting to note that GRAMPC enables the combination of MATLAB and "C" language, in order to provide both the intuitive graphical interface of MATLAB as well as the speed and suitability for embedded systems of "C" code. Hence in this chapter, at first a conceptual overview of the implemented components is given. Thereafter follows the mathematical formulation and implementation of the system dynamics. Then the derivation of the Lyapunov

function and constraints is discussed, as well as the final MPC formulation and how these equations are implemented in GRAMPC. Lastly it is described, how the different components interact and how the controller works.

3.1 Program Overview

This section gives an overview of the methodology and the implementation of the controller. Figure 3.2 shows the controller components more detailed, as well as the name of the functions and files where they are implemented.

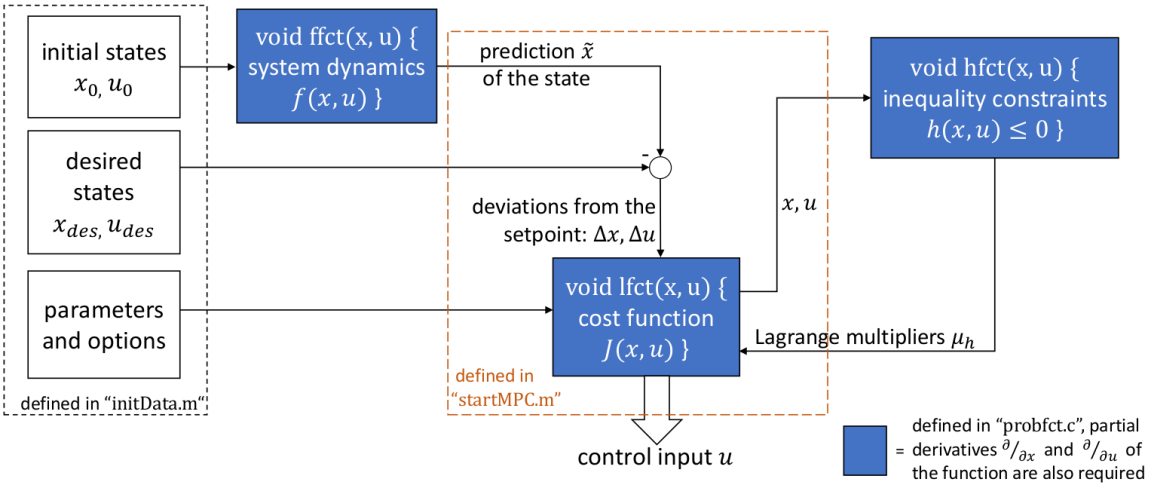

Fig. 3.2: Detailed overview of the MPC components

The user input is made of the initial values, desired setpoints and other parameters such as weights or algorithmic options for tuning. It is defined in the file "initData.m" and is written in MATLAB syntax. The initial states serve as input for the function which calculates the predicted states from the system dynamics and is called `ffct`. The output is the prediction $\tilde{x}$ of the state and in the next step, the difference between the current state and control input and the desired values is determined, which goes into the cost function as input variables. The function `lfct` defines the cost function that is minimized. For easier tuning, the weight matrices are set as parameters in "initData.m" but they are passed to the function `lfct` and hence can be accessed there. Other options that are relevant for the control law and solving the optimization problem, e.g. if constraints are handled as hard constraints or as soft constraints, are defined in the same file as the weight matrices.

The constraints themselves are implemented in another function called `hfct` and are defined as inequality constraints. The function returns Lagrange multipliers,

which act as penalty parameters in the outer multiplier loop of the solver for the optimization problem. When the minimization has succeeded and the constraints are satisfied, the control acceleration u is returned as a result. The final predicted state $\tilde{x}$ and control input u are then led back into the dynamics function and the loop is repeated. This is not visualized in figure 3.2 in order to keep the clarity of the diagram. Thus only the first control loop iteration is shown.

The execution of the control loop, such as calling the correct functions and initializing and passing the variables, is implemented in the file "startMPC.m". It is also written in MATLAB code and the basic structure of the control loop is already provided by GRAMPC in a template. The additions and changes that are made will be explained in the later sections. The functions `ffct`, `lfct` and `hfct` are implemented in the file "probfct.c" and are written in "C" language, as opposed to the other two files. Furthermore, the partial derivatives of the functions with respect to state x and control input u are derived and implemented in the same file, as they are necessary for the gradient-based solving algorithm. In the following sections, all of these mentioned components of the controller are explained in greater detail.

3.2 Relative Dynamic Model

As described in the previous chapter in section 2.1.2.2, the dynamics are expressed as a relative model. The popular CW-equations that were defined in equations2.4 to 2.6 are utilized to implement simple linearised dynamics of the formation. In combination with a formation controller that adds the control input $\vec{u}$ to the equation, the dynamics can be transferred into the state-space form

$$\dot{\vec{x}} = A\vec{x} + B\vec{u}, \tag{3.1}$$

where $\vec{x}$ denotes the state vector and $\vec{u}$ denotes the control input calculated by a controller. [**?**]

The state and control acceleration are defined as follows

$$\vec{x} = \begin{pmatrix} \vec{d} \\ \dot{\vec{d}} \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \quad \text{and} \quad \vec{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} \frac{F_x}{m} \\ \\ \frac{F_y}{m} \\ \\ \frac{F_z}{m} \end{pmatrix}, \tag{3.2}$$

with $\vec{d}$ and $\dot{\vec{d}}$ denoting the relative position and relative velocity of a deputy spacecraft with respect to its chief, F_i representing the LVLH frame components of the control force generated by the controller and m denoting the mass of the deputy spacecraft. The dynamical equations 2.4 to 2.6 thereby change to the following formulas

$$\ddot{x} = 2\omega\dot{z} + u_x \tag{3.3}$$

$$\ddot{y} = -\omega^2 y + u_y \tag{3.4}$$

$$\ddot{z} = 3\omega^2 z - 2\omega\dot{x} + u_z, \tag{3.5}$$

resulting in the following state-space matrices for A and B

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2\omega \\ 0 & -\omega^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3\omega^2 & -2\omega & 0 & 0 \end{pmatrix}, \tag{3.6}$$

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0_{3\times3} \\ I_{3\times3} \end{pmatrix}, \tag{3.7}$$

with the mean motion $\omega = \sqrt{\frac{\mu}{r^3}}$.

By substituting these two matrices for A and B in the state-space equation 3.1, the dynamics of the system are implemented component by component in the `ffct`

function from figure 3.2. This enables the controller to make predictions of the current state and to estimate future trajectories based on the relative dynamics model, which is necessary for calculating the optimal control input.

3.3 Unconstrained MPC

Following the diagram in figure 3.2, the system dynamics model that was described in the last section returns the predicted state, which is compared to the desired value. The state difference, which is the difference between current and desired control acceleration, is passed on to the cost function.

As described in section 2.2.5, adding terminal costs in the objective function to guarantee stability of the controller usually increases the required computational effort. Since processing power is a very limited resource on satellites, this control law uses time-dependant Lyapunov constraints instead of a terminal penalty. Hence the integral cost that is used for the MPC in its unconstrained form, is defined as follows

$$\begin{aligned} &\min_u \quad \frac{1}{2}\int_{t_k}^{t_{k+N}} \tilde{x}(\tau)^T Q\tilde{x}(\tau) + u(\tau)^T Ru(\tau)\ d\tau \\ =\ &\min_u \quad \int_{t_k}^{t_{k+N}} \frac{1}{2}\Big(\tilde{x}(\tau)^T Q\tilde{x}(\tau) + u(\tau)^T Ru(\tau)\Big)\ d\tau, \end{aligned} \tag{3.8}$$

$$Q = \begin{pmatrix} q_1 & 0 & \cdots & 0 \\ 0 & q_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & q_6 \end{pmatrix} \text{ and } R = \begin{pmatrix} r_1 & 0 & 0 \\ 0 & r_2 & 0 \\ 0 & 0 & r_3 \end{pmatrix}, \tag{3.9}$$

which is a quadratic cost function with the weighting matrices $Q = diag(q_1, q_2, ..., q_6)$ and $R = diag(r_1, r_2, r_3)$ and $q_i, r_i \in \mathbb{R}_0^+$ for individually penalizing the state and the control input respectively.

Comparing this to the general MPC formulation in equation 2.7 from the second chapter, the stage cost $l(x(t), u(t), t)$ can be identified as the formulation within the integral in equation 3.8. The cost function is then implemented in the function `lfct` by the scalar term

$$\begin{aligned} &l\ \Big(x(t), u(t), t\Big) \\ =\ &\frac{1}{2}\ \Big(\tilde{x}(t)^T Q\tilde{x}(t) + u(t)^T Ru(t)\Big). \end{aligned} \tag{3.10}$$

So far, this unconstrained formulation of the MPC is a subproblem of the generic MPC formulation given in 2.8. This means, that it can be controlled using GRAMPC. Now it has to be ensured, that the state and control input constraints are expressed in a way that is compatible with the GRAMPC formulation, for instance as inequality constraints $h(x(t), u(t), t) \leq 0$.

3.4 Constrained MPC

This section describes all constraints which are applied on the system in the form of state- and time-dependant inequality constraints, meaning that they are calculated and evaluated by the MPC solver at each time step.

3.4.1 Constraints on the State and Control Input

Depending on the system, it can be necessary to formulate system-dependent constraints, such as the border of a robot arm's workspace. In space applications, the propulsion of the satellite imposes constraints on the control acceleration and the rate of the control input by a maximum and minimum value. The position and velocity of a satellite can be restricted by orbit or mechanical specifications. As MPC enables constraint handling, these minimum and maximum limits are implemented in the form of inequality constraints $h(x, u, t) \leq 0$ in the function `hfct`.

The constraint function is derived as follows:
The relative position vector $\vec{d} = [x, y, z]$ of a deputy spacecraft is limited by the minimum values $[x_{min}, y_{min}, z_{min}]$ and maximum values $[x_{max}, y_{max}, z_{max}]$. This is formulated by the following inequality equations

$$\begin{aligned} x_{min} \leq x \leq x_{max} \\ y_{min} \leq y \leq y_{max} \\ z_{min} \leq z \leq z_{max} \ , \end{aligned}$$

which can be rewritten as follows

$$\begin{pmatrix} x & \geq & x_{min} \\ x & \leq & x_{max} \\ y & \geq & y_{min} \\ y & \leq & y_{max} \\ z & \geq & z_{min} \\ z & \leq & z_{max} \end{pmatrix} \Leftrightarrow \begin{pmatrix} -x & \leq & -x_{min} \\ x & \leq & x_{max} \\ -y & \leq & -y_{min} \\ y & \leq & y_{max} \\ -z & \leq & -z_{min} \\ z & \leq & z_{max} \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} -x & + & x_{min} & \leq & 0 \\ x & - & x_{max} & \leq & 0 \\ -y & + & y_{min} & \leq & 0 \\ y & - & y_{max} & \leq & 0 \\ -z & + & z_{min} & \leq & 0 \\ z & - & z_{max} & \leq & 0 \end{pmatrix} \Leftrightarrow \begin{pmatrix} -x & + & x_{min} \\ x & - & x_{max} \\ -y & + & y_{min} \\ y & - & y_{max} \\ -z & + & z_{min} \\ z & - & z_{max} \end{pmatrix} \leq 0$$

$$\Leftrightarrow h(x, y, z) \leq 0 \ . \tag{3.11}$$

Analogously, boundaries for the relative velocity $\dot{d}$ of the deputy spacecraft are defined. With this formulation, the state constraints $h(d, \dot{d}) = h(x) \leq 0$ are implemented in the controller.

Unlike the state x and control input u, the rate of the control input $\dot{u}$ is not explicitly declared in the system dynamics. However in a discrete-time system like it is given here, $\dot{u}$ can be calculated as the change between the current and the last value of the control acceleration. Hence for $\dot{u}_{t_k} = u_{t_k} - u_{t_{k-1}}$ the inequality constraints $h(0) \leq 0$ are derived in the same way as it is done in 3.11, resulting in the joint inequality function

$$h(x, u) \leq 0 \ . \tag{3.12}$$

3.4.2 Lyapunov Stability Constraints

As described in section 3.3, Lyapunov-based constraints are added to guarantee stability of the controller rather than a terminal penalty term. The principle of Lyapunov stability is discussed in detail in section 2.3 in the previous chapter. According to [**?**], the Lyapunov stability theorem can be applied to the control law by modifying the

problem formulation as follows

$$\begin{aligned} \min_{u} \quad & J(x,u) = \frac{1}{2}\int_{t_k}^{t_{k+N}} \tilde{x}(\tau)^T Q\tilde{x}(\tau) + u(\tau)^T Ru(\tau)d\tau \\ \text{subject to} \quad & \dot{\tilde{x}}(t) = f\Big(\tilde{x}(t), u(t)\Big)\ ,\ \tilde{x}(t_k) = \hat{x}(t_k) = x(t_k),\ u(t) \in U \\ & V(\tilde{x}(t)) \leq V(\hat{x}(t))\ \forall\ t \in [t_k, t_{k+N}], \end{aligned} \tag{3.13}$$

where $\hat{x}$ denotes the estimated state of the system when the control law $u = h(\hat{x}(t))$ from a sole Lyapunov controller is applied. This means that, for the duration of one prediction, the MPC tries to keep the state closer to the setpoint than the Lyapunov controller is able to achieve. Since the Lyapunov controller is proven to be stable, the controller using the MPC approach is then stable as well.

In order to achieve stability of the controller, a Lyapunov function is derived. It has to fulfil the two conditions 2.17 and 2.18, as explained in section 2.3. For reasons of clarity, vector arrows and time-dependencies are not explicitly denoted in the next equations.
The state is given by

$$\dot{x} = Ax + Bu. \tag{3.14}$$

It is assumed that a feedback control matrix K exists, so that the control feedback $u = -Kx$ is able to regulate the system. The matrix K can be derived by solving a LQR problem that is subject to the same system dynamics as the MPC. Using this matrix K, the state-space model given by equation 3.1 can be rewritten as follows

$$\begin{aligned} \dot{x} =& Ax - BKx \\ =& (A - BK)x = \tilde{A}x. \end{aligned} \tag{3.15}$$

For the Lyapunov function, a quadratic function is chosen

$$V(x) = x^T Px, \tag{3.16}$$

where P is a symmetric and positive definite matrix, so that $V(x)$ is positive definite as well and thereby condition 2.17 is fulfilled.

The derivative of the Lyapunov function $V(x)$ has to be negative definite. It is

given by

$$\begin{aligned}\dot{V}(x) =& \frac{\partial V(x)}{\partial x} = \frac{\partial}{\partial t}(x^T P x) \\ =& \dot{x}^T P x + x^T P \dot{x},\end{aligned} \tag{3.17}$$

which is proven to be negative definite by substituting $\dot{x}$ with formulation 3.15. that was derived earlier. Thereby $\dot{V}(x)$ changes as follows

$$\begin{aligned}\dot{V} &= \tilde{A}^T P x + x^T P \tilde{A} \\ \Leftrightarrow \dot{V} &= x^T \tilde{A}^T P x + x^T P \tilde{A} x \\ \Leftrightarrow \dot{V} &= x^T (\tilde{A}^T P + P \tilde{A}) x\end{aligned} \tag{3.18}$$

with $\tilde{A} = A - BK$.

$\dot{V}(x)$ is negative definite, if $\dot{V}(x) = 0$ for $x = 0$ and $\dot{V}(x) < 0$ for $x \neq 0$. The first condition is true, the second one can be proven true if matrix P can be found so that $\dot{V}(x) = -Q$ for an arbitrary positive definite matrix Q

$$\begin{aligned}&\dot{V}(x) + Q = 0 \\ &x^T (\tilde{A}^T P + P \tilde{A}) x + Q = 0 \\ &\Rightarrow \tilde{A}^T P + P \tilde{A} + Q = 0.\end{aligned} \tag{3.19}$$

By solving equation 3.19 for P with matrix Q chosen as $Q = 10^{-3} * I_{3x3}$, the Lyapunov function is then obtained according to equation 3.16 and stability of the system can be guaranteed.

The constraints are implemented in the `hfct` function in the form of

$$V(\tilde{x}(t)) - V(\hat{x}(t)) = h(x) \leq 0, \tag{3.20}$$

where $\tilde{x}$ denotes the state that is controlled by the model predictive controller and $\hat{x}(t)$ is the Lyapunov controller state. They are merged together with the inequality constraints defined in equation 3.12. This means that the inequality constraint function $h(x, u)$ is nonlinear, since the Lyapunov function is a quadratic function.

Conclusively, figure 3.3 illustrates exactly which components of the controller were implemented.

One can see that in GRAMPC, the controller can be realized in mainly three files.

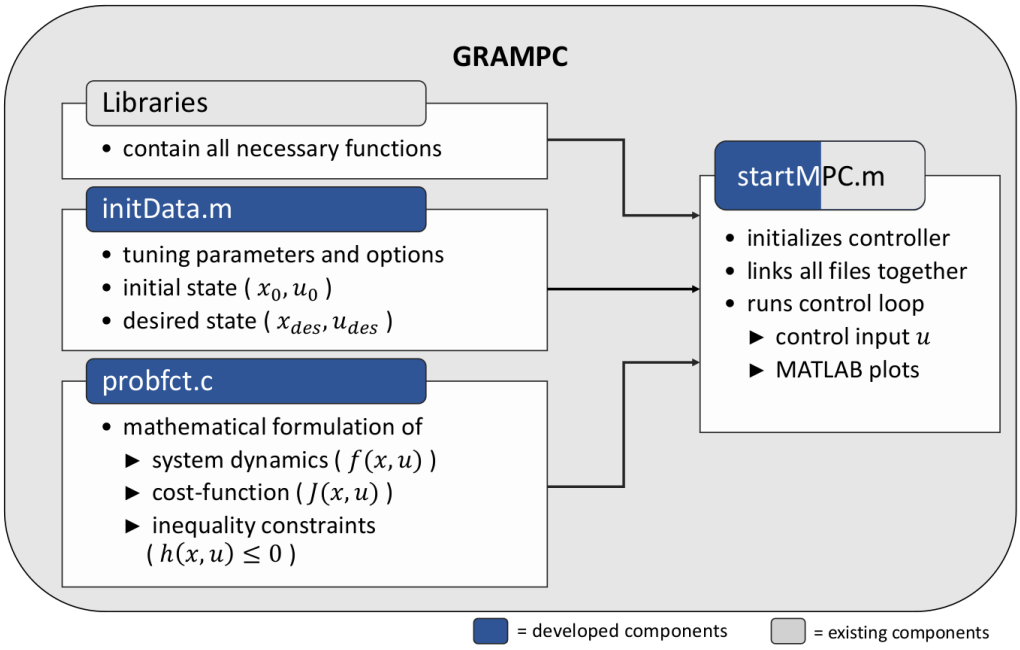

Fig. 3.3: Simplified overview of the MPC components

1 "initData.m":

This file represents the user input. Initial state and desired setpoints have to be defined. Additionally, it is possible to set a variety of parameters, such as the weights of the cost function, the prediction horizon or the constraints on the control input. There are also options for adjusting the solver of the cost function, for instance by selecting which numerical iteration method is used or how tolerant constraints are handled.

2 "probfct.c":

In this file, the mathematical basis for the controller is defined. This means that the dynamical model of the system is implemented to calculate the predicted state. Furthermore, the cost function that is minimized has to be given and finally all constraints which shall be considered by the controller have to be defined.

3 "startMPC.m":

This file is responsible for initializing all required variables and running the control loop to calculate a control input u. The control loop is executed by calling the necessary functions using the libraries provided by GRAMPC and combining the contents of the other two files. It is also possible to plot data of the controller such as the cost, the position, the applied control acceleration

or penalties in case constraints were violated. This facilitates analysing and assessing the controller. GRAMPC provides a template for a standard MPC loop that includes necessary initializations and function calls. Slight modifications of this basic template are necessary, such as resetting initial values for the Lyapunov and $\dot{u}$ constraints, or by adding custom plots for the Lyapunov function and states.

Chapter 4

Evaluation and Discussion

This chapter concerns the evaluation of the developed model predictive controller in a simulation. Based on two different simulated scenarios, the performance in terms of accuracy and convergence is evaluated and secondly the stability of the controller is considered. Lastly follows a discussion of the results as well as of critical points and difficulties that arose during the simulation of the controller.

4.1 Controller Parametrization

The simulation is based on the NetSat mission [**?**], which is a current research project at the Zentrum für Telematik e.V. NetSat is a spacecraft formation consisting of four pico-satellites with low-thrust actuators. Figure 4.1 shows an animation of the NetSat mission. In the simulation the relative position and velocity of one pico-satellite with respect to its chief satellite will be controlled.

In [**?**], typical system characteristics, such as restrictions on the system variables for missions similar to the NetSat mission, are defined. Based on that, the following maxima and minima for the control acceleration and rate of control input are specified:

$$|u| \leq 6.15384615384615\ \mu Nm \tag{4.1}$$

$$|\dot{u}| \leq 3.07692307692307\ \frac{\mu Nm}{s}. \tag{4.2}$$

Since these low-thrust actuators are not so powerful, the sample time has to be chosen accordingly in order to provide enough time for the actuators to accelerate the satellite and change its relative position. Hence the sample time dt is set to $240s$, which is relatively big compared to what is possible computationally [**?**].

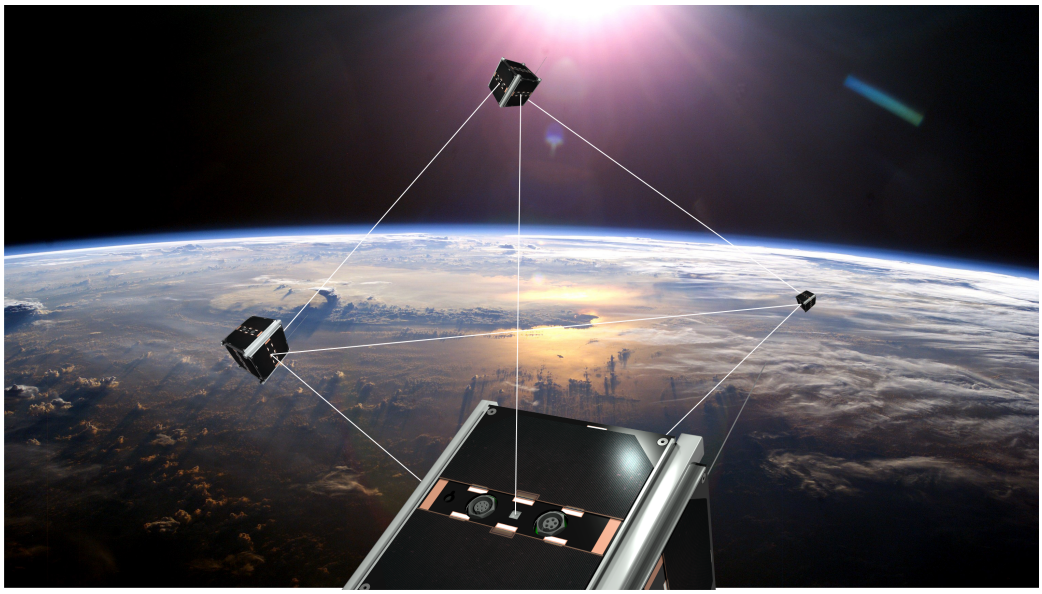

Fig. 4.1: Animation of the NetSat mission [?]

The orbit for the simulation is chosen to be a circular low earth orbit at $h_{orbit} = 500km$ without any modelled disturbances. This rather simple orbit choice is due to the linear dynamics model, as for eccentric or perturbed orbits a more complex nonlinear model is necessary. Refer to section 2.1.2.2 in the second chapter for more details about the formation dynamics model and other alternatives to the linear CW-model.

With this orbit and model choice the following restrictions on the state $x = [d, \dot{d}]$ are defined:

$$|d| \leq 2000\ m \tag{4.3}$$

$$|\dot{d}| \leq 2\ \frac{m}{s}. \tag{4.4}$$

The orbital period T_{orbit} of one revolution around the earth is $5676.8s$, meaning that there are $\frac{T_{orbit}}{dt} = 23.65$ time steps per orbit. The simulation time T_{sim} is set to 4 orbits, so that the controller has enough time to finish its manoeuvres despite the relatively inefficient actuators.

Lastly, scaling factors α for all control variables are defined, to ensure that all variables are of the same order of magnitude when plugged into the cost function. The scaled variables are calculated as follows

$$x_{scaled} = \frac{1}{\alpha_x} x, \tag{4.5}$$

with the scaling factors $\alpha_d = 1.0$ on the position, $\alpha_{\dot{d}} = 1.0 * 10^{-3}$ on the velocity and $\alpha_u = 1.0 * 10^{-4}$ on the control input.

Table 4.1 summarizes all of the above mentioned parameters in one table.

Variable	Value
dt	240 $[s]$
h_{orbit}	500 $[km]$
T_{orbit}	5676.8 $[s]$
T_{sim}	4 $[T_{orbit}]$
$\alpha_{x,y,z}$	1.0
$\alpha_{\dot{x},\dot{y},\dot{z}}$	$1.0 * 10^{-3}$
α_u	$1.0 * 10^{-4}$
d_{min}, d_{max}	-2000, 2000 $[m]$
$\dot{d}_{min}, \dot{d}_{max}$	-2, 2 $[m/s]$
u_{min}, u_{max}	-6.1539, 6.1539 $[\mu Nm]$
$\dot{u}_{min}, \dot{u}_{max}$	-3.0769, 3.0769 $[\mu Nms^{-1}]$

Tab. 4.1: Parameters and settings of the controller

4.2 Docking Manoeuvre

The first scenario that is simulated is a rendezvous manoeuvre, where the deputy spacecraft flies towards the chief and aims to obtain the same position and velocity as the chief satellite. This means, that the initial and desired values for the state x and control input u are defined as follows

$$x_0 = [2.0, 1.0, -1.0, 0.0, 0.0, 0.0]\ m \tag{4.6}$$

$$x_{des} = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]\ \frac{m}{s} \tag{4.7}$$

$$u_0 = [0.0, 0.0, 0.0]\ \mu Nm \tag{4.8}$$

$$u_{des} = [0.0, 0.0, 0.0]\ \frac{\mu Nm}{s}, \tag{4.9}$$

where index $''0''$ indicates that it is a initial value and index $''des''$ denotes a set-point. In addition to the parameters defined in the previous section in table 4.1, the remaining settings and tuning parameters are given in table 4.2.

A quick description of each of these parameters is given in the following list:

PredHor Number of steps for the prediction horizon.

Nhor Number of steps for the system integration.

Variable	Value
PredHor [dt]	20
Nhor	100
MaxGradIter	30
MaxMultIter	5
ConstraintTol	[0 ... 0]
Q	diag(10, 10, 10, 0, 0, 0)
R	diag(1, 1, 1)

Tab. 4.2: Parameters and settings for simulation of a docking manoeuvre

MaxGradIter Maximum number of gradient iterations.
MaxMultIter Maximum number of augmented Lagrangian iterations.
ConstraintTol Thresholds for the inequality constraints (0 = hard constraint).
Q Weighting matrix for the state.
R Weighting matrix for the control input.

For further tuning options or a more detailed description of the above mentioned settings, the reader is kindly referred to [**?**].

4.2.1 MPC with Lyapunov Stability Constraints

At first, the controller that is described in the previous chapter in equations 3.13 with all its constraints is simulated. The Lyapunov constraints are implemented in the form of hard constraints, as stability is a very important aspect of the controller and hence this constraint should be fulfilled at all times. The same applies to the control input constraints. They cannot be violated since the actuators shall not exceed these limits. The weighting matrix Q is chosen as the value 10 for the position and 0 for the velocity vector. The velocities are not weighted because they are very small compared to the position due to the slow actuators, meaning that they are almost zero anyway, and therefore the focus of the controller lies on the positions.

Figure 4.2 shows the development of the state and control input over the course of four orbits. One can see that after approximately two orbits, the satellite reaches its setpoint. It has to be noted, that the states in y-direction oscillate more and converge slower than the states along the x- and z-axis. This can be explained by equations 3.3 where one can see, that the y-component of the movement is decoupled from the x- and z-component of the dynamics model. By increasing the cost of the position along the y-axis it can be attempted to fix this problem. As a results the

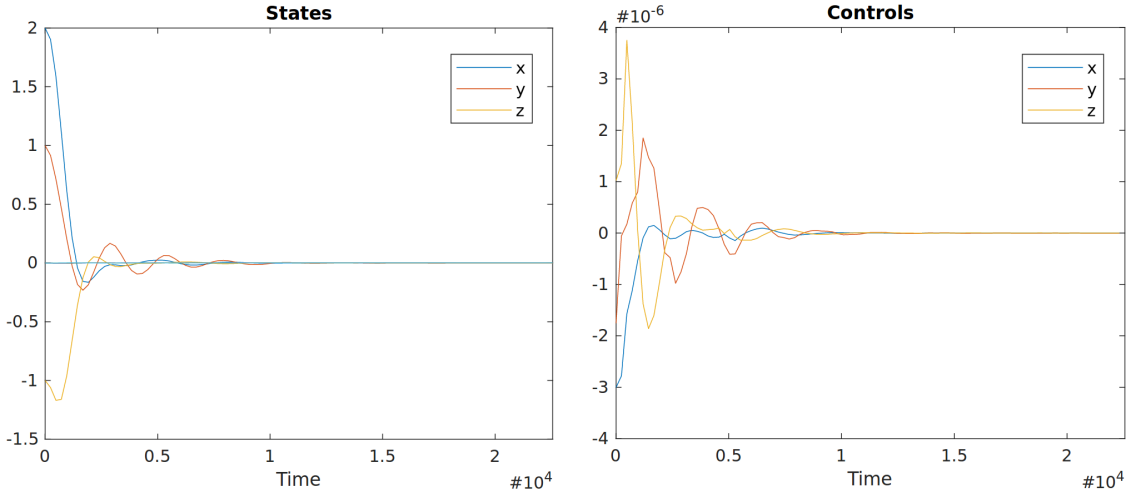

Fig. 4.2: Relative positions and velocities of the satellite (left) in [m, m/s] and the applied control acceleration (right) during the docking manoeuvre

satellite reaches the desired position in y-direction faster, however this has the effect that the x- and z-positions are reached later in return. Therefore it is decided to keep this configuration given in table 4.2, but the controller can be tuned differently to meet other system requirements. The last thing to be concluded from this figure is that the control acceleration never exceeds $4.0\mu Nm$, meaning that the constraints on the control input are fulfilled at all times. The relative velocities appear to be zero since they stay within a magnitude of $10^{-3}m/s$, but in figure 4.4 one can see their development more detailed.

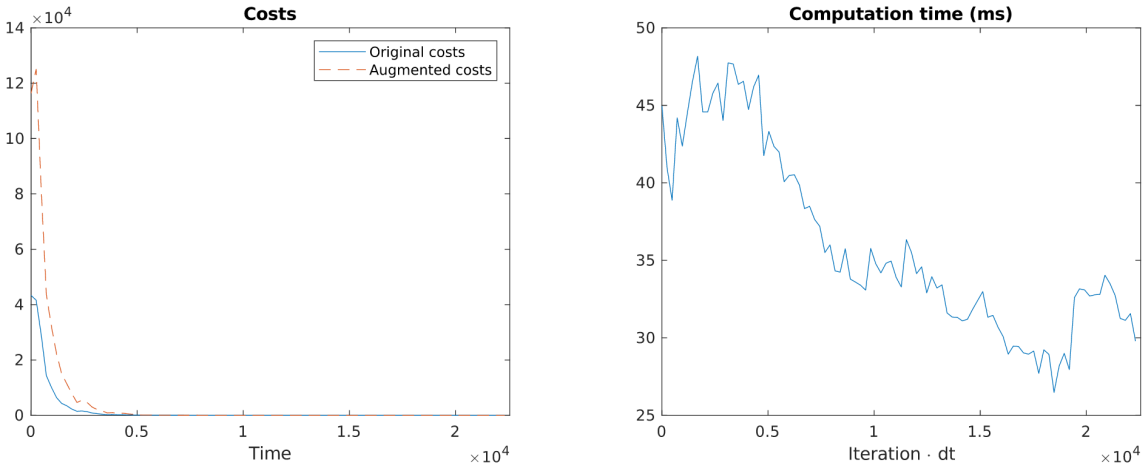

Fig. 4.3: Costs (left) and computational time per iteration (right) during the docking manoeuvre

Figure 4.3 shows the course of the cost function, which decreases nicely with pro-

gressing time. The computational time per time step in milliseconds is displayed to the right, showing how fast the calculation of the controller is with less then 50 ms per iteration.

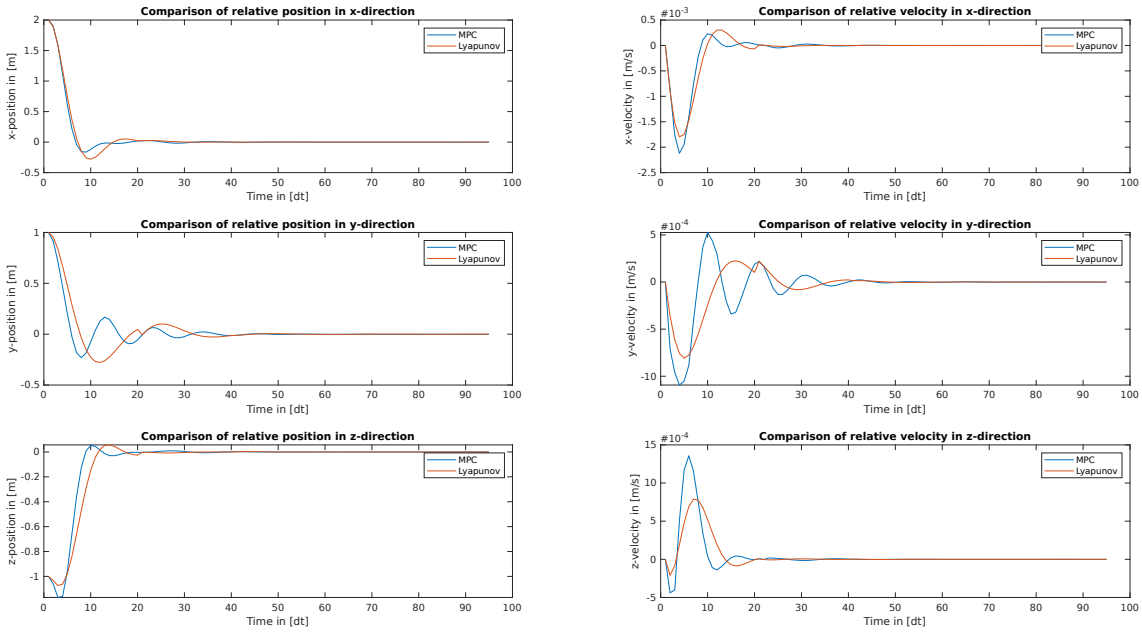

Fig. 4.4: Comparison of relative position and velocity of MPC and Lyapunov controller during the docking manoeuvre

In figure 4.4 the comparison between MPC and Lyapunov controller for each component of the state vector is depicted, showing the course of the velocity more clearly compared to figure 4.2. Furthermore, one can see how the state of the Lyapunov controller is reset after the cycle of one prediction horizon every 20 time steps, as it is defined in the initial conditions in equation 3.13.

The last graphs are depicted in figure 4.5 and show the Lyapunov function and its derivative of the state, once controlled by the MPC and then by the Lyapunov controller respectively. The MPC achieves to stay below the Lyapunov controller at most of the time. The few times where it does not achieve this correspond to the slight oscillations of the relative position along the y-axis, which were discussed previously in figure 4.2.

4.2.2 MPC without Lyapunov Stability Constraints

Now for the same scenario the Lyapunov constraints are disabled, in order to test their functionality and effectiveness.

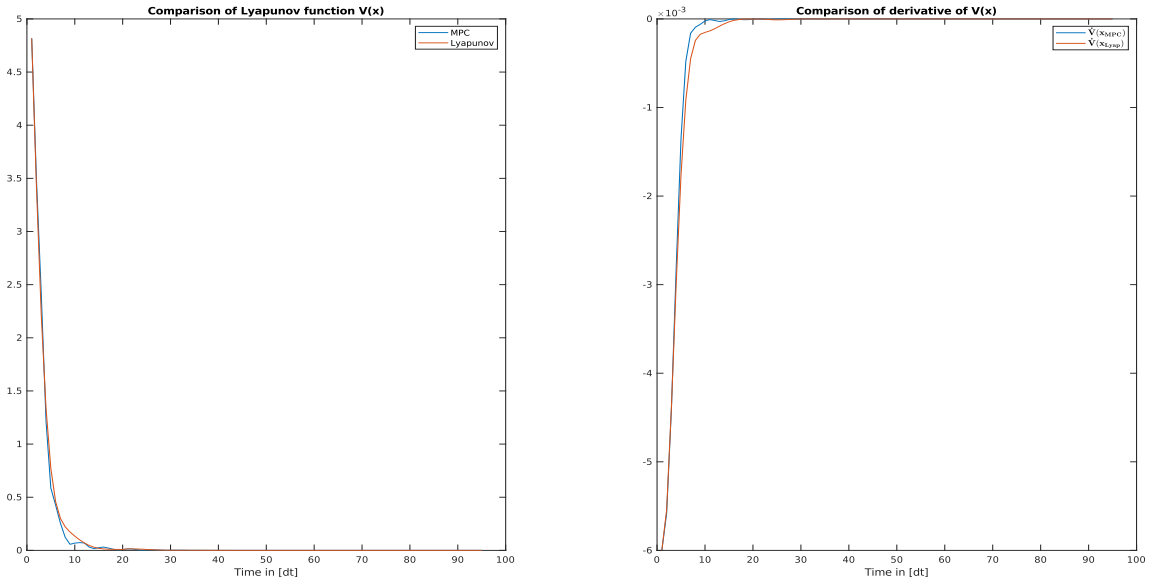

Fig. 4.5: Comparison of Lyapunov function and its derivative of MPC and Lyapunov controller during the docking manoeuvre

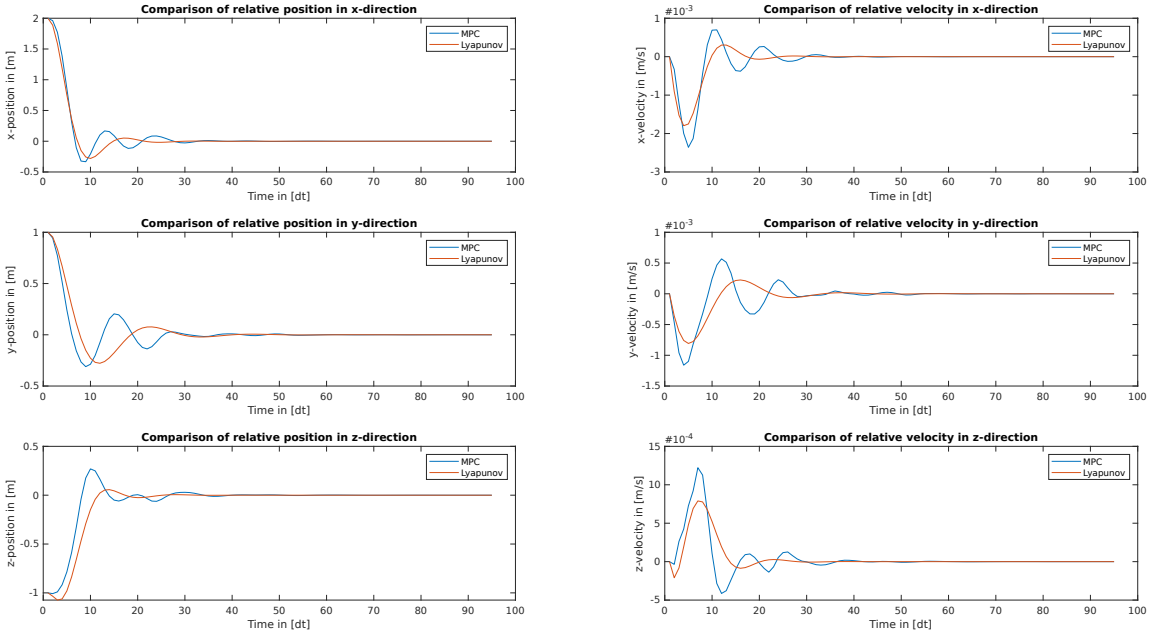

Fig. 4.6: Comparison of relative position and velocity of unconstrained MPC and Lyapunov controller during the docking manoeuvre

In figure 4.6 one can see how the MPC and Lyapunov controller perform independently on their own. Although the MPC still manages to drive the states to the setpoint, it takes approximately half an orbit to an orbit longer for the states to converge to zero. Especially around the x-axis and z-axis, increased overshoot and

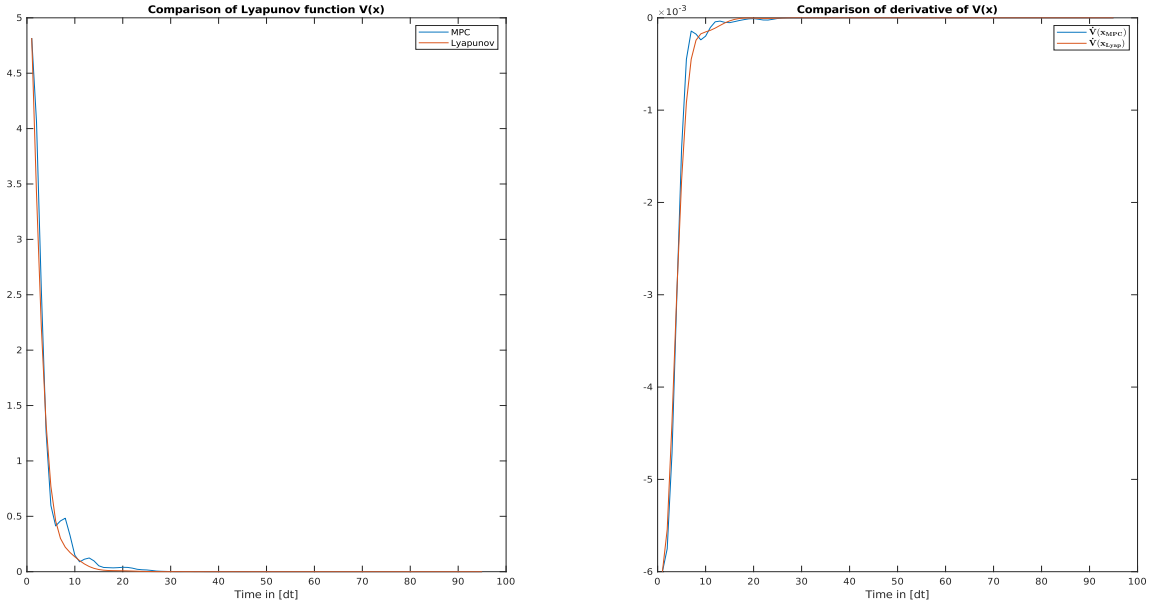

Fig. 4.7: Comparison of Lyapunov function and its derivative of unconstrained MPC and Lyapunov controller during the docking manoeuvre

oscillation are noticeable compared to the unconstrained MPC. This becomes apparent in figure 4.7 as well, where the Lyapunov function $V(x)$ as well as $\dot{V}(x)$ for the MPC and Lyapunov controller are compared. The unconstrained MPC significantly exceeds the Lyapunov controller numerous times. In this regard it can be concluded that the MPC performs worse without Lyapunov constraints, but so far it is stable on its own.

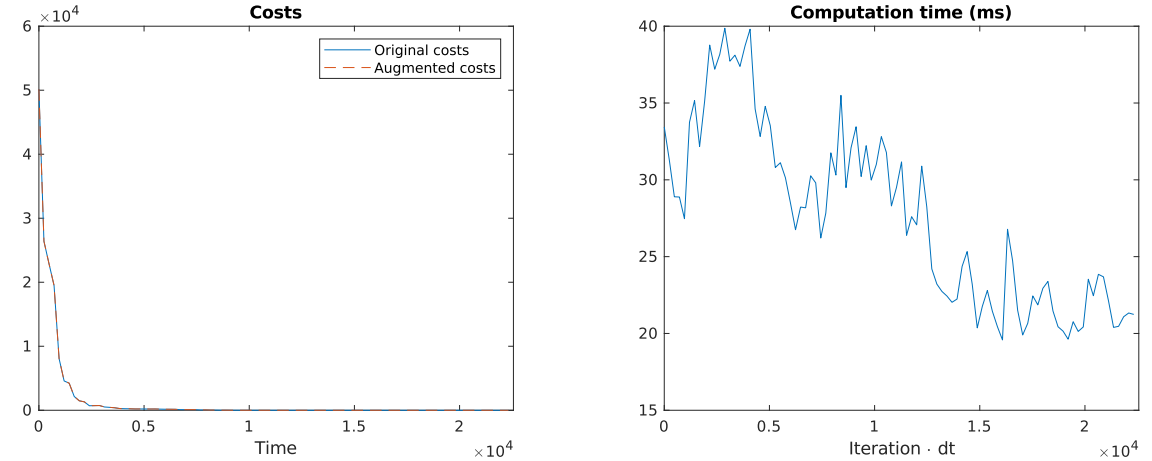

Fig. 4.8: Costs (left) and computational time per iteration (right) of unconstrained MPC during the docking manoeuvre

Figure 4.8 shows the cost function and the computational time of the MPC without

any Lyapunov constraints. The costs are slightly higher, which is due to the bigger oscillation and overshooting as described in the previous paragraph. However, the computation time of the unconstrained MPC averages at around 5 milliseconds less per iteration. Although that number is insignificant on this computer for a sample time of 240 seconds, on less powerful processors such as the ones deployed on CubeSats the difference in computational time might be higher and hence not negligible. This leads to the conclusion that MPC with Lyapunov stability constraints requires slightly more processing power, but on the other hand it provides better performance of the controller.

4.3 Formation Reconfiguration Manoeuvre

Analogously to the previous section, a similar evaluation is conducted for a formation reconfiguration manoeuvre. This means that from an arbitrary starting position relative to the reference satellite, the deputy satellite moves to a different position that is relative to the chief as well. Start position, end position and all other parameters for this simulation scenario are given in table 4.3.

Variable	**Value**
x_0 $[m]$	[-1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
x_{des} $[m/s]$	[0.0, -5.0, 1.0, 0.0, 0.0, 0.0]
PredHor $[dt]$	12
Nhor	25
MaxGradIter	25
MaxMultIter	5
ConstraintTol	[0 ... 0]
Q	diag(10, 10, 10, 0, 0, 0)
R	diag(1, 1, 1)

Tab. 4.3: Parameters and settings for simulation of a reconfiguration manoeuvre

In this scenario, in the beginning the satellite follows the same trajectory as the chief satellite, but one metre "behind" the chief in negative along-track direction. The new desired position has the same x-component as the chief spacecraft, although one metre radially inwards and five metres "upwards" in cross-track direction. Refer to figure 2.1 for a depiction of the coordinate frame.

Besides the state, three more parameters are changed. The prediction horizon $PredHor$ is reduced to 12 time steps which corresponds to approximately half an orbit. The number of discretization steps $Nhor$ and the number of maximum gradient

iterations are both set to 25.

4.3.1 MPC with Lyapunov Stability Constraints

The first simulation is again the MPC with hard Lyapunov constraints for stability.

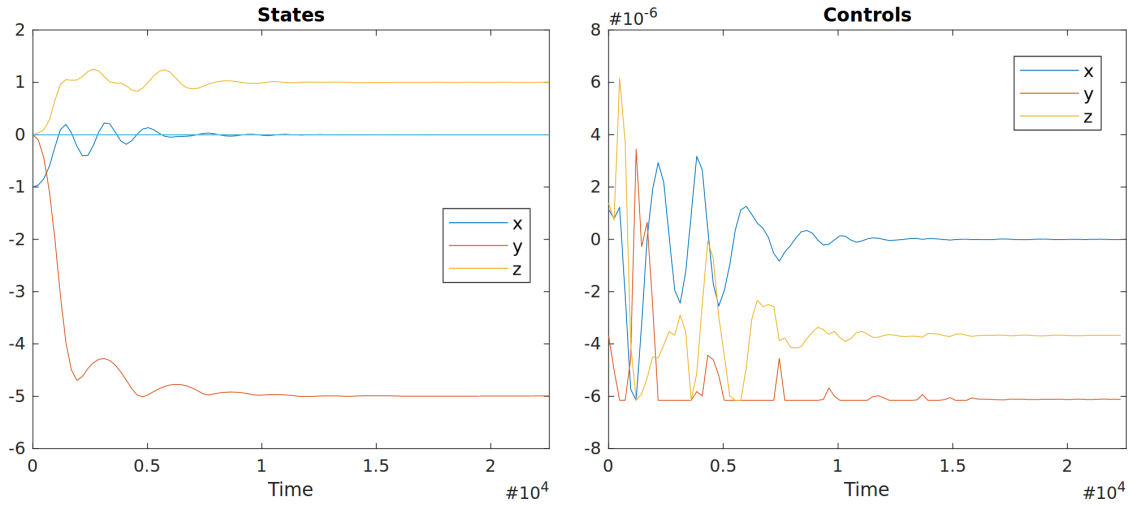

Fig. 4.9: Relative positions and velocities of the satellite (left) in [m, m/s] and the applied control acceleration (right) during the reconfiguration manoeuvre

Figure 4.9 shows the course of the states and control inputs over four orbits. After approximately two and a half to three orbits, convergence is achieved. This is slightly longer than in the last scenario, but it should be considered that the satellite has to travel a farther distance. To be precise, the length of the vector going directly from the initial position to the desired position amounts to 3 metres in the docking scenario and 5.2 metres in this reconfiguration scenario. Hence it can be concluded that this performance is good in terms of convergence time, however the positions oscillate more than they do in the last scenario, which typically is an undesirable behaviour.

This oscillation can be seen in the the costs that are displayed in figure 4.10 as well. The graph has more spikes and the reconfiguration costs a higher amount than the costs in the docking manoeuvre. The computational effort is clearly bigger, as the average computing time per iteration is about 10 to 15 milliseconds higher.

A possible explanation for this is that the controller has to work hard to comply with the constraints which results in more iterations and calculations when solving the optimization problem. This can be derived from figure 4.11, which shows the Lyapunov function and the derivative $\dot{V}(x)$ for the MPC and the Lyapunov controller.

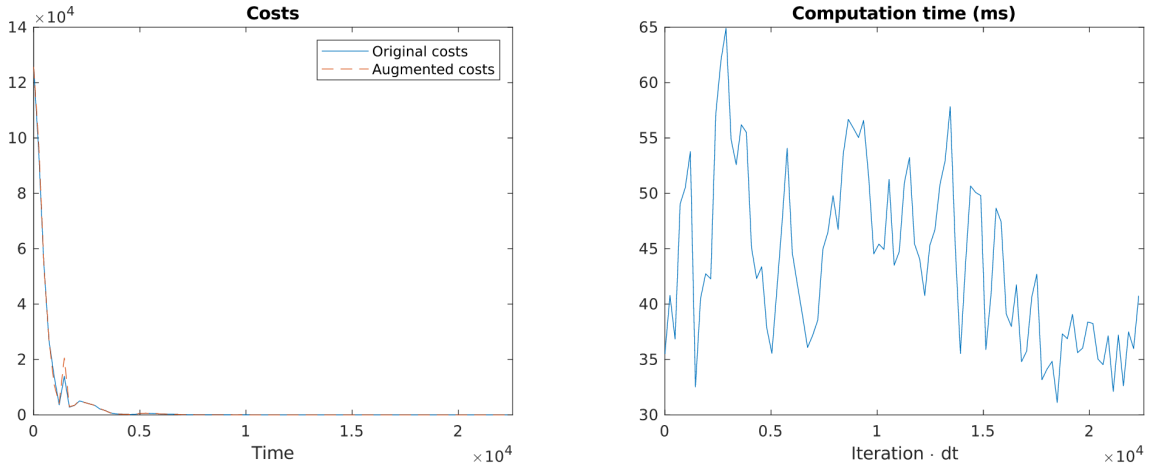

Fig. 4.10: Costs (left) and computational time per iteration (right) during the reconfiguration manoeuvre

In the beginning for the first 6 time steps, the Lyapunov function of the MPC permanently exceeds $V(x)$ of the Lyapunov controller, which means that the Lyapunov constraints $V(x_{MPC}) \leq V(x_{Lyap})$ defined in equation 3.13 are violated. Then between time steps 7 to 15, the MPC achieves to fullfill the constraints. Thereafter however, the oscillating positions result in a few constraint violations again. These difficulties for the solver of the optimization problem are a reason for the higher computational burden.

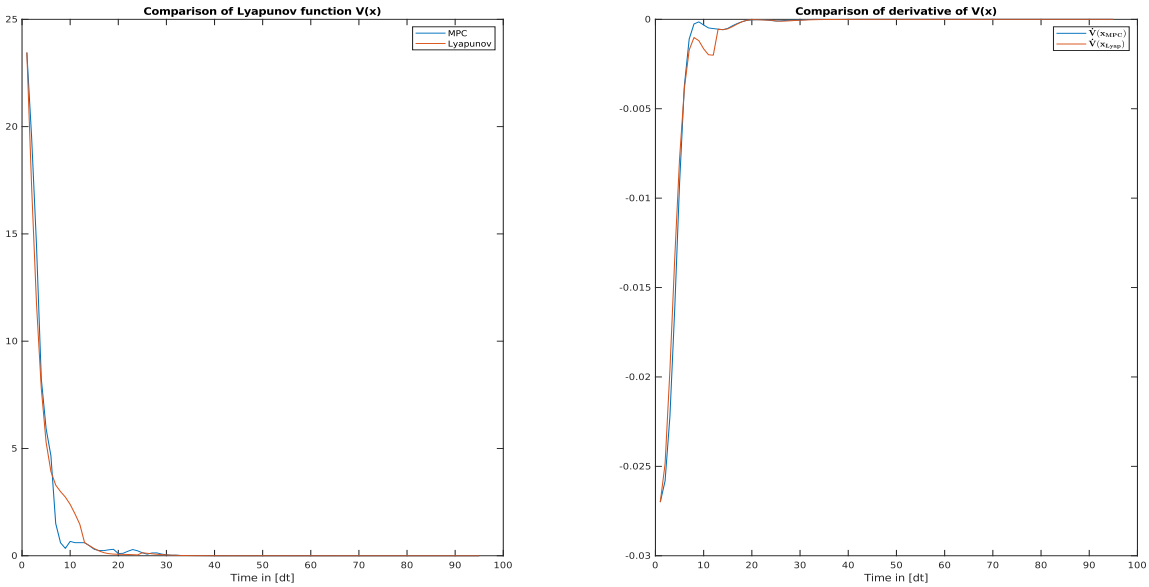

Fig. 4.11: Comparison of Lyapunov function and its derivative of MPC and Lyapunov controller during the reconfiguration manoeuvre

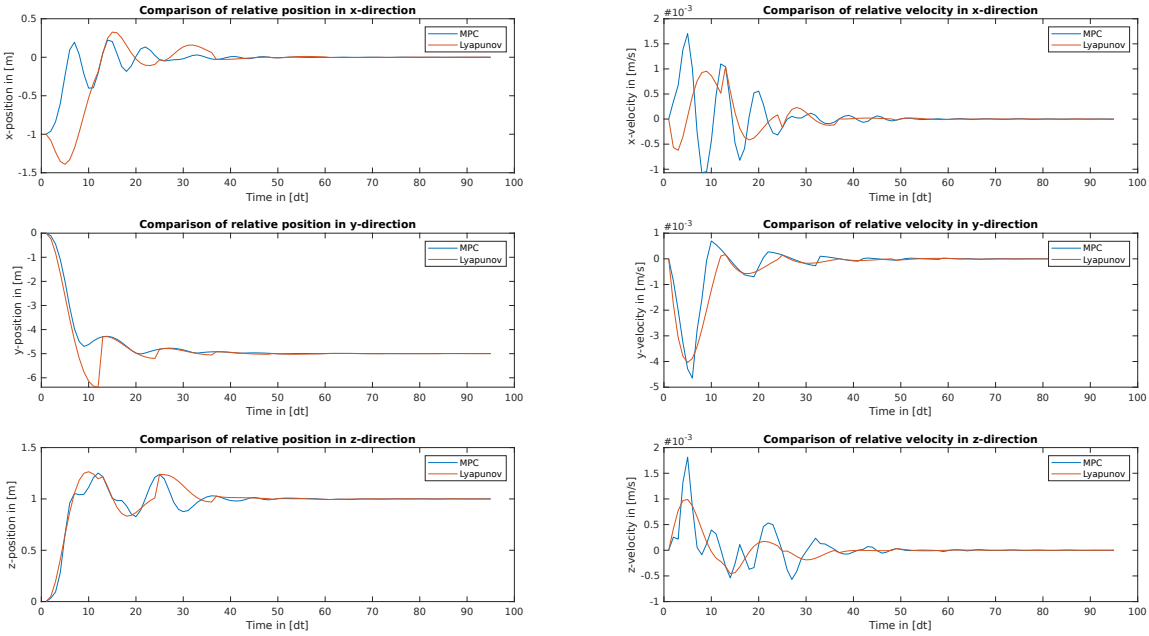

Fig. 4.12: Comparison of relative position and velocity of MPC and Lyapunov controller during the reconfiguration manoeuvre

The last graph is figure 4.12 and shows the difference between the MPC and Lyapunov states in greater detail. The violation of the Lyapunov constraints becomes apparent when looking at the relative velocities. During the first time steps, they clearly exceed the Lyapunov velocities and have an overshoot that is up to twice as big compared to the Lyapunov controller. This explains the constraint violation in the beginning. The bigger MPC velocities in x- and z-direction at time steps 15 to 30, as well as the slightly oscillating positions, are the reason for the violation of the Lyapunov constraints from time step 15 onwards.

But despite the criticism expressed about this reconfiguration manoeuvre and its suboptimal parametrization, the MPC achieves to control the states to the desired values and obtains stability and convergence within the time frame of four orbits. In the next simulation, the Lyapunov constraints are disabled to further inspect their functionality.

4.3.2 MPC without Lyapunov Stability Constraints

The following figures depict the same reconfiguration scenario with identical parametrization, the only difference being that there are no Lyapunov constraints on the controller to achieve stability.

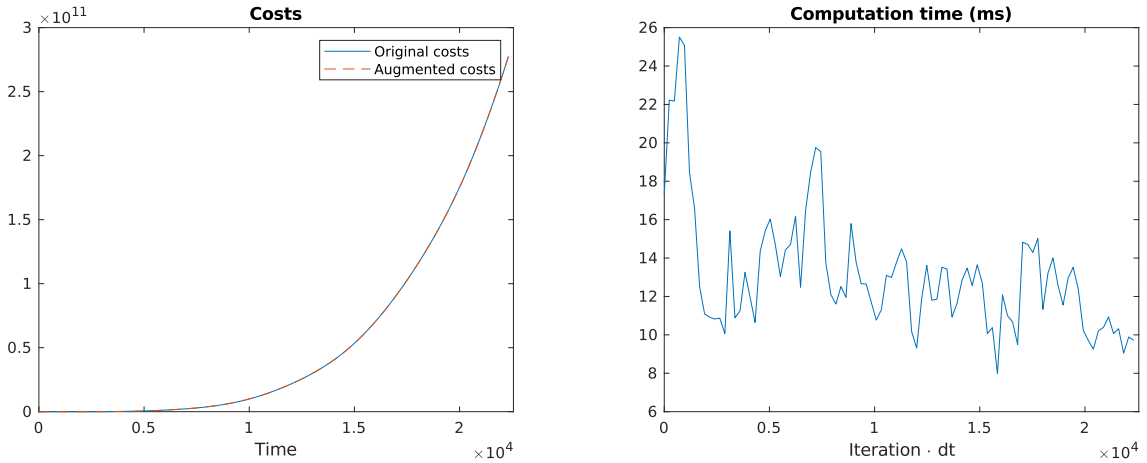

Fig. 4.13: Costs (left) and computational time per iteration (right) of unconstrained MPC during the docking manoeuvre

It instantly becomes apparent that the MPC on its own is not able to stabilize the system. In figure 4.13 one can see that the cost function increases exponentially, which is the opposite of what is supposed to happen, as controlling the state is achieved by minimizing the cost function.

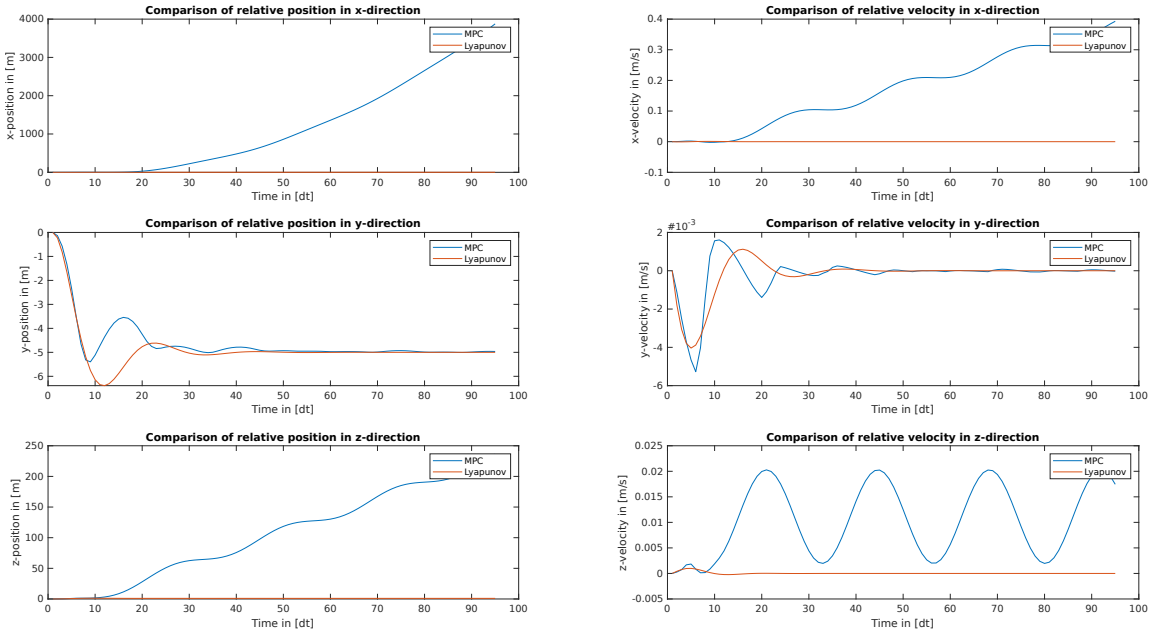

Fig. 4.14: Comparison of relative position and velocity of unconstrained MPC and Lyapunov controller during the reconfiguration manoeuvre

In figure 4.14, where the MPC states are compared to the Lyapunov controller states, the reason for the rising costs can be recognized. The position and velocity in

y-direction, which are decoupled from the x- and z-component, are controlled to the desired values by both the unconstrained MPC and the Lyapunov controller. The desired x- and z-states are achieved by the Lyapunov controller as well, the MPC however fails to regulate these two components. The satellite is permanently accelerated in along-track direction along the x-axis and is simultaneously moved radially inwards along the z-axis in an oscillating manner.

As a conclusion, this scenario shows the importance of the stability constraints very clearly. The Lyapunov constraints force the MPC to regulate the system in a way that prevents the states from diverging, as it is the case in this reconfiguration manoeuvre.

4.4 Discussion

The last section of this chapter is a conclusive evaluation and discussion of the controller and the results it achieved in the simulation. It was shown that the MPC with Lyapunov constraints performs significantly better. And even though these constraints increase the computational effort slightly, they are essential for the controller in order to stabilize the system.

In both simulations, the constrained MPC was able to achieve an accuracy of 10^{-3} metres for the relative position, however it is important to note that this simulation is very idealistic as it did not model any sensor noise or disturbances. Therefore these deviations probably come from numerical errors, the integration scheme that is used or from suboptimal parameters. Apart from that, its performance in terms of constraint handling, convergence speed and computational time is very satisfactory. Although the Lyapunov constraints are violated at times, as seen in figure 4.11, it must be mentioned that the Lyapunov controller does not allow for constraint handling. This means that the feedback control matrix of the Lyapunov controller is not bound to any state or control input limitations and hence might be able to perform control actions, which the MPC is not able to do due to its constraints. Thus a few constraint violations can be excused, as long as the system is still stable.

Lastly some difficulties and drawbacks of the controller that arose during the simulations are pointed out. The first issue that occurred is that the MPC is very sensitive in regards to the parameters. This means that if for instance the prediction horizon

is incremented or reduced by one or two time steps, the performance of the controller changes noticeably. Usually this causes the oscillations to intensify or it takes the system longer to reach its setpoint. The other parameters then have to be tuned as well, to regain a good controller performance. In general "the perfect parametrization" was not found, which is another problem of this controller. While one set of parameters works really good for certain scenarios, it can be a rather bad setting for other manoeuvres. This was the case in the two simulations described in this chapter. The docking manoeuvre is executed almost perfectly by the controller with the parameters defined in table 4.2, yet the reconfiguration scenario oscillated heavily and hardly settled with this same parametrization.

The last drawback of this controller is that for scenarios where the satellite has to cover longer distances such as 50 or 100 metres, the MPC quickly becomes unstable despite the Lyapunov constraints. A reason for this might be the choice of the system dynamics model, as it is linearised under the assumption that the distance between the satellites is infinitesimally small compared to the distance from earth. Larger distances between the spacecrafts might impose errors on the states and therefore make the controller unstable. Another possible reason is that the horizon in which the manoeuvre is performed allows too little time for the system to reach its setpoint, since the actuators are rather slow and the convergence would take longer. Thus a parametrization that suits this scenario better might be able to overcome this problem.

Chapter 5

Conclusion and Further Research

This thesis presents a formation controller that regulates the relative motion of a spacecraft relative to its chief spacecraft. A Model Predictive Control approach is utilized, which is extended by Lyapunov-based constraints to guarantee stability of the system. It is suited for applications in nearly circular low earth orbits and is able to handle docking and formation reconfiguration of the spacecrafts.

Following the introduction and theoretical preliminaries in the first two chapters, chapter 3 presents the model that is used for the relative motion dynamics. Furthermore the problem formulation of the MPC is given, as well as the strategy to ensure the system is asymptotically stable. This is achieved by deriving a Lyapunov controller for the same system and applying this stability principle on the MPC in the shape of inequality constraints.

Chapter 4 describes the simulation of the implemented formation controller. Two different scenarios are tested to show the functionality of the LMPC for both rendezvous and reconfiguration manoeuvres. The results are analysed with respect to overall performance, accuracy, speed and stability of the controller. Besides that, unresolved issues that occurred during the testing are described and potential reasons are named.

Some of these problems pose to be possible fields for future work in order to further improve the proposed formation control scheme.

For instance the simulation performed in this thesis does not model any orbit perturbations, sensor noise or data loss. Testing the controller in a more realistic environment would allow for a stricter and more thorough evaluation and give even more insights into possible deficits of the controller.

Another possibility is extending the dynamics model to a nonlinear model. This

would allow for bigger distances between the spacecrafts and therefore more flexible formation configurations. Furthermore disturbances such as gravitational effects or atmospheric drag can be included in the nonlinear model to enable better disturbance rejection. Besides that, it might be useful to include a strategy for collision avoidance, as this would be necessary when applying the controller in space.

Lastly remains transferring the controller into solely "C"-language code. If the LMPC is to be applied on a satellite formation in space, the implementation of the controller has to be embedded on the spacecraft. Hence "C" code is desirable. GRAMPC provides all necessary libraries as well as a template, in which the contents of the MATLAB files "initData.m" and "startMPC.m" have to be merged into. This facilitates rewriting the LMPC into a shape that is suited for applications in embedded systems. The problem formulation and constraints can be used without any changes, as the file "probfct.c" is already written in "C" language.

List of Figures

List of Tables