Non-linear Model Predictive Control for space debris removal missions

Alexander Korsfeldt Larsén

Space Engineering, master's level
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Foreword and acknowledgments

The great challenges we may soon face due to the space debris problem were not something I had fully grasped when I started this work, but now I am very grateful for the opportunity I was given to work towards a solution to it. The first steps of this project were taken almost a year ago, during a meeting in Kiruna with my professors Leonard Felicetti and Damiano Varagnolo. A few months later I left the Swedish winter behind to go live in Padova, Italy, and learn from some of the most experienced control systems engineers in Europe. This includes Ruggero Carli, Angelo Cenedese, and Mattia Bruschetta. I was also lucky enough to get some much needed advice from the developer of MATMPC himself, Yutao Chen.

I want to thank all of my mentors for giving me a lot of freedom in trying out my own ideas and learning from my own mistakes, while at the same time always being there to support me with guidance and feedback. I also want to thank everyone else at Luleå Tekniska Universitet, Università Degli Studi Di Padvoa, as well as the Erasmus organization, who helped make this project a possibility.
Abstract

The rapidly increasing amounts of space debris orbiting Earth is threatening to reach a critical level, where the near-Earth environment becomes so overfilled with junk that many missions simply become unfeasible. Long-term active debris removal operations appear to be a necessity, but due to the scale of the problem this will likely be an expensive affair spanning decades or even centuries.

Many of the mission-related costs can be significantly reduced by making use of a smaller spacecraft, such as the rapidly developing CubeSat standard. An issue with this approach is the limited actuation capabilities, as that makes it very difficult to perform orbital maneuvers in a fuel-efficient manner. Rather than making a few high-impulse thrusts over the course of the mission, the thrust must be applied continuously for several hundred hours.

This thesis attempts to solve the problem by using a non-linear Model Predictive Control strategy to implement an Orbital and Attitude Control system for a small satellite. This was done in MATLAB, using the fast-NMPC package MATMPC recently developed by Yutao Chen at Padova University. The controller was tested in a realistic model of the near-Earth environment, where disturbances such as drag and gravitational perturbations are simulated.

It was shown through simulations that this method can successfully be used to perform a fuel-efficient rendezvous maneuver with an uncontrollable object, a critical step in any Active Debris Removal operation. Using a 4 kg CubeSat with a 30 µN thruster mounted on each of its six surfaces, the total mass consumption for a phasing maneuver of 10 degrees at 300 km altitude was less than 0.1% of the spacecraft mass. This assumes an $I_{sp}$ of 1,150 s, which is the specific impulse of the S-iEPS Electrostatic thruster flown on the Aero-Cube-8 IMPACT mission.

This is made possible through prediction horizons spanning several days, which in turn forces the controller to operate at sampling rates as low as 1/100 Hz due to the computational load. Fully realizing the potential of this technique would likely require the inclusion of a low-level controller that uses the generated trajectories as input values, as this would negate many of the issues associated with a heavy computational load. The urgency of the space debris problem, and the endless list of other CubeSat applications that would benefit from a flexible and fuel-efficient AOCS, makes this an interesting topic to consider for further research.
## Contents

1 Introduction 13  
1.1 Space debris .................................................. 13  
1.2 Orbital Rendezvous .......................................... 15  
1.3 Model Predictive Control and MATMPC ...................... 15  
1.4 Coordinate systems .......................................... 18  
1.5 Attitude representation ..................................... 19  
1.6 Literature review ............................................ 20  
1.7 The goal of this thesis ...................................... 22  

2 System dynamics and modeling 23  
2.1 Dynamics of the environment ................................. 23  
2.1.1 Earth's gravitational field ............................... 23  
2.1.2 Atmospheric drag ......................................... 23  
2.1.3 Lunar gravity .............................................. 24  
2.1.4 Solar gravity and radiation pressure ..................... 25  
2.1.5 Gravity gradient torque ................................ 26  
2.1.6 Summary of the model ................................... 26  
2.2 Dynamics of the chaser ....................................... 27  
2.2.1 Properties of a CubeSat ................................ 28  
2.2.2 CubeSat propulsion systems ............................... 28  

3 Designing the controller 31  
3.1 Defining the dynamics and choosing constraints .......... 31  
3.2 Sampling rate and prediction horizon requirements ....... 34  
3.3 Generating reference quaternions on-line ................. 34  
3.4 Intercepting the target using a 2 phase maneuver ....... 41  
3.5 Long range ..................................................... 41  
3.5.1 Tuning the weights of the cost function ............... 51  
3.5.2 Length of the prediction horizon ......................... 58  
3.5.3 Sampling rate and number of shooting points ......... 62  
3.6 Short range ..................................................... 68  
3.6.1 Late transition ............................................ 68  
3.6.2 Early transition ........................................... 76  
3.6.3 Comparison of the two transition points .......... 86  
3.7 Solver comparison ............................................ 87  
3.8 Relationship between the cost function, length of the prediction horizon, and thruster limitations ................. 87  
3.8.1 Long range ................................................ 87  
3.8.2 Short range ............................................... 93  

4 Closed loop simulations 100  
4.1 Types of noise ............................................... 100  
4.2 Structure of the closed loop system ......................... 101  
4.3 The effect of modeling errors ............................... 101  
4.3.1 100 km altitude .......................................... 101  
4.3.2 300 km altitude .......................................... 104  
4.3.3 600 km altitude .......................................... 108  
4.3.4 Effects of not considering the J2-effect ............... 112  
4.4 The effect of sensor noise .................................. 113  
4.4.1 Orbital related parameters ............................... 113
4.4.2 Attitude related parameters ........................................... 116
4.4.3 Increased prediction horizon in the presence of noise .......... 127
4.5 Summary of long range results ........................................... 131
4.6 Noise in the short range scenario ....................................... 132
   4.6.1 Modeling noise .................................................... 132
   4.6.2 Sensor noise ....................................................... 134
   4.6.3 Latest iteration of the short range configuration ............... 139

5 Conclusion ........................................................................... 143
   5.1 Discussion of the results ................................................ 143
   5.2 Comparison with an ideal Hohmann maneuver .................... 145
   5.3 Future improvements .................................................... 147
      5.3.1 Accurate modeling of the actuators ......................... 147
      5.3.2 Adding a low-level controller ............................... 148
      5.3.3 Docking with the target ...................................... 148
   5.4 Issues encountered ..................................................... 149

A Appendix A: Matlab Codes .................................................. 152
   A.1 Main file (Closed-loop Simulation) ................................. 152
   A.2 Main file (Open-loop Simulation) .................................. 155
   A.3 Simulated system (chaser) .......................................... 159
   A.4 Simulated system (target) .......................................... 163
   A.5 Controller dynamics ................................................ 166
   A.6 Position reference generator ...................................... 169
   A.7 Attitude reference generator ..................................... 170
   A.8 Init data Long Range Closed-loop ............................... 171
   A.9 Chaser noise ......................................................... 174
   A.10 Target noise ......................................................... 174
   A.11 Calculate Julian Date .............................................. 175

B Appendix B: Simulink Plot ................................................... 175
List of Figures

1.4.1 The Earth Center Inertial (ECI) reference frame. The Z-axis passes through the north pole, while the x- and y-axis pass through the equator line. Origin is always located at the center of the Earth, but the axes are fixed with respect to some distant stars and do not follow the Earth’s rotation. 18

1.4.2 A Body-Fixed frame centered on the chaser satellite. As the satellite rotates, the axes will follow along. The camera points along the y-axis, which can be considered the “forward” direction of the chaser. 19

1.4.3 A LVLH frame centered on the target. The axes are independent of the target’s attitude, but are instead a function of its position in the orbit. 19

2.1.1 Example values of atmospheric density as a function of altitude. Data is taken from NRLMSISE-00, for a latitude of 55 degrees, a longitude of 45 degrees, and with the date set to 2017-01-01, 12:00 UT. 24

2.2.1 The chaser viewed from the front. 27

2.2.2 The chaser viewed from the back. 27

2.2.3 Characteristics of a commercially available gold gas propulsion system. [19] 29

3.3.1 Example of a quaternion reference signal obtained when using the ‘dcm2quat’ function and not performing any corrections. 37

3.3.2 Example of a quaternion reference signal obtained when using the ‘dcm2quat_mod’ function and not performing any corrections. 38

3.3.3 Example of a quaternion reference signal that has been corrected by flipping the sign of the quaternion at the appropriate points. 39

3.3.4 A simple 3d-visualization of the system states during simulation. The block represents the chaser satellite, while the glowing ball represents the target space debris. The red line indicates how the y-axis of the chaser is currently oriented, while the purple line is the reference value. 40

3.5.1 Initial conditions of the long range scenario, shown graphically. 42

3.5.2 Illustration of how the chaser moves compared to the target. It is clear that it can catch up the target by reducing its orbital radius, and it ends the simulation roughly 0.5 meters from the target. 44

3.5.3 The position of the chaser in the target-centered LVLH-frame. 45

3.5.4 The trajectory of the chaser in the target-centered LVLH-frame. The red dot at [0,0,0] represents the target. 46

3.5.5 The trajectory of the chaser in the target-centered LVLH-frame, zoomed in to show the last parts of the maneuver. The red dot at [0,0,0] represents the target. 47

3.5.6 Control thrust applied in the first simulation. 48
3.5.7 The rotation of the chaser expressed in quaternions.

3.5.8 Pointing error of the first simulation.

3.5.9 The torque being applied by the reaction wheels. The saturation point of 0.5 \( \mu \text{Nm} \) is reached several times during the first few hours.

3.5.10 Trajectories in the LVLH-frame for the 5 tested cases of different cost functions.

3.5.11 The relative LVLH position vector, split into its components, for the 5 tested cases of different cost functions.

3.5.12 Orbital trajectories for the 5 tested cases of different cost functions.

3.5.13 Control thrust used in the 5 tested cases of different cost function weights.

3.5.14 Control torque and angular velocity in case 4.

3.5.15 Error quaternion in case 4.

3.5.16 Relative motion between chaser and target in five of the tested cases with different numbers of shooting points.

3.5.17 Components of the relative motion between chaser and target in five of the tested cases with different numbers of shooting points.

3.5.18 Distance to target and relative semi-major axis for the 9 tested cases with different numbers of shooting points.

3.5.19 Distance to target and relative semi-major axis over time, for the four tested cases of different sampling rate and number of shooting points.

3.5.20 Thruster usage for the four tested cases of different sampling rate and number of shooting points.

3.5.21 Trajectory in the LVLH-frame for the three cases with equal prediction horizon length.

3.5.22 Components of the chaser’s relative position vector in the LVLH-frame, for the three cases with equal prediction horizon length but different sampling rate and number of shooting points.

3.6.1 Distance, relative velocity, and difference in semi-major axis between the chaser and target in the short range simulation.

3.6.2 Relative motion of the short range case shown in the LVLH-frame.

3.6.3 Trajectory in the LVLH-frame for the short range phase.

3.6.4 Thruster usage in the short range scenario.

3.6.5 Error quaternion for the short range simulation

3.6.6 Angular velocity and control torque in the short range scenario.

3.6.7 Relative distance, velocity, and semi-major axis in the transition region.
3.6.8 Thruster usage for the long and short range phases combined into one plot.

3.6.9 Distance between chaser and target during the long range phase. Subplots 2 and 3 are zoomed in at regions where they are close to each other.

3.6.10 Trajectory in the LVLH-frame in the close range phase initialized at t=50h.

3.6.11 Components of the LVLH-position vector in the close range phase initialized at t=50h.

3.6.12 Trajectory in the LVLH-frame in the close range phase initialized at t=50h. This has been zoomed in to show the spiral pattern with which the chaser approaches the target once its close enough.

3.6.13 Relative distance and velocity in the close range phase initialized at t=50h.

3.6.14 Thruster usage in the close range phase initialized at t=50h.

3.6.15 Error quaternion in the close range phase initialized at t=50h.

3.6.16 Control torque and angular velocity in the close range phase initialized at t=50h.

3.6.17 Relative distance, velocity, and semi-major axis in the transition region when doing an earlier transition to short range mode.

3.6.18 Thruster usage for the long and short range phases (when doing an early transition) combined into one plot.

3.8.1 Comparison between orbital trajectories of two controller configurations. The first configuration has 1200 shooting points, while the second configuration has 1500. The cost function weights are shown in table 3.8.1.

3.8.2 Comparison between thruster usage of two controller configurations. While the second configuration doesn’t quite reach a zero-output steady state, it uses significantly less fuel. It is clear that the y-thruster, oriented along the velocity vector, is responsible for most of the actuation in both cases. This is seen as a good behavior, as increasing or decreasing velocity is the main technique for switching orbits.

3.8.3 Orbital trajectories based on thruster strength. It is clear that with weaker thrusters, the chaser needs a longer time to catch up to the target.

3.8.4 Thruster usage using constraints of different magnitudes. The y-thruster is still the most used one, being the only thruster that reaches saturation in all three cases. We can also see that the 200 uN configuration is the least reliant on the x-thruster; these results indicate that when the y-thruster gets saturated the controller will delegate some additional work to the x-thruster instead.

3.8.5 Components of the LVLH position vector during the short range phase, using 30µN thrusters.

3.8.6 Trajectory in the LVLH-frame during the short range phase, using 30µN thrusters.
3.8.7 Orbital trajectory during the close range phase. The initial conditions are the final states from the previous 30 uN simulation. The Chaser is successfully able to equalize its semi-major axis with that of the Target, and ends the simulation fluctuating between being 75 meters and just a few centimeters away from the target.

3.8.8 Thruster usage during the close range phase. All thrusters are used extensively during the orbit circularization, but the controller is very close to a zero-output steady state at the end of the simulation.

3.8.9 Close and short range scenarios combined into one graph. It is not until the close range phase that the controller is able to circularize the orbit correctly, as seen in the third subplot.

3.8.10 Thruster usage in the combined scenario. The heavy usage during the close range phase, especially regarding the z-thruster, may be seen as a sign that the cost function is not very well tuned. It is possible that increased efficiency could be achieved by lowering the cost of the position states, but this would increase the steady state error. This could be solved by introducing a third phase (with higher costs) or by having a cost function that scales dynamically based on the current states. The behavior shown here also reflects that more time was devoted to tuning the long range mode of the controller, as that is where most of the fuel is spent.

4.2.1 Block diagram of the simulated system.

4.3.1 Comparison of the orbital trajectories of a noise-less system compared to one where perturbations are applied to the chaser at 100 km altitude. The first 2 subplots show that even with heavy drag present, the controller gets very close to matching the position and speed of the target. It is not able to correct its semi-major axis after however, and ends up drifting away. This could mean that the drag makes the chaser unable to "circularize" its orbit to match that of the target. When perturbations affect the target as well, the controller is not able to find a solution.

4.3.2 Comparison of actuation usage in a noise-filled environment versus a noise-less one, at 100 km altitude. It can be clearly seen that the y-thruster saturates, and is unable to provide the force required to enter a higher orbit. In the noiseless scenario there is, as expected, not much difference from the higher orbits.

4.3.3 Comparison of the LVLH trajectories of a noise-less system compared to a more realistic one at 300 km altitude. The additional perturbations make virtually no difference.

4.3.4 Comparison of the LVLH position vector components of a noise-less system compared to a more realistic one at 300 km altitude. The additional perturbations make virtually no difference.

4.3.5 Comparison of the orbital trajectories of a noise-less system compared to a more realistic one at 300 km altitude.

4.3.6 Comparison of actuation usage in a noise-filled environment versus a noise-less one, at 300 km altitude. No difference in behavior can be seen.

4.3.7 Trajectory of the chaser in the LVLH-frame for the 600 km scenario.

4.3.8 Position vector of the chaser in the LVLH-frame.
4.3.9 Comparison of the orbital trajectories of a noise-less system compared to a more realistic one at 600 km altitude. The first 2 subplots show the relative position and velocity between Chaser and Target for the two models, and the behavior is almost identical. In the third subplot we can see the general shape of the maneuver; just like in earlier tests the chaser lowers its altitude to catch up to the target, but ends up overshooting and having to enter a higher orbit temporarily. After approximately 107 hours the chaser is extremely close to the target, but since its semi-major axis is almost 2 km shorter than that of the target it is not able to catch it at that point. At the end of the simulation, the Chaser's semi-major axis is only around 50 meters shorter than that of the Target.

4.3.10 Comparison of actuation usage in a noise-filled environment versus a noise-less one, at 600 km altitude. No difference in behaviour can be seen.

4.4.1 Example of the white noise applied to the position and velocity states.

4.4.2 The effect of applying sensor noise to the position and velocity states. The trajectories are almost identical at this noise level, but the controller loses stability if the noise levels are increased much further than this. Case 1: No sensor noise. Case 2: Noise on target states. Case 3: Noise on chaser and target states.

4.4.3 Actuation usage when sensor noise is applied to the position and velocity states. Case 1: No sensor noise. Case 2: Noise on target states. Case 3: Noise on chaser and target states.

4.4.4 Example of the quaternion measurements that are passed to the controller when the [1, 0, 0, 0] vector is fed through the noise generator.

4.4.5 Angle error when white noise is applied to the measured quaternion states. This is very different from the noise-less case, where angle errors were only present at the very start of the simulation.

4.4.6 Control torque and angular velocity when white noise is applied to the measured quaternion states. All three actuators are almost constantly saturated, and the chaser wiggles back and forth.


4.4.8 Thrust applied in the three tested scenarios. Though the orbital trajectories are almost identical, there are some visible differences regarding thruster usage. The third case stand out the most when seen in this way, and it also consumes the most fuel.

4.4.9 Angle error when white noise is applied to the measured quaternion states, using the modified cost function and weaker actuators. Compared to 4.4.5 the mean error is smaller.

4.4.10 Orbital trajectories for the three cases discussed. The motion is almost identical, but some small differences can be observed close to the end of the simulation when zooming in.
4.4.11 Example of the randomized noise pattern applied to the controller’s measurements of the chaser’s angular velocity.

4.4.12 Torque applied and angular velocity when a disturbance is applied to the angular velocity measurements.

4.4.13 Error quaternion when noise is applied to the measurement of the angular velocity states.

4.4.14 Distance and relative velocity to the target. Case 1: N=2,000, no sensor noise. Case 2: N=3,000, no sensor noise. Case 3: N=3,000, applied sensor noise.


4.4.16 Motion in the LVLH-frame. Case 1: N=2,000, no sensor noise. Case 2: N=3,000, no sensor noise. Case 3: N=3,000, applied sensor noise.

4.6.1 Trajectory in the LVLH-frame. Case 1: N=2,000 Case 2: N=3,000

4.6.2 Actuation usage in the closed-loop short range scenario. Case 1: N=2,000 Case 2: N=3,000

4.6.3 Relative position and velocity in the close range phase, using cost function 4.6.4. Case 1: No sensor noise. Case 2: With sensor noise.

4.6.4 Relative position and velocity in the close range phase, using cost function 4.6.4. Case 1: No sensor noise. Case 2: With sensor noise.

4.6.5 Relative position and velocity in the close range phase, using cost function 4.6.4. Case 1: No sensor noise. Case 2: With sensor noise.

4.6.6 Thruster usage in the close range phase, using cost function 4.6.4. Case 1: No sensor noise. Case 2: With sensor noise. The behavior is very similar at first, but around the 40h mark the trajectories of the x-thruster diverge.

4.6.7 Relative position in the LVLH-frame. Short range scenario with sensor noise and cost function 4.6.5

4.6.8 Distance and relative velocity between the chaser and the target. Short range scenario with sensor noise and cost function 4.6.5

4.6.9 Actuation usage. Short range scenario with sensor noise and cost function 4.6.5

5.2.1 The actuation required for an Ideal Hohmann transfer, for a phase difference of 10 degrees in a 600 km altitude circular orbit.

B.0.1 Simulink structure for showing all the three reference frames.
List of Tables

3.1.1 Constraints placed on the state and control parameters for the initial cold gas configuration. 34

3.5.1 Initial orbital parameters for the long range scenario. 42

3.5.2 Initial orbital states of the chaser and target for the long range scenario. 42

3.5.3 Initial attitude states of the chaser for the long range simulation. 42

3.5.4 Cost function weights for the first test. 43

3.5.5 5 cases with different cost function weights and corresponding thruster usage. 52

3.5.6 Nine cases with different number of shooting points (N). The total amount of control thrust used in each case, as well as the maximum and average computation time for each step, is also shown. 59

3.5.7 Three cases with a prediction horizon length of 22.22 hours, but different sampling rates and number of shooting points (N). The total amount of control thrust used in each case, as well as the maximum and average computation time for each step, is also shown. Case 2 has only half the prediction horizon length as the other cases; it uses the same computation time as case 1 but performs notably worse in fuel efficiency and settling time. 63

3.6.1 Cost function weights used in the short range simulation. 68

3.6.2 Initial orbital states of the chaser and target for the close range simulation. 68

3.6.3 Initial orbital states of the chaser and target for the close range simulation starting at t=50 hours. 77

3.6.4 Comparison between the two transition points in terms of mission time and thruster usage. 86

3.7.1 Comparison between different solvers regarding thruster usage and computation times. The trajectories were very similar in all cases. 87

3.8.1 Cost function weights, number of shooting points, and thruster usage for the two simulations presented. 88

3.8.2 Cost function weights when using weaker actuators. 91

3.8.3 Summary of the results obtained in this chapter. More tuning and additional insights have allowed us to utilize a longer prediction horizon, weaker thrusters, and a much longer prediction horizon. This significantly reduces fuel consumption. 93

3.8.4 Cost function weights when using weaker actuators in the short range phase. 93

4.3.1 Initial orbital parameters for the 100 km scenario. 102

4.3.2 Initial orbital states of the chaser and target in the 100 km scenario. 102

4.3.3 Initial orbital states of the chaser and target in the 300 km scenario. 105
4.3.4 Initial orbital states of the chaser and target in the 600 km scenario.

4.4.1 Updated cost function weights used in case 3 for the quaternion noise test.

4.4.2 Updated cost function weights used when applying noise on the measurement of the angular velocity states.

4.4.3 Numerical results for the closed-loop prediction horizon test.

4.5.1 The effect of modeling noise at different altitudes. *Did not reach target. **50% longer prediction horizon.

4.5.2 Summary of how white sensor noise applied to the various states affects fuel expenditure. *Uses cost function 4.4.1. **Uses cost function 4.4.2. ***50% longer prediction horizon, and quaternion noise amplitude 15x lower than figure 4.4.4.

4.6.1 Initial orbital states of the chaser and target in the short range scenario.

4.6.2 Initial attitude states of the chaser for short range simulation.

4.6.3 Computation times and thruster usage for two cases of prediction horizon length in the short range scenario.

4.6.4 Cost function weights in the closed-loop close range scenario featuring sensor noise.

4.6.5 Cost function weights used in the latest iteration of the short range configuration (November 25).
Acronyms

ECI  Earth Centered Inertial
MPC  Model Predictive Control
NMPC  Non-linear Model Predictive Control
ESA  European Space Agency
LVLH  Local-Vertical Local-Horizon
LQR  Linear Quadratic Regulator
RTI  Real Time Iteration
AOCS  Attitude and Orbital Control System
LEO  Low Earth Orbit
GEO  Geosynchronous Orbit
SQP  Sequential Quadratic Programming
QP  Quadratic Programming problem
NLP  Non-Linear Problem
DCM  Direction Cosine Matrix

1 Introduction

1.1 Space debris

The issues caused by space debris are receiving more and more attention, and for good reason. There has always been small objects like meteoroids and other rocks in orbit around the Earth, and while they can cause serious damage to expensive satellites or even manned spacecrafts the risks can be reduced to acceptable levels using appropriate protection methods. The vast increase in human spaceflight has not come without consequences however, as the near-Earth space is quickly becoming over-populated with man-made objects such as spent rocket stages, "dead" satellites, and small fragments from various spacecrafts. The population is threatening to reach a critical level, where any collision could lead to a chain reaction that fills the entire LEO region with so many small particles and fragments that any further operations become unfeasible. This scenario was first described by Donald J. Kessler in 1978, and has since then become known as the Kessler Syndrome.

The main factor that determines how long any given object remains in orbit is its altitude. Space junk at an altitude of 100 km will be heavily affected by the drag from Earth’s atmosphere and will only remain in orbit for a few days, while debris at 800 km altitude or above may remain there for centuries. This means that the upper LEO and GEO regions are especially sensitive, as they are heavily utilized by man and will not get cleaned by natural forces in a foreseeable future.

To keep the growth rate of additional space debris to a minimum, international guidelines have been established for mitigation strategies. These guidelines are voluntary for each nation to accept, but several space faring countries have used them as input when creating their own mitigation standards.

Simulations have shown, however, that relying purely on mitigation is not enough to solve the problem. In [13] it is noted that even if all future space launches are permanently suspended, the amount of debris in LEO will continue to grow over the next 200 years. This is due to the chance of
space debris colliding with each other; this can create enormous amounts of new fragments which in turn increases the risks of future collisions even more.

The same study also looked at a more realistic scenario, where future launches are simulated as repetitions of the 1999-2006 launch cycle. Here it was found that even if post-mission disposal is performed with a 90% success rate from the year 2020, stabilizing the amount of space debris larger than 10 cm within the next 200 years will require active removal of the five most problematic pieces of debris every year. The results also indicated that a good way of quantifying the threat posed by each piece of debris is to take the product of its mass and its probability of collision. This means that the individually most threatening objects are larger debris pieces moving through highly populated orbits. Since every large debris is a potential source of many smaller ones, an effective reduction factor (ERF) for each debris removed can be calculated. For example, it was estimated that removing a total of 374 objects until year 2206 would reduce the total population at that point in time by 3899, giving an ERF of 10.4.

Based on their size, the debris orbiting Earth can be classified as follows:

- **Objects larger than 10 cm.** There are more than 20,000 known objects of this category currently in orbit around Earth. This junk is relatively easy to track, and operational spacecraft can therefore plan avoidance maneuvers well in advance. Despite the comparatively low number of objects in this category, their large size means that they make up 95% of the total debris mass. [13]

- **Objects between 1 cm and 10 cm.** The objects in this category are currently the most dangerous, since they are difficult to track but still large enough to cause catastrophic damage. It is estimated that 300,000 of them are currently in orbit.

- **Objects smaller than 1 cm.** Objects this small are nearly impossible to track, but it is believed that more than 100 million exist in this category. As long as sufficient shielding techniques are used, these tiny pieces don’t pose much of a threat to spacecraft.

A significant problem associated with any type of active debris removal are the large financial costs. Traditional spacecraft are expensive to construct and launch into orbit, and once there they must also be able to catch the debris which can cost a significant amount of fuel depending on where the spacecraft was ejected into orbit.

There is also uncertainty about who should finance the debris removal operations. Economists have described the problem as a classic example of the "tragedy of the commons": space is a shared resource that all actors would benefit from keeping clean, but there is no additional personal gain for the one actually financing the operation. One could argue that the actor with the most financial assets invested into a given orbit stands to benefit the most from keeping said orbit clean, but as it stands today the costs still outweigh the benefits. Even in a sun-synchronous orbit, where the risks for collision is the highest, a 10 m² satellite still has a less than 1 % yearly chance of being hit by a piece of debris larger than 1 cm. A common trend is therefore to insure the more expensive assets, and to just hope for the best when it comes to cheaper satellites. [16]

One way to increase the economical feasibility of active debris removal missions is to minimize the mission related costs, and they are in many ways dependent on the mass and size of the spacecraft(s) utilized for the operations. A larger vehicle is more expensive to launch into space, and will, due to its greater mass, need a larger application of force to perform the orbital maneuvers necessary. The recent development of smaller spacecraft, such as the CubeSat standard, means that it may soon be economically viable to perform many space missions previously considered too expensive.
1.2 Orbital Rendezvous

It is clear that removing a relatively small amount of non-controllable objects from the most critical orbits can lead to substantial benefits, but an optimal way of accomplishing that has not yet been found. Changing the orbit of any object in the Earth’s vicinity requires application of a controlled force, either pushing the debris into a graveyard orbit or closer to Earth so that it burns up in the atmosphere. There are several ideas on how to do this, such as using nets, harpoons, or even lasers, but for now it seems that the most feasible solutions demand that a controlled spacecraft is able to get very close to the target space debris. It must then synchronize its rotational movement with that of the target, so that it can perform some kind of docking maneuver and transfer the required force in a predictable way. [14]

The orbital maneuvers necessary to bring the controlled satellite, the chaser, close enough to the target space debris will of course depend on the initial conditions of the scenario; the more similar the orbital parameters of the two objects are, the easier it is to negate the differences. Thus, the more work done by the launch vehicle, the less needs to be done by the satellite itself. One should keep in mind, however, that smaller spacecraft such as CubeSats are often launched as secondary payloads. A design that allows for some flexibility with regards to launch windows will demand more from the chaser itself, but could significantly reduce the cost of launching it into orbit. Certain orbital maneuvers, such as a change of inclination, are however extremely expensive to perform fuel-wise. A likely scenario is therefore that the launch vehicle can get the chaser into roughly the same orbit, but that the chaser needs to perform some phasing maneuvers to match the target’s location in the orbit.

Using conventional thrusters, a phasing maneuver can be executed very efficiently by performing a two-impulse Hohmann transfer. This is done by firing the main thruster for a short period of time when the satellite is located in perigee or apogee, putting it into a faster or slower elliptic orbit depending on which direction the thruster was facing. After a certain number of orbits the chaser should have caught up to the target, and it can then return to its original orbit by performing another thrust of equal magnitude in the opposite direction. The number of revolutions required for this depends on the amount of actuation used, as that will determine how much slower or faster the transfer orbit is compared to the original one.

The CubeSat propulsion systems available today, briefly examined in chapter 2.2.2, are unfortunately far too weak to rely on this method. The only way to reach the target in a reasonable amount of time is to make use of a continuous thrust action that over time supplies enough energy to obtain the desired orbital parameters. The non-linear dynamics involved make this a difficult problem to solve efficiently, as the force must be applied over very long periods of time. While earlier research has indeed proven analytically that orbital maneuvers can be accomplished using very limited actuators [9], it is only in the most recent years, thanks to rapid advancements in non-linear control strategies, that we have been given the opportunity to fully explore the possibilities of such a system. This study is focused on Model Predictive Control, a strategy that has yielded promising results in previous studies like [17] regarding CubeSat formation control and fuel efficiency.

1.3 Model Predictive Control and MATMPC

A Model Predictive Controller is, in its essence, very similar to a Linear Quadratic Regulator (LQR). Both methods revolve around using information about the dynamics and current states of the system to predict the future trajectory of its states. By providing actuation the controller will attempt to influence the trajectory towards the state reference values, along the path that minimizes a cost function defined by the designer. An example of a typical cost function for an LQR is:
I = \int_0^\infty (x^TQx + u^TRu)dt \quad (1)

Where \( x \) is the states vector, \( u \) is the actuation vector, and \( Q \) and \( R \) are vectors containing the relative weights for each state and control variable. By tuning these vectors the controller can be made to prioritize one variable over another, which can completely change its behavior.

The above cost function has its cost integrated for an infinite horizon, and while it is possible to use a finite horizon for an LQR it is a requirement to do so for Model Predictive Control. An MPC will, at each sampling point, predict its future control actions over the entire prediction horizon based on the current state measurement and the cost function. This means that if a prediction horizon of 200 sampling periods is used, the controller will always plan 200 control actions in advance. Only the first control step is actually implemented however, as the controller will redo its calculations each sampling period.

While few real world systems are entirely linear by nature, it often possible (or even required) to linearize the equations that govern a system’s dynamics. Not only does this process open up many useful analytical tools; it also greatly reduces the complexity of a problem, and thus the computational cost for solving it. A large computational cost is problematic for a controller running in real time, as a long computation time will delay the response of the controller accordingly. In terms of overall performance, the increased accuracy from using a non-linear model has seldom outweighed the benefits of using a linearized system. This explains why linear MPC has been so widely used compared to its non-linear counterpart in the past. In recent years however, computer hardware and non-linear optimization algorithms have advanced to a level where an NMPC strategy has become an interesting alternative for many applications. In theory, a non-linear controller should perform much better when there is a wide range of operation, as the accuracy of a linearized equation depends on the distance to the point of linearization. An NMPC can also take non-linear constraints into account, which may be important for describing certain physical limitations of the system.

The recent emerging of NMPC also means that while there is a large amount of available software for linear MPC, there are very few toolboxes for non-linear MPC that are both user friendly and computationally efficient at the same time. This thesis will strictly focus on MATMPC, a packaged developed by Yutao Chen at the University of Padova, that is fully based on MATLAB. It is completely open-source and available at https://github.com/chenyutao36/MATMPC/.

MATMPC uses several techniques that make the optimization problem much easier to solve for the controller. This simplification of the problem is a necessary step for making the computational burden manageable. An example of this is the discretization of the prediction horizon. Using a 4th order Runge-Kutta integrator the prediction horizon is divided into a number of shooting points, or nodes, and the predicted trajectory is compared to the reference values only at these points. The resulting discrete non-linear problem can be written mathematically as

\[
\min_{x,u} \frac{1}{2} \sum_{k=0}^{N-1} ||\eta_k(x_k,u_k)||_W^2 + \frac{1}{2} ||\eta_N(x_N)||_W^2 \quad (2)
\]

so that

\[
x_0 = \hat{x}_0 \quad (3)
\]

\[
x_{k+1} = F(x_k,u_k), k = 0,1,...,N-1 \quad (4)
\]

\[
r_k(x_k,u_k) \leq 0, k = 0,1,...,N-1 \quad (5)
\]

\[
r_N(x_N) \leq 0 \quad (6)
\]
Where $\eta$ is the cost function based on the chosen output parameters and their corresponding weights, defined in the matrix $W$. It is possible to define a different cost function for the last step in the prediction horizon, and it is also possible to just have different weights on the same states for this step, as seen by the distinction between $\eta_k$ and $\eta_N$. This thesis, however, has the same cost function for the final step except it only includes the considered states and not the control input values. $W_N$ is thus just a reduced version of the matrix $W$. The equation also tells us that while we don’t care about what happens in between two nodes, it must always be possible to reach every node from the previous one through the integration represented by $F$, while at the same time satisfying all of the user-defined constraints.

To solve this non-linear problem quickly, MATMPC uses a Real-Time Iteration scheme that performs a single Sequential Programming iteration at each sampling instant. This method consists of linearizing the equation and solving the resulting Quadratic Programming problem. For the linearization point $(x^i, u^i)$, this problem can be written as

$$
\min_{\Delta x, \Delta u} \sum_{k=0}^{N-1} \left( \frac{1}{2} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix}^\top H_k \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} + g_k^\top \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} \right) + \frac{1}{2} \Delta x_N^\top H_N \Delta x_N + g_N^\top \Delta x_N
$$

so that

$$
\Delta x_0 = \hat{x}_0 - x_0,
$$

$$
\Delta x_{k+1} = A_k^i \Delta x_k + B_k^i \Delta u_k + a_k^i, K = 0, \ldots, N - 1
$$

$$
C_k^i \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} \leq c_k^i, k = 0, 1, \ldots N - 1,
$$

$$
C_N^i \Delta x_N \leq c_N^i
$$

where $\Delta x = x - x^i$, $\Delta u = u - u^i$ and

$$
H_k^i = \nabla_{x_k, u_k} \eta_k(x_k^i, u_k^i)^\top \nabla_{x_k, u_k} \eta_k(x_k^i, u_k^i)
$$

$$
A_k^i = \nabla_{x_k} F(x_k^i, u_k^i),
$$

$$
B_k^i = \nabla_{u_k} F(x_k^i, u_k^i),
$$

$$
C_k^i = \nabla_{x_k, u_k} r_k(x_k^i, u_k^i),
$$

$$
a_k^i = F(x_k^i, u_k^i) - x_{k+1}^i,
$$

$$
g_k^i = \nabla_{x_k, u_k} \eta_k(x_k^i, u_k^i)^\top \eta_k(x_k^i, u_k^i),
$$

$$
c_k^i = -r_k(x_k^i, u_k^i)
$$

for $k = 0, 1, \ldots, N - 1$. $\nabla$ is the gradient. The solution of this problem $(\Delta x^*, \Delta u^*)$ is then used to update the controller’s solution to equation 2. The solution is updated through the algorithm

$$
x^* = x^i + \Delta x^*, u^* = u^i + \Delta u^*
$$

These equations were found in [15], where an NMPC was used in conjunction with a low-level controller to control a tethered aerial vehicle. That work uses the NMPC to generate trajectories for the low-level controller to follow; a scheme that was shown to increase the resistance to disturbances. This thesis did not implement that scheme, and we instead directly apply $u^*$ to the CubeSat.

The package also supports a number of different solver algorithms that each have their strengths and weaknesses. While a deeper analysis of the various schemes is outside the scope of this work, the performance of some different algorithms are tested against each other in chapter 3.7.

For a in-depth discussion of MATMPC, see Yutao Chen’s PhD thesis: [3].
1.4 Coordinate systems

A very important decision when designing a controller is the choice of coordinate systems, as this will affect many things such as the equations of system dynamics, numerical properties of the values, and data readability. There exist a number of various coordinate systems that are frequently used for space applications, where some are inertial, some are fixed to the rotation of a body, and some rotate based on the position of a target object.

The design used in this paper relies heavily on the Earth Centered Inertial (ECI) frame. This frame has the middle of Earth in its origin, and its z-axis intersects the two poles pointing north. Both x- and y-axis are confined to the equator, pointing towards distant stars that are considered to have fixed positions. This frame is useful since it’s non-rotating and puts the center of gravity at the origin, greatly simplifying the Newtonian dynamics that characterize the system. Working in a global frame should also allow for having a system with multiple chasers and targets in completely different orbits, which might be difficult when using a target-centered frame. An illustration of the ECI frame is shown below in figure 1.4.1.

![Figure 1.4.1: The Earth Center Inertial (ECI) reference frame. The Z-axis passes through the north pole, while the x- and y-axis pass through the equator line. Origin is always located at the center of the Earth, but the axes are fixed with respect to some distant stars and do not follow the Earth’s rotation.](image)

It is also useful to define a Body-Fixed frame of reference, that moves and rotates along with the chaser. This frame has its origin placed in the center of mass of the chaser satellite, and each axis is normal to two of the six surfaces of the CubeSat body. We will also assume that the chaser’s camera points along the y-axis of the body-frame. An illustration is shown in figure 1.4.2. Since the orientation of the chaser will determine how its thrusters are pointing we must always know the relation between the ECI frame and the Body-Fixed frame.
Lastly, a Local-Vertical Local-Horizon (LVLH) frame is defined. This frame is centered on the target, but the axes do not follow its rotation. Instead, they are determined by the target’s current position in its orbit. The x-axis will always point away from Earth, along the radial direction. The z-axis points normal to the plane of rotation, and the y-axis is the cross product between these vectors. Intuitively, the y-axis can be viewed as the “forward” direction from the target. An illustration is shown below in figure 1.4.3. The chaser with its attached body-fixed frame is also shown in the picture.

Figure 1.4.3: A LVLH frame centered on the target. The axes are independent of the target’s attitude, but are instead a function of its position in the orbit.

1.5 Attitude representation

Just like the satellite’s position can be expressed in several different coordinate systems, there are a number of ways to store and process information regarding its orientation. The traditional way
to do it is with the use of Euler angles, or roll, pitch, and yaw. This method relies on doing a sequence of 3 rotations around the axes of the body, and is considered somewhat intuitive by most people. It can be used to represent almost any orientation, but some angels cause singularity problems that result in the loss of one degree of freedom. Using this method in a control system is computationally expensive due to the all the sines and cosines that must be calculated; this can be especially problematic on low-cost hardware where performance is limited.[8] For space applications, one must also consider that every additional operation that the on-board computer has to perform will increase power consumption and thus heat development. There is therefore strong incitements for keeping the computational load as low as possible.

A more efficient method instead uses the 4-dimensional quaternions to express the attitude. As discussed in [8], there are still relatively few examples of control systems utilizing this technique for attitude control, and even fewer that regulate directly on the quaternion. In the quaternion, one of the four values represent the angle of rotation, while the other three define the axis that the rotation occurs around. This can be expressed mathematically as

\[
\begin{align*}
q_{\text{scalar}} &= \cos \frac{\phi}{2} \\
q_{\text{vector}} &= P \sin \frac{\phi}{2}
\end{align*}
\]

Where \( P \) represents the axis of rotation and \( \phi \) represents the angle of rotation. Any orientation can be expressed without risk of singularities, and the direction of rotation can be controlled by changing the sign of the quaternion. Though it can be more complex to implement, and the simulation results more difficult to interpret, it is apparent that many factors speak in favor of using a quaternion-based attitude control system.

1.6 Literature review

CubeSats and space debris removal are both topics that have been studied intensively the last years. While space missions in general take a long time to prepare, due to the large costs and risks, it is clear that time is not on our side when it comes to the problem of space debris. Apart from the already mentioned study by Liou et al. [13], ESA’s internal studies indicate that initiating debris removal operations in 2060 would be 25% less effective than starting today. [7] The same article points to orbits with an altitude in the 800-1,400 km span as a particularly sensitive region, stating that a series of collisions could make space operations there too dangerous in just a few decades. It is also noted that yearly number of satellites placed into LEO has increased rapidly in recent times; from 72 yearly objects in 2004-2012 to 125 yearly objects in 2013-2016 (largely due to smaller satellites getting more common). Since simulations by both ESA and NASA show that the amount of debris will continue to grow even if all future space launches were suspended, the development of active debris removal technologies have been identified by the European Space Agency as a strategic goal. [6]

While CubeSats are now being considered for a multitude of applications, many of the technologies required for complex missions are still under development. One of these areas is propulsion, as the performance of this subsystem is a main factor in determining how much “work” a satellite can carry out during its lifetime. A study published in 2017 [12] found that a total of less than 10 launched CubeSats have featured propulsion systems. However, the paper also gives an extensive list of propulsion systems still under development today. Based on that information it is reasonable to expect significant advancements in CubeSat propulsion technology in the near future, making the study of potential applications very attractive.
In [9], the theoretical possibilities of performing simple orbital maneuvers using Micro-Cathode Arc Thrusters (µCAT) were examined. These thrusters are both fuel and energy efficient, and also very reliable due to a comparatively simple design. Like most electric propulsion systems the thrust values are however very low, on the order of micro-Newton, and as stated that makes it difficult to properly analyze and control their effect. This study used a simplified orbital analysis based on perturbation theory to show that the output of these thrusters are enough to perform orbit circularization, (very) small inclination changes, and to overcome drag in many situations. One strategy for improving fuel efficiency is to only fire the thrusters during certain parts of the orbit, and this study therefore employed apogee- and perigee-centered burn arcs. It was assumed that the CubeSat had only one thruster that could always be oriented in the most efficient way for the maneuver, but the attitude dynamics themselves were not modeled. While the study was strictly analytic the results suggest that even thrusters in the micro-Newton range are strong enough to perform a variety of simple orbital maneuvers, provided that the thrust can be applied in an efficient manner.

To apply thrust in an efficient manner, and to be able to synchronize its rotational motion with that of the space debris, the chaser satellite will likely need actuators that provide controllability around all its three axes. Many techniques used for conventional spacecraft have in recent years been adapted for use in smaller satellites, and full attitude control systems are now in existence. An example of an attitude control system for nano-satellites can be found in [2]. This system comes in the form of a 5 cm cube, and is therefore small enough to be mounted even on a single-unit CubeSat. It is equipped with three magnetometers, three magnetorquers, and three reaction wheels, allowing it to rotate around all three axes. The reaction wheels measure 0.39 mm x 3 mm, and provide a nominal torque of 210 µNm.

In addition to actuators, the satellite must also be equipped with various sensors for the controller to have access to real time information about the states of the system. To keep costs down a CubeSat may not be equipped with state-of-the-art technology in every respect, but there are examples of cost-efficient solutions that still provide adequate performance. [11] describes how Unscented Kalman Filtering (UKF) can be used for estimating the attitude of a CubeSat even if the sensors are chosen with a strict financial budget in mind. To avoid issues with singularities, the study used a quaternion representation for describing the attitude of the spacecraft. This necessitated an extension of the classical UKF. Simulations showed a pointing accuracy within a few degrees using COTS sensors with a total cost of less than 200 US dollars. Sensors included magnetometers, gyroscopes, and photo-diodes acting as sun sensors. It was found that correctly estimating the bias of the sensors is important for generating accurate measurements, and that attitude uncertainty may be noticeably increased while the satellite is in eclipse due to the lack of data from the sun sensors. The study found that the required computational effort was a large disadvantage of the initial design, but this was mitigated by reducing the number of Runge-Kutta sub-steps used for predictions. The attitude estimation algorithms were implemented on the AAUSAT3 ADCS hardware, which uses a processor of 200 MHz.

The mission phase where accuracy is the most important is during the docking maneuver itself, as the relative motion between the two objects must be close to zero to ensure a soft touch. One way to obtain an accurate estimation of this is with the use of on-board cameras; an example of this being done was the PRISMA mission where two different camera systems was used during a formation flying demonstration, one for long range- and one for short range mode of operation. [5], [20] simulated two cameras in a stereoscopic configuration. This study was focused on the last phase of a docking maneuver with a malfunctioning satellite, with the intent of performing autonomous on-orbit servicing. Both Extended and Unscented Kalman Filtering techniques were employed, and the results showed that using both methods jointly provided benefits when the target has no obvious features that the cameras can detect.

Even more recently it has been shown that if a camera system capable of zooming is used, the
operation range and flexibility of an image based control system can be increased to include far range approaches. [14] presents a vision based orbital and attitude control system used for performing close- and far range rendezvous maneuvers with an uncooperative object. While doing the orbital maneuvers, the controller was simultaneously able to synchronize its attitude with its target. The maximum distance for the far approach maneuvers were 1 km. This study considered a spacecraft with a more conventional mass of 200 kg as the chaser, and the Envisat satellite as the target. The chaser used one main thruster mounted along the z-axis, and a smaller thruster along each of the other axes. Hydrazine, with an $I_{sp}$ of 200 s, was used as fuel. Four reaction wheels were used in a pyramid configuration to provide torque. It was shown both analytically and through simulations that the proposed scheme was stable, and also fuel-efficient enough for the considered maneuvers to be feasible using today’s technology. This gives clear indication that as long as our controlled satellite is able to get within a certain distance of the target, an autonomous docking is achievable. One must however take into account that the thrusters available for a smaller satellite are much weaker than those used in this study, and we can therefore expect that the required time for these maneuvers will be larger on a CubeSat. The weak thrusters also mean that several CubeSats will likely be needed to re-orbit a larger object (Envisat has a mass of over 7,800 kg), and a distributed control system with several chasers utilizing formation flying could potentially accomplish this.

While formation flying is not explored in this thesis, there are books such as [1] on the subject. It has also been shown in a previous LTU Master’s thesis, [17], how MPC can be used to make a swarm of CubeSats fly in formation by maintaining a set distance between each other. This study compared the performance of using MPC versus an LQR-based controller, and found that MPC is suitable for this application due to its ability to incorporate long prediction horizons and counteract disturbances. It was also shown that including the J2-effect in the controller dynamics can lower fuel consumption.

1.7 The goal of this thesis

This work will investigate how a non-linear Model Predictive Control strategy can be used to allow a small spacecraft, such as a CubeSat, to perform a rendezvous maneuver with a non-cooperative target. A simulation environment that takes into account many of the perturbing forces found in the Low Earth Orbits will also be constructed and used to test the performance of the controller.

The advantages of using a non-linear control strategy become more apparent when there is a wide range of operation, and the work will therefore be focused on long range maneuvers such as orbit phasing. It has been proven many times that once the chaser and target are close to each other, their relative motion can be described very accurately using means such as a target centered coordinate system and the Clohessy-Wiltshire equations. One of the main strengths of MATMPC is the possibility of having a very long prediction horizon, and since the orbital phasing part of the mission is the most time consuming it also benefits the most from this fact.

This work will therefore focus on finding an efficient way of getting a small spacecraft close enough to a target to allow for a docking maneuver to be executed. The docking maneuver itself will not be simulated, as the performance of this phase is heavily reliant on the estimation techniques used to determine the relative position, velocity, and attitude of the two objects. While no such estimators will be modeled in this thesis, the literature review has shown several examples of mathematical techniques like Kalman Filtering being used to process sensor data. We also saw in [14] that if a zooming camera is used, getting within 1 kilometer of the target may be enough to transition entirely into an image-based control algorithm.

While active debris removal missions will likely feature several CubeSats flying in formation and cooperating with each other, this work is focused on a single chaser scenario. However, since the simulator and controller are working in an inertial Earth-Centered coordinate system, a future up-scaling of the system should be possible. This choice of coordinate system also makes it convenient to model a great variety of disturbances.
2 System dynamics and modeling

2.1 Dynamics of the environment

The LEO environment is dominated by the Earth’s gravitational field, of which a fairly good approximation can be made by just treating the Earth as a point mass. Creating a more accurate model requires looking at the gravitational field of the Earth more closely, as well as those of the Sun and Moon. At low altitudes, atmospheric drag also becomes a major factor. This chapter will explore all of these aspects in more detail, as they govern the natural behavior of both the chaser and the target. All equations presented in this chapter are discussed in [4].

2.1.1 Earth’s gravitational field

The Earth’s gravitational field is by far the strongest force acting on any orbiting object. Assuming that Earth is a perfectly symmetrical sphere, the acceleration of the body (in $km/s^2$) can be simply written as

$$a_g = -\frac{\mu}{r^3}$$  \hspace{1cm} (22)

The letter $\mu$ denotes Earth’s gravitational parameter, which has a value of 398,600 $km^3/s^2$, and $r$ is the position vector of the object in the ECI-frame, given in $km$.

In reality however, Earth is not a perfect sphere, which means that the gravity field is not symmetric. To account for this one can superimpose additional harmonic terms that model the oblateness of the Earth, giving a more accurate approximation. The most significant term by far is called $J_2$. The resulting acceleration from this effect can be written mathematically as

$$a_{J_2,x} = \frac{3}{2} J_2 \frac{R_E^2}{r^5} x \left( \frac{5}{r^2} - 1 \right)$$ \hspace{1cm} (23)

$$a_{J_2,y} = \frac{3}{2} J_2 \frac{R_E^2}{r^5} y \left( \frac{5}{r^2} - 1 \right)$$ \hspace{1cm} (24)

$$a_{J_2,z} = \frac{3}{2} J_2 \frac{R_E^2}{r^5} z \left( \frac{5}{r^2} - 3 \right)$$ \hspace{1cm} (25)

The symbol $R_E$ is Earth’s equatorial radius; the value used in this thesis is 6,378 km. The symbol $J_2$ is a constant derived from the oblateness of the Earth; it has a value of $1082.63 \times 10^{-6}$. $x$, $y$, and $z$ are used to describe the position of the object on the X, Y, and Z axes of the ECI-coordinate system. The other harmonics are not included in the model, as their relative significance is much lower.

2.1.2 Atmospheric drag

The Earth’s atmosphere plays an important role in orbit perturbation, particularly so for Low Earth Orbits. While its density is much lower there than on sea level, it still contains a considerable amount of particles that, when colliding with the spacecraft, gives rise to a force. The acceleration caused by this is given by
The atmospheric density, \( \rho \), depends on the altitude, the location, and also varies over time. An exact value can therefore only be obtained by using real time measurements. The model constructed here only takes the altitude into account, as that is by far the most significant factor. Data is taken from NRLMSISE-00, for a latitude of 55 degrees, a longitude of 45 degrees, and with the date set to 2017-01-01, 12:00 UT. Above an altitude of 1,000 km the atmosphere is so sparse that its impact is considered negligible.

\[
a_{\text{drag}} = -\frac{1}{2} \rho v_{\text{rel}} \left( \frac{C_D A}{m} \right) v_{\text{rel}}
\]  

(26)

\( v_{\text{rel}} \) is the relative velocity of the spacecraft compared to the atmosphere, and \( C_D \) is the drag coefficient. \( A \) is the area of the spacecraft normal to the direction of movement, and thus depends on the orientation of the satellite. This model assumes a constant value of \( A \), thereby treating the spacecraft as a sphere for this calculation. In reality, this interaction would also give rise to a torque on the satellite depending on the distribution of the frontal area compared to its center of mass. This has not been considered in this model.

### 2.1.3 Lunar gravity

The gravitational field produced by the moon is much weaker than that of the Earth, but its impact can be felt even on the planets surface through phenomena such as tidal waters. As an object travels...
further from Earth, the relative significance of the gravitational fields from other celestial bodies is dramatically increased.

Treating the moon as a point mass, the perturbing acceleration due to the moon can be expressed as

$$a_{\text{moon-sc}} = \mu_{\text{moon}} \frac{r_{\text{moon-sc}}}{r_{\text{moon-sc}}^3}$$  \hspace{1cm} (27)

The lunar gravity’s effect on Earth, and thus the coordinate system itself, can be written as

$$a_{\text{moon-earth}} = \mu_{\text{moon}} \frac{r_{\text{moon-earth}}}{r_{\text{moon-earth}}^3}$$  \hspace{1cm} (28)

The resulting acceleration of the spacecraft in the ECI coordinate system is then

$$a_{LG} = a_{\text{moon-sc}} - a_{\text{moon-earth}}$$  \hspace{1cm} (29)

The symbol $\mu_{\text{moon}}$ is the moon’s gravitational parameter (4,903 $\text{km}^3/\text{s}^2$). The orbit of the moon is well known, so the position vector of the moon with respect to Earth is simply a function of the Julian Date. The Julian Date in turn depends directly on the Universal Time. The Julian Date can be calculated by the MATLAB function found in A.11.

### 2.1.4 Solar gravity and radiation pressure

Although the Sun is very far away, its gravity is still the main factor deciding the orbit of our planet. Naturally this force affects any object in orbit around Earth as well, causing almost the same acceleration to them as it does to Earth. These small differences in acceleration depend on the distance to the Earth center of mass, and will therefore become more noticeable as the altitude increases.

The acceleration done to a spacecraft by the solar gravity can be written as

$$a_{\text{sun-sc}} = \mu_{\text{sun}} \frac{r_{\text{sun-sc}}}{r_{\text{sun-sc}}^3}$$  \hspace{1cm} (30)

Following the same pattern, the acceleration done to Earth (and thus the ECI coordinate system) is written as

$$a_{\text{sun-earth}} = \mu_{\text{sun}} \frac{r_{\text{sun-earth}}}{r_{\text{sun-earth}}^3}$$  \hspace{1cm} (31)

The resulting acceleration of the spacecraft in the ECI coordinate system is then

$$a_{SG} = a_{\text{sun-sc}} - a_{\text{sun-earth}}$$  \hspace{1cm} (32)
The gravitational parameter of the sun is denoted as \( \mu_{\text{sun}} \) and has the value \( 132.712 \times 10^9 \text{ km}^3/\text{s}^2 \). All position values are given in km. The position of the sun can be calculated as a function of the Julian Date, since its orbit is well known.

Another factor to take into account is the solar radiation pressure, caused by the photons emitted from the sun. Although these are massless, they still have energy and momentum that will be transferred to the spacecraft upon impact. The intensity of the solar radiation at a given location depends on the distance to the sun and the solar activity at the time. For simplicity, this model assumes a solar constant of \( S = 1367 \text{ W/m}^2 \) as is often done. The acceleration due to this effect, in \( \text{km/s}^2 \), is given by

\[
a_{\text{SRP}} = -\nu \frac{S}{c} C_R \frac{A_{\text{sc}} r_{\text{sun-sc}}}{m_{\text{sc}} r_{\text{sun-sc}}} \frac{1}{1000}
\]

The symbol \( \nu \) is called the shadow function, and takes the value of 0 or 1 depending on if the spacecraft is in the shadow of the Earth or not. \( C_R \) is the radiation pressure coefficient, ranging from 1 to 2. A 1 means that all the photons are absorbed by the spacecraft, while a 2 means that all the photons are reflected. A photon that is reflected has its momentum reversed, meaning that the force applied to the satellite is doubled. \( A_{\text{sc}} \) is the surface area of the spacecraft normal to the sun; in reality it therefore depends on the orientation of the satellite. This work instead adopts the cannonball model, keeping the value constant as if the body was a sphere. \( c \) is the speed of light, which is approximated to 300,000,000 \( \text{m/s} \). \( m_{\text{sc}} \) is the mass of the spacecraft given in kilograms. The division by 1000 is to get the acceleration in \( \text{km/s}^2 \).

### 2.1.5 Gravity gradient torque

Any asymmetrical object in orbit may experience a torque from the Earth’s gravity field due to the uneven mass distribution. This torque depends on the chaser’s distribution of mass, altitude, and orientation. With the attitude expressed in quaternions, this effect can be expressed as

\[
\begin{align*}
gg_x &= 3 \frac{\mu}{r^3} (I_z - I_y)(2q_2q_3 - 2q_1q_4)(1 - 2q_1^2 - 2q_2^2) \\
gg_y &= -3 \frac{\mu}{r^3} (I_z - I_x)(2q_1q_3 + 2q_2q_4)(1 - 2q_1^2 - 2q_2^2) \\
gg_z &= -3 \frac{\mu}{r^3} (I_x - I_y)(2q_1q_3 + 2q_2q_4)(2q_2q_3 - 2q_1q_4)
\end{align*}
\]

### 2.1.6 Summary of the model

The net acceleration that the chaser will experience from the external forces acting on it can be expressed as follows:

\[
a = a_g + a_{J_2} + a_{\text{drag}} + a_{\text{LG}} + a_{\text{SG}} + a_{\text{SRP}}
\]

For rotational perturbations, this model only considers the gravity gradient torque.
2.2 Dynamics of the chaser

Now that the dynamics of the environment have been examined, it is time to take a look at what the chaser itself could look like. A graphical representation is shown in figures 2.2.1 and 2.2.2. The chaser has one identical thruster mounted on each of its sides, for a total of six, and we assume that they are all aligned with the chaser’s center of mass. It can be seen in figure 2.2.1 that there is a camera mounted on the front of the satellite. It faces the $+y$ direction, and is in this thesis only used for illustrative purposes.

![Figure 2.2.1: The chaser viewed from the front.](image1)

![Figure 2.2.2: The chaser viewed from the back.](image2)
2.2.1 Properties of a CubeSat

The small size of the CubeSat means that it is much cheaper to construct and launch than a conventional spacecraft, but it also means that there are some serious limitations in the design space. A CubeSat is made up of one or several 10x10x10 cm units, and it is allowed a maximum mass of 1.33 kg per unit. This thesis is focused on using a 3U CubeSat with a mass of 4 kg.

The rotational properties of the chaser will mainly depend on its distribution of mass; this must therefore be carefully considered in the design phase. The CubeSat Design Specification requires that the center of mass is within 2 cm of the geometrical center in the x and y directions, while the requirements in the z direction depends on the size of the spacecraft. In many cases it is desirable to have the center of mass as close to the geometrical center as possible.

This work simplifies the problem by assuming that the mass is evenly distributed within the spacecraft. Principal axes are used, which simplifies the rotational dynamics substantially.

The moment of inertia around each principal axis for a homogeneous block can be expressed as

\[
I_x = \frac{1}{12} m (y^2 + z^2)
\]

\[
I_y = \frac{1}{12} m (x^2 + z^2)
\]

\[
I_z = \frac{1}{12} m (x^2 + y^2)
\]

Where \( m \) is the mass of the block in kilograms and \( x, y, \) and \( z \) are the dimensions of the block in meters. Given a mass of 4 kg and dimensions of [0.1, 0.3, 0.1] meters we get the following result to use as an approximation in our simulations:

\[
I_x = 0.0333
\]

\[
I_y = 0.0067
\]

\[
I_z = 0.0333
\]

Where all the values are given in \( kgm^2 \).

It should be emphasized that this does not take into account any objects protruding from the chaser body. This includes solar panels and antennas, which may significantly influence these numbers.

2.2.2 CubeSat propulsion systems

When it comes to propulsion there several limitations on what can be used on a CubeSat. First and foremost the small size of the spacecraft limits the amount of fuel that can be brought on board, which means that a high specific impulse from the propulsion system is desirable to make the most out of every mass unit of propellant. Secondly, the amount of power that can be generated by the
solar panels is also very limited, so the power consumption of the propulsion system must be kept to a minimum. Another thing to consider is that the amount of chemical energy that is allowed on a CubeSat is severely limited by standard regulations, and there are strict safety protocols that must be followed so as to not damage the main payloads of the launch vehicle.

For conventional spacecraft there are many different types of propulsion systems with extensive flight heritage available, including cold gas, chemical, and electrical types. In recent years massive efforts have been spent on adapting these technologies for use on much smaller satellites, but so far there is very limited flight heritage for CubeSats featuring propulsion systems with less than 10 missions in total. [12]

Of the technologies available for CubeSat propulsion, cold gas thrusters currently have the highest maturity level. The CanX-2, a 3U CubeSat flown by The University of Toronto Institute for Aerospace Studies in 2008, featured a cold gas system fueled by $SF_6$ that successfully delivered a thrust of 35 mN with an $I_{SP}$ of 46 s. The technology has since been developed further, and today there are cold gas propulsion systems readily available for purchase. An example of this is VACCO’s Reaction Control Propulsion Module, shown below in figure 2.2.3.

![Reaction Control Propulsion Module](Image)

**Figure 2.2.3**: Characteristics of a commercially available cold gas propulsion system. [19]

A cold gas thruster operates by simply releasing gas particles into the vacuum of space, creating a counter-force that propels the spacecraft forward. While the particles are accelerated through a nozzle to increase the force generated, there is no combustion and only minimal heating of the gas. This means that very low amounts of power are required for operation, and the thrust levels that
can be generated are therefore relatively high. The main disadvantage of this method is that it’s not very fuel efficient when compared to an electrical or chemical thruster.

Electrical propulsion systems have much less flight heritage on CubeSats, and public access to flight data from these more advanced systems is very limited. Contrary to cold gas systems, electrical propulsion relies on using electrical power to add more energy to the particles released. Depending on the type of thruster, this can be done by heating the gas or even ionizing it. This means that a much higher $I_{SP}$ can be achieved at the cost of an increased power consumption. An electrical propulsion system that has flight heritage is the SiEPS developed by MIT. This system flew on the AeroCube-8 (IMPACT) mission in 2015, and featured 8 thrusters capable of delivering a thrust of 74 $\mu$N with an $I_{SP}$ of 1150 s, consuming 1.5 W in the process.[10]

Chemical thrusters are also being developed for CubeSats, but so far there is no system with flight heritage. This propulsion technique has a fuel efficiency that lands somewhere in between that of cold gas and electrical systems, but is capable of delivering thrust levels on the order of Newtons. Many of the propellants used are however toxic, which leads to stricter safety requirements and increased costs in general. Safer green mono-propellants are under development, but issues such as increased ignition temperatures must be solved before they can be effectively utilized.[12]
3 Designing the controller

This section describes how a combined attitude and orbital controller was designed in MATMPC, and includes simulation results from an open loop point of view. This means that the controller and simulated system operate with exactly the same dynamics, and no noise is introduced into the system. The results obtained from this will give us an idea on how the controller performs under ideal circumstances, as well as giving us a benchmark when measuring the controller’s noise resistance.

All tests were made with just one QP iteration, which is needed for the controller to run in real-time. The computation times were obtained on a Windows 7 PC with a Core i5-4670K @ 3.40 GHz and 16 GB DDR3 1600 MHz RAM, using MATLAB R2018a.

3.1 Defining the dynamics and choosing constraints

Since we are building a non-linear controller, the equations from the previous chapter do not need to be linearized. However, we must still make a decision on which parts of the dynamics to include in the controller. Even though today’s processors have vastly more computational power than older models, this power is still not unlimited, especially when it comes to space-qualified processors. Every dynamic not considered by the controller will lead to additional noise in the system, so one must therefore weigh the computational costs and difficulty of implementation against the relative significance for each of the dynamics.

It was decided early on in the design phase to only include the Earth’s gravity in the controller dynamics. As discussed in the previous chapter, a very good estimation of this force can be made by starting from a point mass approximation and adding the $J_2$ term to compensate for the planet’s flattening around the poles. These dynamics are relatively easy to model, and should give reasonable results in most scenarios.

At lower altitudes the atmospheric drag will play a noticeable effect, but there are several difficulties with considering it in the controller dynamics. One is the computational requirements; to get accurate values the controller must calculate its frontal-facing area at each sampling instant, and also determine its velocity relative to the atmosphere at that particular point. Another reason is that it is very difficult to get accurate values on the atmospheric density; as mentioned earlier this parameter depends on many different factors and varies heavily over time.

The effect from the solar and lunar gravity are much easier to model accurately, as their positions are simply a function of time and well known in advance. Even so, including this dynamic this will require calculations of the relative positions of the Sun and Moon over the entire prediction horizon. The relative significance of these dynamics depend on the altitude, but are generally low. One could argue that it would be worthwhile to include them for missions in higher orbits, like GEO, but to save computational power this was not done here. Solar radiation pressure was not included either, as it plays a relatively small impact while also being time-varying and expensive to compute.

Even with these simplifications, the system still requires a relatively large number of states to be represented. The full 19 states can be grouped as follows:

- 3 states for the 3D position in the ECI frame. ($x, y, z$)
- 3 states for the 3D velocity in the ECI frame. ($v_x, v_y, v_z$)
- 3 states for the thrust action along each body frame axis. ($T_x, T_y, T_z$)
- 4 states to store the quaternion of the body frame’s orientation relative to the Earth Inertial Frame. ($q_4, q_1, q_2, q_3$, with $q_4$ being the scalar)
- 3 states to store the angular velocity in the body frame. ($\omega_x, \omega_y, \omega_z$)
3 states for the control torque being applied around each body frame axis. \((\tau_x, \tau_y, \tau_z)\)

As can be seen, the control thrust and torque are considered states instead of controls. The 6 controls are instead defined as the derivative of the control thrust and torque, as this leads to a better controller behavior.

In MATMPC, the dynamics are defined by writing the equation for the derivative of each state. All 19 are written below. It can be seen that the acceleration of the spacecraft only takes into account the gravity from the Earth (including the \(J_2\)) and the force from the thrusters, while the rotational acceleration is determined by the control torque from the reaction wheels and the Earth gravity gradient.

\[
\begin{align*}
\dot{x} &= v_x \\
\dot{y} &= v_y \\
\dot{z} &= v_z \\
\dot{v}_x &= -\mu \frac{x}{r^3} + \frac{15}{2} J_2 \mu \frac{R^2}{r^3} x z^2 - \frac{3}{2} J_2 \mu \frac{R^2}{r^3} x + \frac{T_{x, ECI}}{m} \\
\dot{v}_y &= -\mu \frac{y}{r^3} + \frac{15}{2} J_2 \mu \frac{R^2}{r^3} y z^2 - \frac{3}{2} J_2 \mu \frac{R^2}{r^3} y + \frac{T_{y, ECI}}{m} \\
\dot{v}_z &= -\mu \frac{z}{r^3} + \frac{15}{2} J_2 \mu \frac{R^2}{r^3} z^3 - \frac{9}{2} J_2 \mu \frac{R^2}{r^3} z + \frac{T_{z, ECI}}{m} \\
\dot{T}_x &= \text{control}_1 \\
\dot{T}_y &= \text{control}_2 \\
\dot{T}_z &= \text{control}_3 \\
\dot{q}_1 &= -\omega_y q_2 - \omega_z q_3 \\
\dot{q}_2 &= \omega_y q_1 - \omega_z q_3 \\
\dot{q}_3 &= \omega_y q_1 - \omega_z q_3 \\
\dot{q}_4 &= \text{control}_4 \\
\dot{q}_5 &= \text{control}_5 \\
\dot{q}_6 &= \text{control}_6 \\
\dot{\omega}_x &= \frac{\tau_x + g g_x}{I_x} - \frac{(I_x - I_y) \omega_y \omega_z}{I_x} \\
\dot{\omega}_y &= \frac{\tau_y + g g_y}{I_y} - \frac{(I_y - I_x) \omega_z \omega_x}{I_y} \\
\dot{\omega}_z &= \frac{\tau_z + g g_z}{I_z} - \frac{(I_z - I_x) \omega_x \omega_y}{I_z} \\
\dot{\tau}_x &= \text{control}_4 \\
\dot{\tau}_y &= \text{control}_5 \\
\dot{\tau}_z &= \text{control}_6 \\
\end{align*}
\]

Where \(r\) is the euclidean distance to the center of mass of the Earth and \(gg\) is the gravity gradient torque:

\[
\begin{align*}
\text{(44)}
gg_x &= 3 \frac{\mu}{r^3} (I_z - I_y)(2q_r q_{r3} - 2q_r q_{r4})(1 - 2q_{r1}^2 - 2q_{r2}^2) \\
gg_y &= -3 \frac{\mu}{r^3} (I_z - I_x)(2q_r q_{r3} + 2q_r q_{r4})(1 - 2q_{r1}^2 - 2q_{r2}^2)
\end{align*}
\]
\[ g\dot{z} = -3 \frac{\mu}{r^3} (I_x - I_y) (2q_1q_3 + 2q_2q_4)(2q_1q_3 - 2q_2q_4) \]  

(46)

It is assumed that the chaser has one thruster on each surface of its body, perfectly aligned with the principal axes of the satellite. The thrust vectors intersect the center of mass of the chaser, so no torque is produced when thrust is applied. Note that the thrusters rotate together with the body-frame, so converting the thrust vector to ECI coordinates requires multiplying with a function of the quaternion. The \( T_{ECI} \) symbol is used for these rotated values that are obtained using the following equations:

\[ T_{x,ECI} = (1 - 2\bar{q}_2^2 - 2\bar{q}_3^2)T_x + (2\bar{q}_2\bar{q}_4 + 2\bar{q}_3\bar{q}_4)T_y + (2\bar{q}_4 + 2\bar{q}_3\bar{q}_4)T_z \]  

(47)

\[ T_{y,ECI} = (2\bar{q}_1\bar{q}_2 - 2\bar{q}_3\bar{q}_4)T_x + (1 - 2\bar{q}_1^2 - 2\bar{q}_3^2)T_y + (2\bar{q}_2\bar{q}_3 + 2\bar{q}_1\bar{q}_4)T_z \]  

(48)

\[ T_{z,ECI} = (2\bar{q}_1\bar{q}_3 + 2\bar{q}_2\bar{q}_4)T_x + (2\bar{q}_2\bar{q}_3 - 2\bar{q}_1\bar{q}_4)T_y + (1 - 2\bar{q}_1^2 - 2\bar{q}_2^2)T_z \]  

(49)

Note that, since the quaternion represents the orientation of the chaser with respect to the inertial frame, we must use its conjugate to convert from body-frame to ECI. The quaternion conjugate is represented as \( \bar{q} \), and is obtained by changing the sign of the vector part of the quaternion:

\[ \bar{q} = \begin{bmatrix} \bar{q}_1 \\ \bar{q}_2 \\ \bar{q}_3 \\ \bar{q}_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix} \]  

(50)

Since the quaternions represent how the body-frame of the satellite is rotated with respect to to the ECI frame, the quaternion vector is \([1, 0, 0, 0]\) or \([-1, 0, 0, 0]\) when all their 3 axes are perfectly aligned.

For attitude control, it is assumed that the chaser has one reaction wheel mounted along each axis of its body-frame. The reaction wheels themselves have not been modeled; the controller instead directly applies its control torque to the system.

The units for each orbital-related state was chosen based on the expected numerical values, since MATMPC operates more efficiently if the states have numerical values comparable to each other. Position is given in Megameters, velocity in kilometers per second, and thrust in milliNewtons. This means that some division and multiplication is required in the equations to scale the values appropriately.

Now that the equations of the dynamics have been established, some constraints must be added to simulate the physical limitations of the real world system. Most importantly, all the actuators will have a range of operation dependent on the hardware used on the satellite. Constraints are therefore placed on both thrusters and reaction wheels. It was also found that placing constraints on the maximum angular velocity improved controller behavior by making it more stable. Care must be taken, however, to not set this constraint so low that it stops the chaser from achieving its attitude objectives. When designing the CubeSat it may be desirable to use one stronger thruster to allow for
larger orbit maneuvers, and several smaller ones for precision maneuvering, but this work will assume that all thrusters are of equal size and power. The constraints used for the early simulations are shown in table 3.1.1. Later on in the design process, the constraints placed on the actuators were made even stricter to represent an electrical propulsion system with smaller thrust values.

<table>
<thead>
<tr>
<th>Control and state bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Min value</td>
</tr>
<tr>
<td>Max value</td>
</tr>
</tbody>
</table>

Table 3.1.1: Constraints placed on the state and control parameters for the initial cold gas configuration.

MATMPC also supports constraints on non-linear combinations of the states, making it possible to put a limit on the minimum altitude of the spacecraft. This will limit the possible trajectories of the spacecraft, which might lead to reduced overshoot and less fuel consumed at the cost of a longer settling time. An attempt was made to implement this, but it was not successful.

3.2 Sampling rate and prediction horizon requirements

Due to the vast distances in space, and the low amount of power available on a CubeSat, it can take days or even weeks to perform a larger orbit maneuver. As both debris and chaser are constantly moving and being affected by various forces, it becomes very important to predict their future movements in order to plan future control action. The controller will try to minimize the cost function over the prediction horizon, and so using a long prediction horizon in combination with a good cost function should greatly decrease the amount of fuel spent to intercept the target.

The issue with a long prediction horizon is that it is computationally expensive, especially so for a non-linear system with a large number of states. The computation time depends on many other factors such as the cost function, constraints, and the numerical solver used. There are two factors that determine the length of the prediction horizon; the number of shooting points (amount of control intervals to predict) and the sampling rate (length of each control interval). Multiplying the two values thus gives the length of the prediction horizon. For a given prediction horizon one can dramatically increase the computation speed by lowering the amount of shooting points and instead increase the time between each sample. A slow sampling rate might, however, lead to an increased sensitivity to noise, and make precision maneuvers more difficult. The effect of varying the sampling rate is tested in section 3.5.3.

3.3 Generating reference quaternions on-line

For the chaser to be able to 'see' the target, and not rely solely on external data such as GPS-measurements, it must continuously point its on-board camera at the target. This means that a required reference orientation must be calculated for each sampling instant, and these calculations must be done on-line as the required attitude depends on the position of both chaser and target.

One way to do this is to construct a Direction Cosine Matrix (DCM) that relates the Earth Centered Inertial Frame with the Body-fixed frame of the satellite, which can then be converted into a quaternion representation. To do this, we first need to define the axes of our desired body-frame in terms of these ECI bases:
Assuming that the camera is mounted on the side of the chaser that faces the +y direction of the body-frame, we can make it point at the target by defining the y-axis of the body frame to be aligned with the normalized position difference vector of the two objects in terms of ECI-coordinates. This can be expressed mathematically as

$$BODY_y = \frac{r_{Target} - r_{Chaser}}{|r_{Target} - r_{Chaser}|}$$

(54)

Where \(r_{Target}\) is the position vector of the target and \(r_{Chaser}\) is the position vector of the chaser, both given in ECI-coordinates.

Next we must choose where the other two axes should be pointed. Since we are assuming that the three axes of the body-frame are orthogonal to each other, this choice will in practice determine how the chaser shall be rotated around its own y-axis. This won’t affect its ability to see the target, but it will affect the orientation of the thrusters which in turn has an effect on the fuel efficiency of the satellite. For an interception maneuver it is reasonable to assume that the objects are moving in the same 2D-plane around the Earth, and that the vast majority of control action is used to maneuver within this plane. It is therefore beneficial to have some thrusters oriented so that their entire force is applied within the plane of motion, as any out-of-plane component will have to be canceled by another one of the thrusters. For the simplified configuration used in this paper, this can be achieved by having the z-axis of the chaser be oriented normal to the plane of motion. This will make the x- and y-thrusters operate within the plane of motion, while the z-thrusters can be used to counteract any out-of-plane disturbances.

Since we assume that the chaser and target are in the same plane, we can define the z-axis of the body-frame to be the cross product of their ECI position vectors:

$$BODY_z = \frac{r_{Chaser} \times r_{Target}}{|r_{Chaser} \times r_{Target}|}$$

(55)

To keep the axes orthogonal, the x-axis must be the cross product of the y- and z-axes:

$$BODY_x = BODY_y \times BODY_z$$

(56)

The elements of the DCM that accomplishes this rotation can then be computed as follows:
\begin{align*}
c_{11} &= \text{BODY}_x \cdot \text{ECI}_x \\
c_{12} &= \text{BODY}_x \cdot \text{ECI}_y \\
c_{13} &= \text{BODY}_x \cdot \text{ECI}_z \\
c_{21} &= \text{BODY}_y \cdot \text{ECI}_x \\
c_{22} &= \text{BODY}_y \cdot \text{ECI}_y \\
c_{23} &= \text{BODY}_y \cdot \text{ECI}_z \\
c_{31} &= \text{BODY}_z \cdot \text{ECI}_x \\
c_{32} &= \text{BODY}_z \cdot \text{ECI}_y \\
c_{33} &= \text{BODY}_z \cdot \text{ECI}_z
\end{align*}

And the DCM itself is written as:

\[
\begin{bmatrix}
c_{11} & c_{12} & c_{13} \\
c_{21} & c_{22} & c_{23} \\
c_{31} & c_{32} & c_{33}
\end{bmatrix}
\]

(67)

A DCM can be directly transformed into a quaternion vector. In MATLAB this can easily be done with the ’dcm2quat’ function. One must take care however, as there are 2 unique quaternion vectors correlating to each unique orientation. This can be seen by looking at the definition of the quaternion as a rotation operator:

\[
q_{\text{scalar}} = \cos \frac{\phi}{2}
\]

(68)

\[
q_{\text{vector}} = P \sin \frac{\phi}{2}
\]

(69)

Where \( P \) represents the axis of rotation and \( \phi \) represents the angle of rotation. Since we also know that

\[
cos \frac{\phi}{2} = -\cos \frac{\phi + 2\pi}{2}
\]

(70)

\[
sin \frac{\phi}{2} = -\sin \frac{\phi + 2\pi}{2}
\]

(71)

we can see that even if the sign of the quaternion is inverted, it will still represent the same orientation. For the controller however, a quaternion and its negative are two completely different states, separated by a rotation of 360 degrees. If one does not take this into account, the controller may therefore "spin the wrong way" when trying to reach its reference orientation.

Figure 3.3.1 shows an example of a reference signal that does not take this into account. Predicted future positions of the chaser and target are used to generate a DCM for each time step, and they are then converted into quaternions with the ‘dcm2quat’ function. The two objects are in the same circular orbit, separated by a true anomaly of 10 degrees. Ordering the chaser to always point its y-axis towards the target should correspond to a smooth reference signal, but that is not what we see here. It appears that the function output follows a somewhat random pattern in regards to giving the positive or negative version of the quaternion. This means the chaser might randomly do an
extra rotation every now and then, or wobble back and forth as the reference signal updates, which is not a tolerable behaviour.

![Graphs of q1, q2, q3, and q4 over time steps](image)

Figure 3.3.1: Example of a quaternion reference signal obtained when using the 'dcm2quat' function and not performing any corrections.

A modified custom version of the function, called 'dcm2quat_mod' was also tested. An example of the obtained output is shown in figure 3.3.2. This function sticks to the convention of keeping the scalar quaternion positive, which corresponds to a rotation angle between -180 and +180 degrees. We can now see the cyclic behaviour of the rotation required, but this configuration will force the chaser to do one full extra rotation every time a non-continuous point is reached. While somewhat better than the previous results, this is still not a good behaviour.
This problem can be solved by adding some software that determines which of the two possible quaternion vectors is closest to the chaser’s current orientation, and then correcting the reference value if necessary. This process can then be repeated for each sampling point in the prediction horizon, to always ensure a smooth signal.

The difference between the reference and actual value of the quaternion be computed by means of the error quaternion:

\[
q_{error} = q_{ref} \otimes q_{real}^*
\]  

(72)

Note that the conjugate of the actual quaternion is used. The quaternion multiplication, denoted by \( \otimes \), is performed as follows:

\[
q \otimes p = \begin{bmatrix}
p_4q_4 - p_1q_1 - p_2q_2 - p_3q_3 \\
p_4q_1 + p_1q_4 + p_2q_3 - p_3q_2 \\
p_4q_2 - p_1q_3 + p_2q_4 + p_3q_1 \\
p_4q_3 + p_1q_2 - p_2q_1 + p_3q_4
\end{bmatrix}
\]  

(73)

[8] states that if the first element of the error quaternion is negative, the difference in orientation is more than 180 degrees. This fact can be used to detect the non-continuous points in the reference...
signal, as a rotation difference of more than 180 degrees implies that rotating in the opposite direction would be closer. The sign of the reference quaternion can then be manually flipped at these points, ensuring that the chaser is always told to take the closest rotational path.

The corrected reference signal is shown in figure 3.3.3. This signal is smooth and continuous, and we can therefore expect that the controller will be able to track it.

![Figure 3.3.3: Example of a quaternion reference signal that has been corrected by flipping the sign of the quaternion at the appropriate points.](image)

It can be difficult, however, to interpret the actual orientation of the chaser just by looking at the numbers in the quaternion vector. Even if we convert the quaternions into something more intuitive like Euler angles, it is still not entirely trivial to confirm that the chaser is indeed always pointing towards the target.

To verify that this reference signal accomplishes what it’s meant to do, a 3D-world was connected to the Simulink environment. In there, signals containing the states of chaser and target are assigned to 3d-objects moving in orbit around the Earth, so that the performance can be evaluated visually. An example of this is shown in figure 3.3.4.
Further tests in Matlab has revealed some issues with this approach however. When the chaser is far from the target the rotation required to track it is slow and constant, generating a smooth and continuous reference signal that can easily be followed by the controller. However, once the two objects get closer to each other their relative position changes rapidly, and the out-of-plane component of the position error starts to play a significant role. This means that the generated reference axes for the body-frame may see some very sudden changes, as the current scheme relies on the position error being constrained to a loosely fixed plane of motion. The requested maneuver therefore becomes difficult to perform for the controller, which causes stability issues with its internal predictions.

It is likely that a more sophisticated algorithm must be used to track the target at close range, as the goal is to ultimately synchronize its rotation with that of the target. The algorithm should therefore use both the orientation and relative position of the objects as inputs. That reference generator must also be able to compute a safe and efficient approach vector, meaning that we can no longer simply use the position of the target as reference value for the controller.

For the tests presented in this thesis, the chaser is instead instructed to point its camera along its own velocity vector. This means that one pair of thrusters will also be aligned with the velocity vector, which should be beneficial when analyzing the results.

Another issue regarding the sign of the quaternion was later noticed. As stated earlier, we can ensure that the chaser will always chose the closest direction of rotation by comparing the generated reference value with the current quaternion state, as shown in equation 73. This method does not take the angular velocity into account however, and if the chaser is not able to accelerate its rotation quickly enough it may get stuck in an endless loop going back and forth. To solve this, the algorithm was changed to look at the last reference values instead of the chaser's actual states. This method guarantees a smoother reference signal, and was used without issues in all the simulations presented here.
3.4 Intercepting the target using a 2 phase maneuver

It is not enough for the chaser to intercept the target from hundreds, or even thousands, of kilometers away; it must then perform a high precision maneuver and "dock" with the space debris in a controlled fashion. It is likely that several CubeSats are required to de-orbit larger objects, and the docking process must therefore not disturb the motion of the target too much.

These two different phases have very different characteristics, and therefore require different control strategies to solve. The first phase, where the chaser is far from its target, relies on accurately predicting how the states will evolve for a very long time and then using this information to efficiently guide the chaser towards the target. Precision is not of extreme importance in this phase, and a slower sampling rate can therefore be used to increase the length of the prediction horizon. One must also consider that a longer sampling period gives more time for the on-board computer to perform its calculations, potentially allowing for an even longer prediction horizon or less expensive components.

The second phase, where the chaser has already gotten close to its target, lasts for a much shorter time and does not require a prediction horizon that spans for days. A shorter sampling time can therefore be used to increase the controllers responsiveness, and potentially allow for a smooth docking maneuver.

3.5 Long range

In this section, we will examine how the controller performs in a scenario where the chaser starts far from the target, and has to perform an orbital phasing maneuver in order to reach it. We will assume that the target is in a circular equatorial orbit, with a semi-major axis of 7,800 km. This puts it in the upper region of LEO, where there is still a high concentration of debris, but outside the region where atmospheric drag plays a significant role. The chaser will start in the same orbit as the target, but with a negative offset in true anomaly. The initial orbital parameters are summarized in table 3.5.1. In the initialization process these are converted into a state space representation; the corresponding initial states are shown in table 3.5.2.

The chaser starts with its body-frame being aligned to the ECI-frame and with zero angular velocity. A state space representation of this is shown in table 3.5.3.

A graphical representation is shown in figure 3.5.1. Note that the size of the two objects have been greatly increased in this picture, as they are just a few centimeters large in reality, and are separated by a distance of almost 2,500 km.

The weights used in the cost function for this test are shown in table 3.5.4. As seen in later sections, these values where changed frequently to further improve the fuel efficiency and other aspects.
Figure 3.5.1: Initial conditions of the long range scenario, shown graphically.

<table>
<thead>
<tr>
<th>Initial orbital parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Semi-major Axis</strong></td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td><strong>Chaser</strong></td>
</tr>
<tr>
<td><strong>Target</strong></td>
</tr>
</tbody>
</table>

Table 3.5.1: Initial orbital parameters for the long range scenario.

<table>
<thead>
<tr>
<th>Initial orbital states</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x-pos (Mm)</strong></td>
</tr>
<tr>
<td>Chaser</td>
</tr>
<tr>
<td>Target</td>
</tr>
</tbody>
</table>

Table 3.5.2: Initial orbital states of the chaser and target for the long range scenario.

<table>
<thead>
<tr>
<th>Chaser initial attitude states</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>q_4</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.5.3: Initial attitude states of the chaser for the long range simulation.

For the first scenario, the chaser and target will be separated by a true anomaly of 18 degrees, or \( \frac{1}{20} \) of an orbit. This corresponds to 2,440 km at the present altitude of 7,800 km. To cover this distance...
in an efficient way, the chaser must slow down and temporarily enter a lower orbit where the orbital period is shorter. By timing it right, it can then increase its speed and intercept the target in its original orbit.

<table>
<thead>
<tr>
<th>Weight on the cost function</th>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>3</td>
<td>2,000</td>
<td>6</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 3.5.4: Cost function weights for the first test.

Figure 3.5.2 shows the relative distance and velocity between the chaser and target, as well as the difference in semi-major axis between them. 26.3 hours into the simulation the chaser is less than 9 km away from the target, and they have a relative velocity of just over 3 m/s. However, the semi-major axis of the chaser is almost 20 km shorter than that of the target at this point, which causes their relative distance to increase again temporarily. It takes 40.7 hours to get within one kilometer of the target. The simulation lasted for 200,000 seconds (55.5 hours), at which point the chaser was half a meter away from the target with a relative velocity of less than 0.5 mm/s.
Figure 3.5.2: Illustration of how the chaser moves compared to the target. It is clear that it can catch up the target by reducing its orbital radius, and it ends the simulation roughly 0.5 meters from the target.

To show the components of the relative position vector, the LVLH-frame is convenient. The position of the chaser in the LVLH-frame is shown in figure 3.5.3. At $t = 26.3$ hours the $y$-component is less than 1 km from the target, but the $x$-component (corresponding to the radial direction) is almost 10 km too small. Due to the chaser being closer to Earth than the target is at this point in time, it overshoots and has to enter a higher orbit temporarily to let the target catch up. The LVLH-frame is also useful for showing the relative trajectory in 3D. Figure 3.5.4 shows the later half of the trajectory, including the mentioned point where the objects pass close to each other before the chaser has to temporarily enter a higher orbit. A zoomed in plot of the last part of the maneuver is shown in figure 3.5.5.
Figure 3.5.3: The position of the chaser in the target-centered LVLH-frame.
Figure 3.5.4: The trajectory of the chaser in the target-centered LVLH-frame. The red dot at [0,0,0] represents the target.
Figure 3.5.5: The trajectory of the chaser in the target-centered LVLH-frame, zoomed in to show the last parts of the maneuver. The red dot at [0,0,0] represents the target.

While the chaser is successfully able to rendezvous with the target, the thrusters spend a majority of the time saturated. This is shown in figure 3.5.6. Even the z-thruster, whose applied force is perpendicular to the plane of motion, constantly switches between its maximum and minimum bounds. This indicates that the weights put on the derivative of control thrust are too low, which may not be very fuel efficient. A total of 420.1259 Ns was supplied by the thrusters over the duration of the simulation.
The attitude of the spacecraft can be shown in terms of quaternions as in figure 3.5.7. At the start of the simulation the chaser’s body-frame is aligned with the ECI-frame; it therefore has \([1, 0, 0, 0]\) as its initial quaternion states. Since the orbit is equatorial, the rotation should occur mainly around the z-axis. This is what happens in the simulation. The reference tracking looks good apart from some minor issues early in the simulation, but it is difficult to tell in this plot.
Another way of representing the attitude error is with the error quaternion. This is done in figure 3.5.8. Since there is no noise in the system the errors eventually goes to zero, but this only happens after the 30 hour mark.
The control torque and angular velocity of the chaser is shown in figure 3.5.9. As expected, only the z-component of the angular velocity is non-zero at the end of the simulation. However, the slow sampling rate forces all three reaction wheels to be used at the start of the simulation in order to match both the reference angular velocity and reference orientation at once.
Figure 3.5.9: The torque being applied by the reaction wheels. The saturation point of 0.5 μNm is reached several times during the first few hours.

### 3.5.1 Tuning the weights of the cost function

The configuration used in chapter 3.5 shows some clear issues with saturation of the actuators. The weights on the cost function were therefore altered, with the intention of making the controller prioritize higher fuel efficiency over a shorter mission time. The weights on the quaternion states were also lowered. The original configuration, as well as four new cases, are shown in table 3.5.5. The fuel consumption for each case is also shown.
Weights on the cost function

<table>
<thead>
<tr>
<th>Case</th>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>3</td>
<td>2,000</td>
<td>6</td>
<td>500</td>
<td>420.13 Ns</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.01</td>
<td>1,000</td>
<td>6,000</td>
<td>500</td>
<td>456.54 Ns</td>
</tr>
<tr>
<td>3</td>
<td>1e-3</td>
<td>1e-4</td>
<td>1,000</td>
<td>6,000</td>
<td>500</td>
<td>276.53 Ns</td>
</tr>
<tr>
<td>4</td>
<td>1e-4</td>
<td>1e-5</td>
<td>100</td>
<td>6,000</td>
<td>500</td>
<td>243.28 Ns</td>
</tr>
<tr>
<td>5</td>
<td>1e-3</td>
<td>1e-4</td>
<td>1,000</td>
<td>60,000</td>
<td>500</td>
<td>237.05 Ns</td>
</tr>
</tbody>
</table>

Table 3.5.5: 5 cases with different cost function weights and corresponding thruster usage.

Figure 3.5.10 shows the trajectories in the LVLH-frame that were obtained when testing the different cost functions. This figure is zoomed in to highlight the differences in overshoot between the cases. The overshoot magnitude is similar for cases 1-3 at roughly 150 km, despite the large differences in actuation usage. In case 5 the overshoot magnitude is 100 km greater, while case 4 lands in between at an amplitude of 200 km. The components of the relative position vector over time are shown in figure 3.5.11. The distance to target and relative semi-major axis are shown in figure 3.5.12.

In the two lower subplots the later half of the simulation has been zoomed in to illustrate the differences. Case 1 is the only configuration that manages to equalize the semi-major axes within the simulation time, but it consumes above 50% more fuel than any of the three last cases.
Figure 3.5.10: Trajectories in the LVLH-frame for the 5 tested cases of different cost functions.
Figure 3.5.11: The relative LVLH position vector, split into its components, for the 5 tested cases of different cost functions.
Figure 3.5.12: Orbital trajectories for the 5 tested cases of different cost functions.

Figure 3.5.13 shows how the thrust is applied in each case. In all cases we experience some form of saturation, but the extent varies greatly. Only in case 4 are the control signals smooth over the entire simulation time, and it was therefore selected for further study. Case 5 consumes slightly less fuel, but also ends the simulation further away from the target. When the simulation time was increased for some further tests, it also experienced stability issues. No such issues were experienced for case 4. The more conservative configurations use the y-thruster considerably more than the x- and z-thrusters. The y-thruster is pointed along the velocity vector, and we can see that it is initially used to slow down the spacecraft, entering a lower orbit, and then switched to a positive value around the 15-hour mark. The z-thruster is used sparingly; in case 4 it is only responsible for 1.2% of the total fuel spent. The x-thruster is responsible for 51.51%, and the y-thruster 47.29%.
Figure 3.5.13: Control thrust used in the 5 tested cases of different cost function weights.

The control torque and angular velocity for case 4 is shown in figure 3.5.14. The error quaternion is shown in figure 3.5.15.
Figure 3.5.14: Control torque and angular velocity in case 4.
3.5.2 Length of the prediction horizon

This test aims to find a first indication of how performance is affected when the length of the prediction horizon is modified. As we saw in the previous test, it can take many hours to reach the target when an orbital phasing maneuver is required. The target is constantly moving during this time, so predicting its future position accurately for a longer time should have a positive effect on the controller’s performance.

The same scenario as last time was used, but the simulation time as now increased to 500,000 seconds (138.89 hours). The controller uses the same weights as case 4 in table 3.5.5. Nine different numbers of shooting points were tested, these are shown in table 3.5.6. As the prediction horizon is lengthened, the computation time required keeps increasing. There is no clear trend with regards to actuation spent, except for the first 2 cases where consumption is far above average.
### Number of shooting points vs computation time and thruster usage

<table>
<thead>
<tr>
<th>Case</th>
<th>N</th>
<th>Pred. Horizon length</th>
<th>avg cpt</th>
<th>max cpt</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>8.33 h</td>
<td>12.99 ms</td>
<td>20.41 ms</td>
<td>614.28 Ns</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>11.11 h</td>
<td>17.05 ms</td>
<td>27.46 ms</td>
<td>439.30 Ns</td>
</tr>
<tr>
<td>3</td>
<td>250</td>
<td>13.89 h</td>
<td>21.34 ms</td>
<td>37.75 ms</td>
<td>349.18 Ns</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>16.67 h</td>
<td>26.05 ms</td>
<td>46.39 ms</td>
<td>392.37 Ns</td>
</tr>
<tr>
<td>5</td>
<td>350</td>
<td>19.44 h</td>
<td>31.84 ms</td>
<td>59.63 ms</td>
<td>405.47 Ns</td>
</tr>
<tr>
<td>6</td>
<td>400</td>
<td>22.22 h</td>
<td>38.13 ms</td>
<td>71.91 ms</td>
<td>399.41 Ns</td>
</tr>
<tr>
<td>7</td>
<td>450</td>
<td>25.00 h</td>
<td>42.65 ms</td>
<td>91.52 ms</td>
<td>371.42 Ns</td>
</tr>
<tr>
<td>8</td>
<td>500</td>
<td>27.78 h</td>
<td>47.80 ms</td>
<td>93.01 ms</td>
<td>354.68 Ns</td>
</tr>
<tr>
<td>9</td>
<td>550</td>
<td>30.56 h</td>
<td>53.70 ms</td>
<td>103.79 ms</td>
<td>376.65 Ns</td>
</tr>
</tbody>
</table>

Table 3.5.6: Nine cases with different number of shooting points (N). The total amount of control thrust used in each case, as well as the maximum and average computation time for each step, is also shown.

Figures 3.5.16 and 3.5.17 below illustrate how the relative movement, in the LVLH-frame, between chaser and target is affected by changing the number of shooting points. The distance and relative semi-major axis are shown in figure 3.5.18. It is clear that a longer prediction horizon generally leads to a smaller overshoot and faster settling time, but the differences become smaller as the prediction horizon is increased further. Between cases 5 and 8 we can see very little difference in terms of trajectory, but the actuation usage in case 8 is more than 10% lower than in case 5. In case 9 the actuation usage once again increases, and further increases to the prediction horizon cause the controller to lose stability and the simulation to end. In case 1 and 2 the chaser is not able to reach the target within the simulation time of 200,000 seconds.
Figure 3.5.16: Relative motion between chaser and target in five of the tested cases with different numbers of shooting points.
Figure 3.5.17: Components of the relative motion between chaser and target in five of the tested cases with different numbers of shooting points.

Over all, these results indicate that increasing the number of shooting points provides substantial benefits to controller behavior, but only up to a certain point. After this point the controller faces stability issues, which may be linked to the increased actuation usage in case 9.
Figure 3.5.18: Distance to target and relative semi-major axis for the 9 tested cases with different numbers of shooting points.

3.5.3 Sampling rate and number of shooting points

As mentioned previously, the length of the prediction horizon is the product of the number of shooting points and the sampling time. Computationally it is often less expensive to use a longer sampling time and lower amount of shooting points than vice versa [3], but it is difficult to predict to what extent this will affect the performance of the controller. The results obtained in chapter 3.5.2 indicated that the length of the prediction horizon will play an important role for the controller’s performance, but that it also affects its stability. The greater computational load associated with more shooting points may also prove to be a limiting factor if those stability issues are resolved.

To see if it is possible to reduce the computational load without negatively affecting performance, a new round of simulations were performed in Matlab. Case 6 from table 3.5.6 was used as a starting point, and the length of the prediction horizon was kept constant for all cases. Instead, the sampling rate was varied and the number of shooting points adjusted correspondingly. It was found that increasing the sampling time lead to controller instability, and only faster sampling rates than
the original could therefore be tested. This means that the number of shooting points had to be increased to keep the prediction horizon at a constant length. The result is a longer computation time, but also significant differences in actuation usage. The data is summarized in table 3.5.7. A sampling time of 100 seconds uses the least amount of fuel, while a sampling time of 50 seconds uses the most. In case 2 the prediction horizon is only half that of the other cases. It uses roughly as much computational power as case 1, but performs notably worse in all other aspects.

The orbital trajectories are shown in figure 3.5.19. Apart from case 2, which is not able to reach the target, they behave similarly despite the large differences in actuation usage. The actuation usage is shown in figure 3.5.20. It can be noted that with the exception of case 2, nearly all of the differences lie in the utilization of the x-thrusters. The trajectory in LVLH-frame is shown in figure 3.5.21 for case 1, 3, and 4. There are visible differences between the trajectories, though the general shape is the same. The components of the LVLH position vector are shown in figure 3.5.22. The oscillations of the z-component have the same period in all cases, but different amplitude. Case 3 also has a phase difference of 180 degrees compared to the other cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>N</th>
<th>Sampling time</th>
<th>avg cpt</th>
<th>max cpt</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>400</td>
<td>200 s</td>
<td>38.13 ms</td>
<td>71.91 ms</td>
<td>399.41 Ns</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>100 s</td>
<td>38.96 ms</td>
<td>72.40 ms</td>
<td>423.15 Ns</td>
</tr>
<tr>
<td>3</td>
<td>800</td>
<td>100 s</td>
<td>81.33 ms</td>
<td>188.15 ms</td>
<td>325.26 Ns</td>
</tr>
<tr>
<td>4</td>
<td>1600</td>
<td>50 s</td>
<td>173.74 ms</td>
<td>415.62 ms</td>
<td>462.86 Ns</td>
</tr>
</tbody>
</table>

Table 3.5.7: Three cases with a prediction horizon length of 22.22 hours, but different sampling rates and number of shooting points (N). The total amount of control thrust used in each case, as well as the maximum and average computation time for each step, is also shown. Case 2 has only half the prediction horizon length as the other cases; it uses the same computation time as case 1 but performs notably worse in fuel efficiency and settling time.
Figure 3.5.19: Distance to target and relative semi-major axis over time, for the four tested cases of different sampling rate and number of shooting points.
Figure 3.5.20: Thruster usage for the four tested cases of different sampling rate and number of shooting points.
Figure 3.5.21: Trajectory in the LVLH-frame for the three cases with equal prediction horizon length.
Figure 3.5.22: Components of the chaser’s relative position vector in the LVLH-frame, for the three cases with equal prediction horizon length but different sampling rate and number of shooting points.

One might expect that added computational load should always translate to better, or at least equal, performance, but this did not happen here. While the configuration running with a sampling rate of 100 seconds and 800 shooting points (case 3) performs notably better than the original case, using a sampling rate of 50 seconds and 1600 shooting points leads to a worse performance than both of the previous cases. Further investigation is required to fully understand the cause of these unexpected results. It is possible that this more complicated problem statement requires additional iterations by the controller to find a good solution, and it is also likely that these results would be different if noise was present in this simulation. It is clear, however, that a larger computational effort doesn’t always translate to better performance, and that a substantial amount of computational power can easily be wasted if the controller parameters are not tuned properly.
3.6 Short range

This section explores the behavior of the controller when the chaser starts much closer to the target. As the need for precision is expected to be greater in this phase, the controller will run with a faster sampling rate of 10 seconds. The number of shooting points is kept at 800, giving a total prediction horizon length of 8,000 seconds. This means that the prediction horizon is ten times shorter than in the long range phase, but since the close range phase is expected to last for a much shorter time that should not be an issue. Two different scenarios are tested, both using initial conditions based on data from the same long range simulation.

Through testing it was found that the weights related to the position and velocity errors had to be greatly increased to ensure a rendezvous. If the costs are not increased, the chaser maintains the same distance to the target as it had in the end of the long range scenario. The weights used for the short range scenarios are shown in table 3.6.1.

<table>
<thead>
<tr>
<th>Weights on the cost function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
</tr>
<tr>
<td>100,000</td>
</tr>
</tbody>
</table>

Table 3.6.1: Cost function weights used in the short range simulation.

3.6.1 Late transition

For the first test, the final states from case 3 in table 3.5.7 are used as initial conditions. The orbital related parameters are shown in table 3.6.2. This means that the chaser starts just over 5 km from the target, with a relative velocity of less than 2.5 m/s.

<table>
<thead>
<tr>
<th>Initial orbital states</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-pos (Mm)</td>
</tr>
<tr>
<td>Chaser</td>
</tr>
<tr>
<td>Target</td>
</tr>
</tbody>
</table>

Table 3.6.2: Initial orbital states of the chaser and target for the close range simulation.

Figures 3.6.1 and 3.6.2 below show the relative position and velocity between the chaser and target during the simulation. After 4.18 hours, the distance between them is just 5.8 meters, and their relative velocity is less than 5 mm/s. This means that the approach is slow and smooth, which is exactly what is needed for a successful docking. The trajectory in the LVLH-frame is shown in figure 3.6.3. Here we can see that the approach happens in the shape of a spiral.
Figure 3.6.1: Distance, relative velocity, and difference in semi-major axis between the chaser and target in the short range simulation.
Figure 3.6.2: Relative motion of the short range case shown in the LVLH-frame.
Figure 3.6.3: Trajectory in the LVLH-frame for the short range phase.

The total actuation need for the thrusters with this configuration is 13.99 Ns. While this value can no doubt be further optimized, it is clear that the vast majority of the fuel is spent during the long range phase of the mission. The thruster usage is plotted in figure 3.6.4. The workload is spread almost evenly between the x- and y-thrusters; the x-thrusters consume 48.48% of the fuel while the y-thruster consumes 51.47%. The z-thrusters consume just 0.05% of the total fuel used.
Rotation-wise, the satellite experiences a very small orientation error at the start of the simulation that it quickly corrects, and it then tracks the reference value perfectly. As the chaser begins with a correct attitude compared to the reference value, this small error is believed to by the shooting points from the long range scenario not being exported to the close range scenario. The error quaternion is shown in figure 3.6.5, while the angular velocity and control torque is shown in figure 3.6.6.
Figure 3.6.5: Error quaternion for the short range simulation
Figure 3.6.6: Angular velocity and control torque in the short range scenario.

Figure 3.6.7 shows the region of transition between the two modes, with the different colors indicating the point of transition. Within this timespan of 4 hours, the distance to the target decreases from 3-5 km to just 35 meters. The thruster usage for the two phases combined is shown in figure 3.6.8.
Figure 3.6.7: Relative distance, velocity, and semi-major axis in the transition region.
The total fuel usage when combining the two phases is 339 Ns, out of which only 14 Ns is spent in the short range case.

### 3.6.2 Early transition

We have now seen that we can make a smooth transition to short range mode once we have let the long range mode bring the chaser close to a steady state. However, in the long range simulations we have seen that the chaser will get very close to the target before overshooting it, and a transition might be possible at those points. This could reduce the amount of actuation used in the long range phase significantly, as it never reached a zero-output steady state during the simulation time of almost 140 hours (see case 3 in figure 3.5.20). Figure 3.6.9 contains a plot of the relative distance during the entire long range phase of the simulation, as well as zoomed in regions where the objects are pass close by each other. At the first point, where \( t=27.7 \) hours, the chaser is 10.76 km from the target, while at the second point, at \( t=49 \) hours, they are at one instant separated by a distance of just 5 km.
Figure 3.6.9: Distance between chaser and target during the long range phase. Subplots 2 and 3 are zoomed in at regions where they are close to each other.

Short range simulations using initial conditions from both of the potential transition regions were performed. In the first test, using the states from $t=27.8$ hours, the controller could not find a solution and the simulation stopped. The orbit-related initial states from the second case, using states from $t=50$ hours, are shown in table 3.6.3. All controller parameters were kept identical to the ones shown previously in table 3.6.1.

<table>
<thead>
<tr>
<th>Initial orbital states</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-pos (Mm)</td>
</tr>
<tr>
<td><strong>Chaser</strong></td>
</tr>
<tr>
<td><strong>Target</strong></td>
</tr>
</tbody>
</table>

Table 3.6.3: Initial orbital states of the chaser and target for the close range simulation starting at $t=50$ hours.
The trajectory in the LVLH-frame is shown below in figure 3.6.10, and the components of the position vector are shown in figure 3.6.11. Compared to the previous case there is more movement in the z-direction, and the chaser does not immediately enter the spiral pattern previously observed. If we zoom in closer to the target though, we can see the same spiral pattern. This is shown in figure 3.6.12.

The relative distance, velocity, and semi-major axis are shown in figure 3.6.13. After 6.4 hours, the chaser is within one meter of the target with a relative velocity of 3.67 mm/s. It is clear that even with this earlier transition to the close range mode, the chaser is able to rendezvous with the target. Significantly more actuation is spent in the short range phase though, with a total of 32.95 Ns compared to the 13.99 Ns used in the earlier simulation. The control thrust is shown in figure 3.6.14. Contrary to the previous case, the z-thrusters are responsible for 10% of the total actuation used. The rest of the workload is split fairly evenly between the x- and y-thrusters, whose respective consumptions are 46.69% and 43.17% of the total.

Figure 3.6.10: Trajectory in the LVLH-frame in the close range phase initialized at t=50h.
Figure 3.6.11: Components of the LVLH-position vector in the close range phase initialized at $t=50\,\text{h}$. 
Figure 3.6.12: Trajectory in the LVLH-frame in the close range phase initialized at t=50h. This has been zoomed in to show the spiral pattern with which the chaser approaches the target once its close enough.
Figure 3.6.13: Relative distance and velocity in the close range phase initialized at t=50h.
The attitude errors expected at the start of the simulation were significantly larger than in the previous case. The error quaternion is shown in figure 3.6.15 and the angular velocity as well as the control torque is shown in figure 3.6.16. A large amount of control torque is applied during the first 1.5 hours; this did not happen in the previous scenario. This may be connected to the heavy usage of the z-thruster, as that brings the chaser slightly out of plane.
Figure 3.6.15: Error quaternion in the close range phase initialised at $t=50h$. 
Figure 3.6.16: Control torque and angular velocity in the close range phase initialized at t=50h.

Figure 3.6.17 shows the region of transition between the two modes, with the different colors indicating the point of transition. Within this time span of 4 hours, the distance to the target decreases from 3-5 km to just 35 meters. The thruster usage for the entire simulation is shown in figure 3.6.18.
Figure 3.6.17: Relative distance, velocity, and semi-major axis in the transition region when doing an earlier transition to short range mode.
Figure 3.6.18: Thruster usage for the long and short range phases (when doing an early transition) combined into one plot.

### 3.6.3 Comparison of the two transition points

<table>
<thead>
<tr>
<th>Case</th>
<th>Time&lt;sub&gt;LR&lt;/sub&gt;</th>
<th>Thrust&lt;sub&gt;LR&lt;/sub&gt;</th>
<th>Time&lt;sub&gt;SR&lt;/sub&gt;</th>
<th>Thrust&lt;sub&gt;SR&lt;/sub&gt;</th>
<th>Time&lt;sub&gt;TOT&lt;/sub&gt;</th>
<th>Thrust&lt;sub&gt;TOT&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>118 h</td>
<td>325.26 Ns</td>
<td>4 h</td>
<td>13.99 Ns</td>
<td>122 h</td>
<td>339.25 Ns</td>
</tr>
<tr>
<td>2</td>
<td>50 h</td>
<td>217.70 Ns</td>
<td>4 h</td>
<td>32.96 Ns</td>
<td>54 h</td>
<td>250.65 Ns</td>
</tr>
</tbody>
</table>

Table 3.6.4: Comparison between the two transition points in terms of mission time and thruster usage.

In table 3.6.4 the performance of the two transition points is compared. Transitioning early reduces the total actuation amount by more than 25%, as well as the mission time by more than 50%. The larger attitude errors experienced in the early transition scenario, as well as the larger usage of the
z-thruster, indicates that it may be necessary to further tune the weight of the cost function. It seems probable that different weights will be required depending on the initial conditions of the short range phase.

We can conclude that the current implementation of the controller should be able to handle both short and long range scenarios, but switching between the scenarios requires some tuning with respect to cost function, sampling rate, and number of shooting points. For a controller utilizing two distinct modes of operations, the point at which we make the transition will play a large role in determining fuel efficiency. Since the short range mode will rely on using the camera for estimations, its maximum range should also be taken into account.

### 3.7 Solver comparison

All the simulations shown in this thesis were made with the HPIPM Sparse solver, as it proved to be the faster one of only two controllers that were able to handle the test scenarios. The tested scenarios were the same as the long and short range phases previously analyzed.

IPOPT Sparse was used previously to this, in tests that are not showcased in this paper. The vast differences in computation time, shown in 3.7.1, shows how important it is to chose the correct solver for the job. HPIPM was on average around 15-20 times faster than IPOPT, depending on the scenario. Several of the solvers supported by MATMPC, such as quadprog and OSQP, didn’t work in either case, and only those two solvers were able to tackle the long range scenario. Thruster usage is very similar across most solvers, with the exception of the qpOASES solver utilizing a moving block strategy. The same holds true for the orbital trajectory of the chaser, though it appears that HPIPM gives a somewhat lower settling time in the short range scenario.

<table>
<thead>
<tr>
<th>Short range</th>
<th>$T_s = 10$</th>
<th>Solver</th>
<th>avg cpt (ms)</th>
<th>max cpt (ms)</th>
<th>thruster usage (Ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPOPT Partial Condensing, N2=50</td>
<td>184.3297</td>
<td>292.6718</td>
<td>20.8107</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPOPT Sparse</td>
<td>444.7861</td>
<td>715.8357</td>
<td>20.8031</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qpOASES with Moving Block</td>
<td>209.3465</td>
<td>300.0086</td>
<td>24.6205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPIPM Sparse</td>
<td>28.6444</td>
<td>46.3188</td>
<td>20.8038</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPIPM pcond</td>
<td>29.2121</td>
<td>43.3643</td>
<td>20.8036</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Long range</th>
<th>$T_s = 200$</th>
<th>Solver</th>
<th>avg cpt (ms)</th>
<th>max cpt (ms)</th>
<th>thruster usage (Ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPIPM Sparse</td>
<td>41.2722</td>
<td>257.0004</td>
<td>152.8454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPOPT Sparse</td>
<td>649.1575</td>
<td>1269.1057</td>
<td>154.0323</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.7.1: Comparison between different solvers regarding thruster usage and computation times. The trajectories were very similar in all cases.

### 3.8 Relationship between the cost function, length of the prediction horizon, and thruster limitations

#### 3.8.1 Long range

In the tests performed in chapter 3.5.2, it was found that the controller lost stability when the prediction horizon was set to more than 30.56 hours. The results also indicated that after a certain point, further increases to the number of shooting points does not improve the controller’s performance.

Further testing of different weights on the cost function, for varied lengths of the prediction horizon, revealed that there is a correlation between the cost function weights and the prediction horizon requirements. When the controller is tuned to consume less fuel it needs a longer time to reach the target, and this in turn resulted in a need for a longer prediction horizon to optimize the fuel.
efficiency. This seems reasonable, as the standard practice is to keep the prediction horizon no less than a set percentage of the controller’s settling time. It also highlights the inter-dependency of the various controller parameters, and the iterative approach used when designing this control system.

### Controller weights and parameters

<table>
<thead>
<tr>
<th>Case</th>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
<th>N</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1e-7</td>
<td>1e-7</td>
<td>1</td>
<td>30</td>
<td>20</td>
<td>1,200</td>
<td>94.4563</td>
</tr>
<tr>
<td>2</td>
<td>7e-5</td>
<td>7e-5</td>
<td>20</td>
<td>1e+6</td>
<td>100</td>
<td>1,500</td>
<td>78.3396</td>
</tr>
</tbody>
</table>

Table 3.8.1: Cost function weights, number of shooting points, and thruster usage for the two simulations presented.

An example of what can be achieved by increasing the prediction horizon and tuning the controller less aggressively is shown in the figures below. The orbital trajectories are shown in figure 3.8.1, while thruster usage is shown in 3.8.2. The weights used on the cost function are shown in table 3.8.1. While the weights on the position and velocity states have been increased by a factor of 700 in the second case, the weights put on the derivative of control thrust have been increased by a factor of 200,000. Overall this results in a longer settling time and larger overshoot, but the fuel expenditures are reduced from 94.46 Ns to 78.33 Ns. Note that the attitude-related costs have also been modified, which does have an effect on the orbital related parameters.
Figure 3.8.1: Comparison between orbital trajectories of two controller configurations. The first configuration has 1200 shooting points, while the second configuration has 1500. The cost function weights are shown in table 3.8.1.
Figure 3.8.2: Comparison between thruster usage of two controller configurations. While the second configuration doesn’t quite reach a zero-output steady state, it uses significantly less fuel. It is clear that the y-thruster, oriented along the velocity vector, is responsible for most of the actuation in both cases. This is seen as a good behavior, as increasing or decreasing velocity is the main technique for switching orbits.

Though the maximum thruster force in these tests were limited to 1 mN, the second configuration never uses more than 421 µN. To see if the longer prediction horizon allows us to put an even stricter limit on the actuators, a new series of tests were launched. Three cases were tested, with respective limits of 200 uN, 70 uN, and 30 uN. The first two cases use 1,800 shooting points, while in the third case this number was pushed even further to 3,200. Compared to the previous case, the weights put on the derivative of control thrust were further increased in an effort to avoid saturation, but the relative weights were then kept the same throughout the entire test. The weights on the cost function are shown in table 3.8.2
Weights on the cost function

<table>
<thead>
<tr>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td>7e-5</td>
<td>7e-5</td>
<td>20</td>
<td>5e+6</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 3.8.2: Cost function weights when using weaker actuators.

The orbital trajectories are shown in figure 3.8.3, while thruster usage is shown in figure 3.8.4.

Figure 3.8.3: Orbital trajectories based on thruster strength. It is clear that with weaker thrusters, the chaser needs a longer time to catch up to the target.
Figure 3.8.4: Thruster usage using constraints of different magnitudes. The y-thruster is still the most used one, being the only thruster that reaches saturation in all three cases. We can also see that the 200 uN configuration is the least reliant on the x-thruster; these results indicate that when the y-thruster gets saturated the controller will delegate some additional work to the x-thruster instead.

Results regarding the actuation usage for the two tests are summarized in table 3.8.3. From an open-loop perspective, this data confirms that very long prediction horizons can effectively be utilized provided that the controller is tuned appropriately. The second test indicates that we can reduce the maximum thruster strength to just a fraction of the previously used value, and still expect an efficient maneuver. This is only true if the prediction horizon is increased even further, as the 70 uN case demonstrates. In fact, extending prediction horizon gives us even better performance than before from a fuel-optimum perspective. Attempts were made to apply the same number of shooting points to the stronger thrusters to give a fair comparison, but this caused the controller to become unstable. This is likely due to the fact that using stricter constraints heavily limits the amount of possible trajectories that the controller can take.
<table>
<thead>
<tr>
<th>Test 1</th>
<th>Max thrust</th>
<th>Pos&amp;vel cost</th>
<th>d/dt-Thrust cost</th>
<th>N</th>
<th>T_{TOT}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mN</td>
<td>7e-5</td>
<td>30</td>
<td></td>
<td>1200</td>
<td>94.4563</td>
</tr>
<tr>
<td>1 mN</td>
<td>1e-7</td>
<td>1e+6</td>
<td></td>
<td>1500</td>
<td>78.3396</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test 2</th>
<th>Max thrust</th>
<th>Pos&amp;vel cost</th>
<th>d/dt-Thrust cost</th>
<th>N</th>
<th>T_{TOT} (Ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 uN</td>
<td>1e-7</td>
<td>5e+6</td>
<td></td>
<td>1800</td>
<td>64.0781</td>
</tr>
<tr>
<td>70 uN</td>
<td>1e-7</td>
<td>5e+6</td>
<td></td>
<td>1800</td>
<td>71.6324</td>
</tr>
<tr>
<td>30 uN</td>
<td>1e-7</td>
<td>5e+6</td>
<td></td>
<td>3200</td>
<td>45.3886</td>
</tr>
</tbody>
</table>

Table 3.8.3: Summary of the results obtained in this chapter. More tuning and additional insights have allowed us to utilize a longer prediction horizon, weaker thrusters, and a much longer prediction horizon. This significantly reduces fuel consumption.

### 3.8.2 Short range

The latest tests have indicated that we can achieve much better fuel-efficiency by using weaker thrusters, as it significantly improves controller stability when longer prediction horizons are used. To make sure that these actuators are enough also for the later phases of the rendezvous maneuver, the final states from the 30 uN scenario were exported into a new short range simulation.

Just like in the previous short range simulations, the sampling time was reduced to 10 seconds, and the relative costs were altered to prioritize a correct position and velocity and be less conservative about fuel usage. The cost function weights used are shown in table 3.8.4.

<table>
<thead>
<tr>
<th>Weight on the cost function</th>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>20</td>
<td>500</td>
<td>5,000</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 3.8.4: Cost function weights when using weaker actuators in the short range phase.

The components of the chaser’s position vector in the LVLH-frame are shown in figure 3.8.5. The corresponding 3D-plot is shown in figure 3.8.6. Just like in the previous short range cases the chaser approaches the target using a spiral pattern, but the radius for each revolution decreases much slower than before. The spiral is also more tilted with regards to the target’s orbital plane, as indicated by the larger z-component. We can see how the z-component takes the longest to correct. The relative distance and velocity between the objects are shown in figure 3.8.7 while thruster usage is shown in figure 3.8.8. A total of 15.00 Ns is spent during the maneuver, of which 38.05% is consumed by the z-thruster. The x- and y-thrusters consume 30.03% and 31.92%, respectively. The long periods of saturation indicate that the fuel cost may be reduced by additional tuning to the controller parameters.
Figure 3.8.5: Components of the LVLH position vector during the short range phase, using 30µN thrusters.
Figure 3.8.6: Trajectory in the LVLH-frame during the short range phase, using 30μN thrusters.
Figure 3.8.7: Orbital trajectory during the close range phase. The initial conditions are the final states from the previous 30 uN simulation. The Chaser is successfully able to equalize its semi-major axis with that of the Target, and ends the simulation fluctuating between being 75 meters and just a few centimeters away from the target.
Figure 3.8.8: Thruster usage during the close range phase. All thrusters are used extensively during the orbit circularization, but the controller is very close to a zero-output steady state at the end of the simulation.

The long and short range phases are combined in the two figures below, using different colors to show the point of transition. The resulting orbital trajectory is shown in figure 3.8.9 while thruster usage is shown in figure 3.8.10.
Figure 3.8.9: Close and short range scenarios combined into one graph. It is not until the close range phase that the controller is able to circularize the orbit correctly, as seen in the third subplot.
Figure 3.8.10: Thruster usage in the combined scenario. The heavy usage during the close range phase, especially regarding the z-thruster, may be seen as a sign that the cost function is not very well tuned. It is possible that increased efficiency could be achieved by lowering the cost of the position states, but this would increase the steady state error. This could be solved by introducing a third phase (with higher costs) or by having a cost function that scales dynamically based on the current states. The behavior shown here also reflects that more time was devoted to tuning the long range mode of the controller, as that is where most of the fuel is spent.
4 Closed loop simulations

Up until this point, we have used identical dynamics for the controller and the simulated system, and no noise has been considered. Under those ideal circumstances we can expect that the controller’s prediction of the future is accurate, and that it will be able to reach the target as long as that prediction spans far enough into the future. However, when we construct a more realistic scenario things look very different. The controller does not have full knowledge of all the dynamics governing itself and its target, so the predictions will always be a little bit off from what really happens. Moreover, when we consider sensor inaccuracies the controller won’t even know exactly where it is located in space, and neither can it obtain full knowledge about the exact position of its target until it gets within camera range.

This chapter explains how a closed loop scenario was constructed in MATLAB, and analyzes the controller’s behaviour when it has to battle against perturbations from the Sun, Moon, and atmosphere of the Earth, as well as the limitations of its own hardware.

4.1 Types of noise

The various kinds of noises we can expect can largely be grouped as follows:

- **Inaccuracies in the modeling of the environment**: It is simply impossible to accurately model all of the forces affecting the system, since even distant planets and stars distort Earth’s gravitational field somewhat. Air resistance varies depending on time and location, and the output from the Sun is constantly shifting. Even if all the dynamics could be expressed mathematically, today’s computers would have difficulties processing all the information. These errors can be viewed as a small uncontrollable force that affects both chaser and target. Since the controller does not know about this force, it’s predictions of the future states will not be entirely accurate. As the unknown forces are relatively small compared to the known ones, we can expect that this will mostly affect the long-term predictions.

- **Sensor noise**: The software of the controller sees the world through a number of sensors, whose task is to sample different aspects of the environment and convert it into electrical signals that can be numerically interpreted. Since no sensors are perfect, the information supplied to the controller will not always be accurate. Up to now we have assumed that we have knowledge about the exact position and velocity of both chaser and target with respect to Earth, but due to errors in GPS measurements we will likely only have access to an approximate value. Likewise, errors from the attitude sensors could mean that the satellite isn’t oriented as expected, which would alter the direction of the force applied by the thrusters. The ability to obtain accurate measurements will depend on the types, quality, and quantity of the sensors used. As an example it is very difficult to estimate the distance to an object using a single camera, since the only relevant information obtained is the size of the object in the camera’s image. Better results can be obtained using stereoscopic vision, and if multiple chasers are able to share their measurements with each other there is a potential for obtaining very accurate measurements.

- **Thrusting errors**: A real thruster will always have physical limitations to some degree. Depending on the type of propulsion system and its throttling mechanisms, the precision with which the control forces are applied may vary. Some thrusters don’t allow throttling, and others have a minimum thrust that must be fired whenever the thruster is active. Many of these factors are very difficult to model, but some degree of thrusting uncertainty can be simulated by adding a small random component to the control signals. This should be looked at in more detail when simulating the docking phase of the operation, as that is where the need for precision maneuvers are the greatest.
4.2 Structure of the closed loop system

A block representation of the simulated closed loop system is shown in figure 4.2.1. The chaser and target are now simulated as two separate systems within the same loop; and contrary to the controller these systems consider all of the dynamics presented in chapter 2. This is all done in MATLAB, as the attempted Simulink implementations faced several issues.

No reference values are generated before the start of the simulation, they are now computed online based on the estimated states of the target and chaser at each sampling instant. The reference generator is responsible for estimating the future states of the target’s orbital parameters; this is done by integrating the sampled position and velocity over the span of the prediction horizon. This integration takes into account the $J_2$ effect, but none of the other perturbations. The estimated position and velocity states of the target are passed directly to the controller as reference values.

Reference values for the chaser’s attitude are then calculated in a separate function. They are based on these estimations as well as the controller’s internal prediction of the chaser’s future position and velocity states. The chaser is still instructed to point along its own velocity vector by using the method from chapter 3.3.

Sensor noise is simulated by adding a random component to the systems’ output states before passing them to the controller and reference generator.

![Figure 4.2.1: Block diagram of the simulated system.](image)

The solar and lunar positions are calculated at each sampling instant, and passed to the simulated systems as parameters. The same is done for the atmospheric density around the chaser and target as a function of their respective altitude. Their line-of-sight to the Sun is also calculated at every sample to determine if they are subject to solar radiation pressure.

4.3 The effect of modeling errors

Before adding any sensor noise, the effects of the modeling errors were investigated. A series of simulations, with an orbital phasing maneuver carried out at different altitudes, were run. All simulations used 2000 shooting points, and a sampling rate of 100 seconds. While using a longer prediction horizon should improve the results, the simulation times would be drastically longer. In these tests it was found that the modeling errors have negligible little impact on the performance of the controller, except for in very low orbits where the drag is high.

4.3.1 100 km altitude

It was shown in figure 2.1.1 that in our model, the atmospheric density decreases rapidly in the region around 100 km altitude. The density at 200 km altitude is just 0.1 % of this value. We can therefore expect that the drag will significantly alter the trajectories of both chaser and target, especially when
the chaser goes into its lower transfer orbit. The initial orbital parameters are summarized in table 4.3.1. The state space representation of those values are shown in table 4.3.2. The chaser’s initial attitude-related states are the same as the ones shown in table 3.5.3.

<table>
<thead>
<tr>
<th>Initial orbital parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-major Axis</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td><strong>Chaser</strong></td>
</tr>
<tr>
<td><strong>Target</strong></td>
</tr>
</tbody>
</table>

Table 4.3.1: Initial orbital parameters for the 100 km scenario.

<table>
<thead>
<tr>
<th>Initial orbital states</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-pos (Mm)</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td><strong>Chaser</strong></td>
</tr>
<tr>
<td><strong>Target</strong></td>
</tr>
</tbody>
</table>

Table 4.3.2: Initial orbital states of the chaser and target in the 100 km scenario.

Three simulations were made; in the first one neither chaser not target were affected by any perturbations, in the second case only the chaser was affected, and in the third case they were both affected. The relative distance, velocity, and semi-major axes are shown in figure 4.3.1.

The noiseless system behaves just like it would at a higher altitude, while applying noise to the chaser causes it to end up further from the target than it even began. The first half of the simulation still looks promising however, and it may be that with slightly more powerful thrusters the drag could be overcome even at this altitude. However, when noise was applied to both the chaser and target, the controller lost stability. No results were therefore obtained for this case.
Figure 4.3.1: Comparison of the orbital trajectories of a noise-less system compared to one where perturbations are applied to the chaser at 100 km altitude. The first 2 subplots show that even with heavy drag present, the controller gets very close to matching the position and speed of the target. It is not able to correct its semi-major axis after however, and ends up drifting away. This could mean that the drag makes the chaser unable to “circularize” its orbit to match that of the target. When perturbations affect the target as well, the controller is not able to find a solution.
Figure 4.3.2: Comparison of actuation usage in a noise-filled environment versus a noise-less one, at 100 km altitude. It can be clearly seen that the y-thruster saturates, and is unable to provide the force required to enter a higher orbit. In the noiseless scenario there is, as expected, not much difference from the higher orbits.

### 4.3.2 300 km altitude

The test was repeated, but this time at an altitude of 300 km. The atmosphere is several orders of magnitude thinner here than in the previous case, and while it is still thick enough to slowly de-orbit an object over time we can expect a much smaller impact from it. The corresponding initial orbital states of the chaser and target are shown in table 4.3.3.
<table>
<thead>
<tr>
<th></th>
<th>x-pos (Mm)</th>
<th>y-pos (Mm)</th>
<th>z-pos (Mm)</th>
<th>x-vel (km/s)</th>
<th>y-vel (km/s)</th>
<th>z-vel (km/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chaser</strong></td>
<td>6.576 546 174</td>
<td>1.004 262 570</td>
<td>0.579 811 265</td>
<td>1.341 577 203</td>
<td>6.589 121 723</td>
<td>3.804 231 200</td>
</tr>
<tr>
<td><strong>Target</strong></td>
<td>6.678 000 000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6.690 769 546</td>
<td>3.862 917 598</td>
</tr>
</tbody>
</table>

Table 4.3.3: Initial orbital states of the chaser and target in the 300 km scenario.

Figure 4.3.3: Comparison of the LVLH trajectories of a noise-less system compared to a more realistic one at 300 km altitude. The additional perturbations make virtually no difference.
Figure 4.3.4: Comparison of the LVLH position vector components of a noise-less system compared to a more realistic one at 300 km altitude. The additional perturbations make virtually no difference.

The LVLH trajectories of both cases are shown in figure 4.3.3. The components of the relative position vector in the LVLH-frame are shown in figure 4.3.4. The relative distance, velocity, and semi-major axis are shown in figure 4.3.5. Zooming in reveals that the position differences between the two cases lie in the area of just a few meters. While atmospheric density can vary a lot over time, something that is not considered in this test, we can still say with a fair degree of certainty that the environmental effects in LEO will play a very small role except at the lowest altitudes. The precise altitude under which operation is no longer feasible will however depend both on the CubeSat’s size, mass and available actuators, as well as external factors such as solar activity. The actuation usage for both cases are shown in figure 4.3.6. The thrusters are used an equal amount in both cases, with 23.6038 Ns being applied over the length of the simulation.
Figure 4.3.5: Comparison of the orbital trajectories of a noise-less system compared to a more realistic one at 300 km altitude.
Figure 4.3.6: Comparison of actuation usage in a noise-filled environment versus a noise-less one, at 300 km altitude. No difference in behavior can be seen.

4.3.3 600 km altitude

For the next scenario the altitude was increased to 600 km, while all other parameters were kept equal. The initial orbital states of the target and chaser are shown in table 4.3.4.

<table>
<thead>
<tr>
<th>Initial orbital states</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-pos (Mm)</td>
</tr>
<tr>
<td>Chaser</td>
</tr>
<tr>
<td>Target</td>
</tr>
</tbody>
</table>

Table 4.3.4: Initial orbital states of the chaser and target in the 600 km scenario.

The trajectory in the LVLH-frame is shown in figure 4.3.7 and the position vector is shown in figure...
4.3.8. The relative distance and velocity is shown in figure 4.3.9, and actuation usage is displayed in figure 4.3.10. It can be seen that the two scenarios evolve almost identically, and we have to go down to very insignificant digits to find any differences in the amount of thrust used. At this altitude, it is clear that the modeling errors play a very negligible effect.

Figure 4.3.7: Trajectory of the chaser in the LVLH-frame for the 600 km scenario.
Figure 4.3.8: Position vector of the chaser in the LVLH-frame.
Figure 4.3.9: Comparison of the orbital trajectories of a noise-less system compared to a more realistic one at 600 km altitude. The first 2 subplots show the relative position and velocity between Chaser and Target for the two models, and the behavior is almost identical. In the third subplot we can see the general shape of the maneuver; just like in earlier tests the chaser lowers its altitude to catch up to the target, but ends up overshooting and having to enter a higher orbit temporarily. After approximately 107 hours the chaser is extremely close to the target, but since its semi-major axis is almost 2 km shorter than that of the target it is not able to catch it at that point. At the end of the simulation, the Chaser’s semi-major axis is only around 50 meters shorter than that of the Target.
Figure 4.3.10: Comparison of actuation usage in a noise-filled environment versus a noise-less one, at 600 km altitude. No difference in behaviour can be seen.

4.3.4 Effects of not considering the J2-effect

Early on in the design phase, it was decided to include the dynamics of the J2-effect in the controller dynamics. Previous research [17] has shown that this inclusion can lead to fuel saving effects, but it would be interesting to back this up with further data. The J2-dynamics were therefore removed from the controller and reference generator, but still kept in the simulated objects, and the same scenario as in the last test was run. It was quickly found that the QP solver could not even find a convergent solution, and the simulation readily came to a halt. This confirms that the additional computational cost of taking the J2-effect into account is worthwhile, and indicates that it might be outright necessary in some cases. It’s possible that the controller could be made to run with some additional tuning, but this was not attempted due to the time investment necessary.

These findings suggest that modeling the Earth’s gravitational field even more accurately might be worthwhile, in order to further validate the controller’s ability to withstand unexpected gravitational forces. While we have seen that it is able to counter the gravitational pull from the Sun and Moon
without any visible effects to its performance, there are clearly some issues when larger errors are present in the controller model.

4.4 The effect of sensor noise

After establishing that the environmental noise plays a small effect in all but the lowest orbits, the effects of sensor noise were tested. This was done by adding a random component to some of the objects’ output states before they were passed to the controller and reference generator. The random component was generated with Matlab’s `randn` function to give the pattern of white noise, and then multiplied by a scalar to simulate various noise levels. The simulations were run with the same initial conditions as in the 300 km case (see table 4.3.3).

4.4.1 Orbital related parameters

As a first step, noise was added only to the 3 position states of the target. This simulates some uncertainty of the exact location of the target, but still assumes that we have exact knowledge of its velocity. Multiple tests were run with various noise levels, and it was found that the controller can handle uncertainties of up to a few meters. Higher noise levels cause the controller to fail after a part of the simulation, often at the point where it should start to increase its altitude again. The next step was to add some uncertainty regarding velocity as well. The same procedures as in the previous step were employed, and similar results were obtained. It appears that uncertainties greater than a few mm/s cause the controller to fail; just as in the previous case this often occurs at the lowest point in the transfer orbit. During the simulations it was noted that the controller’s internal prediction fluctuates much more than in a noise-less environment. The current implementation of the reference generator only takes the latest state measurement into account, so this is bound to happen with the present noise pattern. Small differences in velocity quickly builds up to a noticeably different trajectory over the length of the prediction horizon. In terms of performance however, no clear effects can be seen in the long range scenario. An example of the noise pattern applied is shown in figure 4.4.1.
Figure 4.4.1: Example of the white noise applied to the position and velocity states.

Noise of the same pattern and magnitude were then applied to the position and velocity states of the chaser itself. A comparison between three cases was made. The first case has no sensor noise, the second case has noise on the position and velocity states of the target, and the third case has noise on the position and velocity states of both the chaser and target. The orbital trajectory is shown in figure 4.4.2, while actuator utilization is shown in figure 4.4.3. There are no real differences in performance, and the fuel consumption in this test is actually slightly lower for the most noisy case. As the noise is random for each simulation, however, it is likely that the result would be different if the test was repeated.
Figure 4.4.2: The effect of applying sensor noise to the position and velocity states. The trajectories are almost identical at this noise level, but the controller loses stability if the noise levels are increased much further than this. **Case 1:** No sensor noise. **Case 2:** Noise on target states. **Case 3:** Noise on chaser and target states.
Figure 4.4.3: Actuation usage when sensor noise is applied to the position and velocity states. **Case 1:** No sensor noise. **Case 2:** Noise on target states. **Case 3:** Noise on chaser and target states.

### 4.4.2 Attitude related parameters

So far we have assumed perfect knowledge about the chaser’s orientation relative to the ECI-frame, but we can expect some uncertainty here as well with a magnitude that depends on the sensors used. The same type of white noise was applied individually to each of the 4 quaternion states, and the quaternion vector was then normalized to keep its magnitude to 1. An example of this shown in figure 4.4.4. The [1, 0, 0, 0] quaternion has been passed to the noise-adding algorithm 200 times, producing the pattern displayed.
Figure 4.4.4: Example of the quaternion measurements that are passed to the controller when the [1, 0, 0, 0] vector is fed through the noise generator.

Attitude data from the first test are shown in figures 4.4.5 and 4.4.6. The chaser is still able to reach its target, with only a slight increase to its fuel usage, but attitude-wise there are several issues.
Figure 4.4.5: Angle error when white noise is applied to the measured quaternion states. This is very different from the noise-less case, where angle errors were only present at the very start of the simulation.
Figure 4.4.6: Control torque and angular velocity when white noise is applied to the measured quaternion states. All three actuators are almost constantly saturated, and the chaser wiggles back and forth.

In previous tests the chaser has been able to reach its reference orientation relatively quickly, and has then easily remained there since there have been no unknown disturbances regarding these states. The heavy noise that is now applied to the measurements means that the controller constantly thinks its attitude is incorrect, and the high weights placed on the quaternion-states in the cost function causes it to apply all the torque that it can muster throughout the entire simulation. This gives the chaser a constantly high angular velocity, and makes pointing with any degree of accuracy impossible. While the applied noise may be much larger than one should expect with proper filtering techniques and such, it is still clear that lowering the weights placed on the quaternion states is necessary to avoid wasting power when heavy noise is present. This was done in the next test, by a factor of 1000, as well as a reduction in the maximum torque available to the controller to 0.05 $\mu$Nm with a maximum rate of change of 0.01 $\mu$Nm/s (10% of the previous values). Table 4.4.1 shows the updated weights on the cost function. These two cases are compared below, along with the previously shown case that only had noise in the position and velocity states. Figure 4.4.7 shows the amount of control
torque that is applied in each case, and figure 4.4.8 shows the thrust applied in each case. The control signals for torque in case 3 are still saturated, which suggest that the cost function needs further modification to accommodate these noise levels. There are some differences in thruster usage between all three cases, but the case with a modified cost function stands out the most. The fuel consumption in case 2 is 1.1% higher than in case 1, but case 3 in turn uses 9.6% more than case 2. This demonstrates that in terms of fuel efficiency, modifying the cost function often plays a larger effect on performance than sensor noise. The error quaternion for case 3 is shown in 4.4.9. Compared to the error quaternion for case 2, shown in figure 4.4.5, the angle error is smaller.

<table>
<thead>
<tr>
<th>Weights on the cost function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
</tr>
<tr>
<td>7e-5</td>
</tr>
</tbody>
</table>

Table 4.4.1: Updated cost function weights used in case 3 for the quaternion noise test.
Figure 4.4.7: Torque applied in the three tested scenarios. **Case 1:** Noise on position and velocity states. **Case 2:** Noise on position, velocity, and quaternion states. **Case 3:** Noise on position, velocity, and quaternion states. Modified cost function and 10x weaker actuators.
Figure 4.4.8: Thrust applied in the three tested scenarios. Though the orbital trajectories are almost identical, there are some visible differences regarding thruster usage. The third case stand out the most when seen in this way, and it also consumes the most fuel.
Figure 4.4.9: Angle error when white noise is applied to the measured quaternion states, using the modified cost function and weaker actuators. Compared to 4.4.5 the mean error is smaller.
Figure 4.4.10: Orbital trajectories for the three cases discussed. The motion is almost identical, but some small differences can be observed close to the end of the simulation when zooming in.

The next test was made to find out how the chaser is affected by an uncertainty regarding its angular velocity. White noise was added to the three angular velocity states, with an example shown in figure 4.4.11. The noise previously applied to the quaternions was removed for this test, but the noise applied to the position and velocity states was kept.
Figure 4.4.11: Example of the randomized noise pattern applied to the controller’s measurements of the chaser’s angular velocity.

The cost function was once again modified; the updated weights are shown in table 4.4.2. The control torque and angular velocities are shown in figure 4.4.12. Torque is applied through the entire simulation by all three actuators, but saturation only occurs for the first 120 hours. The error quaternion is shown in figure 4.4.13. The errors are significantly smaller than when the noise was applied to the measurement of the quaternion states.

<table>
<thead>
<tr>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td>7e-5</td>
<td>7e-5</td>
<td>0.2</td>
<td>5e+6</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4.4.2: Updated cost function weights used when applying noise on the measurement of the angular velocity states.
Figure 4.4.12: Torque applied and angular velocity when a disturbance is applied to the angular velocity measurements.
4.4.3 Increased prediction horizon in the presence of noise

The tests done in this chapter have been done with 2,000 shooting points, giving a total prediction horizon length of 55.56 hours. We saw in the open-loop simulations that the controller is capable of utilizing even longer prediction horizons, but this can make the simulations last for several more hours. In a noise-filled environment we can also expect that a longer prediction horizon is somewhat less useful, as the long-term predictions are not as accurate. To see if this effect is significant, the 300 km scenario was simulated two more times, but now with 3,000 shooting points (83.33 hours). The original configuration of \( N=2,000 \) and no sensor noise is referred to as case 1. In the second and third cases \( N \) is increased to 3,000, and case 3 has sensor noise included on the position, velocity, quaternion, and angular velocity states. The noise levels are the same as in chapter 4.6.

Numerical results are summarized in table 4.4.3. The 50% longer prediction horizon increased the average computational time by 83% (and the maximum time by 119%), but the fuel consumption was also reduced by 12.5%. Adding sensor noise did not cause any observable negative effects regarding
fuel efficiency, as even less actuation is used in the noise-filled case.

<table>
<thead>
<tr>
<th>Case</th>
<th>N</th>
<th>Pred. Horizon length</th>
<th>avg cpt</th>
<th>max cpt</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,000</td>
<td>55.56 h</td>
<td>278.07 ms</td>
<td>398.41 ms</td>
<td>23.60 Ns</td>
</tr>
<tr>
<td>2</td>
<td>3,000</td>
<td>83.33 h</td>
<td>508.46 ms</td>
<td>871.62 ms</td>
<td>20.65 Ns</td>
</tr>
<tr>
<td>3</td>
<td>3,000</td>
<td>83.33 h</td>
<td>508.24 ms</td>
<td>804.35 ms</td>
<td>20.37 Ns</td>
</tr>
</tbody>
</table>

Table 4.4.3: Numerical results for the closed-loop prediction horizon test.

The distance, velocity, and semi-major axis relative to the target is shown in figure 4.4.14 for all three cases. Case 2 and 3 have almost identical trajectories, experiencing less overshoot than case 1. The trajectories start to visibly diverge from the first case after the 60 hour-mark, as the increased prediction leads to the increase in altitude happening a few hours earlier than in the N=2000 case. This can also be seen by looking at the y-thruster usage in figure 4.4.15. The trajectories follow the same general pattern, but the switch from full negative to full positive happens a few hours earlier. The switch happens around the 60 hour-mark, correlating with the divergence of the orbit trajectories. The orbit trajectories are shown in the LVLH coordinate system in figure 4.4.16. The trajectories of case 2 and 3 follow each other very closely, despite some notable differences in the actuation applied by the x-thrusters. The largest difference with respect to case 1 is seen in the y-coordinate, where case 1 experiences an overshoot of 190 meters compared to 75 meters in the other two cases. All three cases experience oscillations of around 1 km in the z-coordinate, which is the out of plane component. This is probably a result of the low weights put on the position states in the cost function, as the z-thruster is never used to produce more than 0.5 µN at any instant in any of the three cases.
Figure 4.4.14: Distance and relative velocity to the target. **Case 1:** N=2,000, no sensor noise. **Case 2:** N=3,000, no sensor noise. **Case 3:** N=3,000, applied sensor noise.
Figure 4.4.15: Thruster usage. **Case 1:** $N=2,000$, no sensor noise. **Case 2:** $N=3,000$, no sensor noise. **Case 3:** $N=3,000$, applied sensor noise.
Figure 4.4.16: Motion in the LVLH-frame. **Case 1**: N=2,000, no sensor noise. **Case 2**: N=3,000, no sensor noise. **Case 3**: N=3,000, applied sensor noise.

### 4.5 Summary of long range results

The simulations so far indicate that noise plays a very small role to the chaser’s ability to perform an orbit phasing maneuver under most circumstances. Even if the attitude actuators are constantly saturated due to sensor noise and inefficient tuning, there is only a marginal difference in the amount of fuel spent to perform the operation. Data from the tests conducted are summarized in table 4.5.1 and table 4.5.2. It can be noted that the largest change in fuel expenditure occurred when the cost function was modified. While reducing the cost for attitude errors seem to decrease the pointing errors (due to less aggressive controller behavior), the amount of fuel spent increased by roughly 10%. This is probably due to the fact that reducing the cost of one parameter in the cost function in practice increases the relative cost of all the other parameters. This is another example of things that must be kept in mind when manually tuning the cost function.
The effect of modeling noise at different altitudes

<table>
<thead>
<tr>
<th>Altitude</th>
<th>Model noise</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 km</td>
<td>Yes</td>
<td>Simulation failed</td>
</tr>
<tr>
<td>100 km</td>
<td>Chaser only</td>
<td>49.1375 Ns*</td>
</tr>
<tr>
<td>100 km</td>
<td>No</td>
<td>23.1169 Ns</td>
</tr>
<tr>
<td>300 km</td>
<td>Yes</td>
<td>23.6038 Ns</td>
</tr>
<tr>
<td>300 km</td>
<td>No</td>
<td>23.6038 Ns</td>
</tr>
<tr>
<td>300 km</td>
<td>Yes</td>
<td>20.6502 Ns**</td>
</tr>
<tr>
<td>600 km</td>
<td>Yes</td>
<td>25.0859 Ns</td>
</tr>
<tr>
<td>600 km</td>
<td>No</td>
<td>25.0859 Ns</td>
</tr>
</tbody>
</table>

Table 4.5.1: The effect of modeling noise at different altitudes. *Did not reach target. **50% longer prediction horizon.

The effect of sensor noise

<table>
<thead>
<tr>
<th>Pos- and Vel- noise</th>
<th>Quaternion noise</th>
<th>ω noise</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target only</td>
<td>No</td>
<td>No</td>
<td>23.6220 Ns</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>23.6000 Ns</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>23.8819 Ns</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>26.1629 Ns*</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>23.8643 Ns**</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>20.3736 Ns***</td>
</tr>
</tbody>
</table>

Table 4.5.2: Summary of how white sensor noise applied to the various states affects fuel expenditure. *Uses cost function 4.4.1. **Uses cost function 4.4.2. ***50% longer prediction horizon, and quaternion noise amplitude 15x lower than figure 4.4.4.

4.6 Noise in the short range scenario

4.6.1 Modeling noise

The final states of the target and chaser, from the last simulation shown here, were now exported into the short range scenario. The initial orbital states are shown in table 4.6.1. The initial attitude states of the chaser are shown in table 4.6.2.

Initial orbital states

<table>
<thead>
<tr>
<th></th>
<th>x-pos (Mm)</th>
<th>y-pos (Mm)</th>
<th>z-pos (Mm)</th>
<th>x-vel (km/s)</th>
<th>y-vel (km/s)</th>
<th>z-vel (km/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chaser</strong></td>
<td>5.576 061 482</td>
<td>1.592 966 628</td>
<td>3.278 084 560</td>
<td>1.759 226 587</td>
<td>7.510 326 402</td>
<td>0.664 226 927</td>
</tr>
<tr>
<td><strong>Target</strong></td>
<td>5.578 188 092</td>
<td>1.586 674 875</td>
<td>3.279 058 490</td>
<td>1.754 605 971</td>
<td>7.510 856 508</td>
<td>0.660 240 183</td>
</tr>
</tbody>
</table>

Table 4.6.1: Initial orbital states of the chaser and target in the short range scenario.

Chaser initial attitude states

<table>
<thead>
<tr>
<th></th>
<th>q4</th>
<th>q1</th>
<th>q2</th>
<th>q3</th>
<th>ωx (rad/s)</th>
<th>ωy (rad/s)</th>
<th>ωz (rad/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chaser</strong></td>
<td>-0.123 613</td>
<td>-0.257 346</td>
<td>0.012 999</td>
<td>0.958 292</td>
<td>3.495 134e-06</td>
<td>3.338 912e-06</td>
<td>0.001 161 869</td>
</tr>
</tbody>
</table>

Table 4.6.2: Initial attitude states of the chaser for short range simulation.

In this first test model noise is applied to both the chaser and the target, but sensor noise is applied
to neither. As in all the previous short range simulations, the controller runs at a sampling rate of 10
seconds in this mode. Two cases of N were tested; the first case used 2,000 shooting points while
the second case had 3,000. The weights on the cost function for this test can be found in table
3.8.4. The LVLH position vectors for both cases are shown in figure 4.6.1, while actuation usage
is shown in figure 4.6.2. Average and maximum computation times are shown in table 4.6.3. The
chaser isn’t able to correct the small inclination error within the simulation time of 300,000 seconds;
this can be seen by looking at the oscillating z-component of the relative position vector. The error
is, however, still decreasing by the end of the simulation. Increasing the prediction horizon does lead
to a slightly different trajectory, and the amplitude of the final z-oscillations was reduced from 300
meters in case 1 to 200 meters in case 2. The 50% longer prediction horizon increased the average
computation time by 59.6%, but the maximum computation time was only increased by 18.4%. Case
2 also consumes 10.6% more fuel than case 1. Especially the x-thruster behaves differently towards
the end of the simulation, where the trajectories in both cases pan out to different values on opposite
sides of 0.

Figure 4.6.1: Trajectory in the LVLH-frame. **Case 1:** N=2,000 **Case 2:** N=3,000
Table 4.6.3: Computation times and thruster usage for two cases of prediction horizon length in the short range scenario.

<table>
<thead>
<tr>
<th>Case</th>
<th>N</th>
<th>Pred. Horizon length</th>
<th>avg cpt</th>
<th>max cpt</th>
<th>Total thrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,000</td>
<td>5.56 h</td>
<td>314.75 ms</td>
<td>953.99 ms</td>
<td>11.79 Ns</td>
</tr>
<tr>
<td>2</td>
<td>3,000</td>
<td>8.33 h</td>
<td>501.18 ms</td>
<td>1130.08 ms</td>
<td>13.04 Ns</td>
</tr>
</tbody>
</table>

Figure 4.6.2: Actuation usage in the closed-loop short range scenario. **Case 1:** N=2,000 **Case 2:** N=3,000

4.6.2 Sensor noise

The effect of sensor noise in the short range scenario was now tested. For this test, the cost function was once again updated with the aim of reducing the settling time. The updated cost function is shown in table 4.6.4. The number of shooting points were set to 3,000, giving a prediction horizon
length of 30,000 seconds or 8.33 hours. Using a shorter prediction horizon than that caused stability issues with this cost function. Simulation length was set to 300,000 seconds as in the previous test.

Two cases were tested. In the first case, the target and chaser are affected by all the background forces in the model, but there is no sensor noise. In the second case, there are both background forces and sensor noise. The sensor noise is placed on the position, velocity, and angular velocity states with same magnitude as before. Some noise is also placed on the quaternion states, but the magnitude used is 15 times weaker than in figure 4.4.4.

<table>
<thead>
<tr>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td>30,000</td>
<td>5,000</td>
<td>500</td>
<td>5e+6</td>
<td>5e+6</td>
</tr>
</tbody>
</table>

Table 4.6.4: Cost function weights in the closed-loop close range scenario featuring sensor noise.

A comparison of both trajectories in the LVLH-frame is shown in figure 4.6.3, and the LVLH position vector is shown in figure 4.6.4. The approach follows the same general pattern as in the previous test, with the chaser moving in a slowly decreasing spiral around the target, but this configuration struggles even more with eliminating the error in the z-component. The oscillation amplitude at the end of the simulation is in both cases around 550 meters. Relative distance and velocity for both cases is shown in figure 4.6.5, and only small differences can be seen. Thruster usage is shown in figure 5.2.1. 15.9039 Ns of actuation is used in the first case, but when sensor noise is included this value unexpectedly drops to 15.1367 Ns. This is still more than what was used with the previous cost function, despite the larger position errors.
Figure 4.6.3: Relative position and velocity in the close range phase, using cost function 4.6.4. **Case 1:** No sensor noise. **Case 2:** With sensor noise.
Figure 4.6.4: Relative position and velocity in the close range phase, using cost function 4.6.4. **Case 1:** No sensor noise. **Case 2:** With sensor noise.
Figure 4.6.5: Relative position and velocity in the close range phase, using cost function 4.6.4. **Case 1**: No sensor noise. **Case 2**: With sensor noise.
Figure 4.6.6: Thruster usage in the close range phase, using cost function 4.6.4. Case 1: No sensor noise. Case 2: With sensor noise. The behavior is very similar at first, but around the 40h mark the trajectories of the x-thruster diverge.

4.6.3 Latest iteration of the short range configuration

This section shows simulation results for the latest iteration of the short range mode configuration. The weights on the cost function have been updated in an attempt to reduce the effects of quaternion sensor noise, and the simulation time has been increased to see just how close the chaser is able to get to the target in the presence of sensor noise. These results were obtained on November 25th, and the simulation took 22 hours to complete. The updated weights are shown in table 4.6.5.
Weights on the cost function

<table>
<thead>
<tr>
<th>Position</th>
<th>Velocity</th>
<th>Quaternion</th>
<th>d/dt-Thrust</th>
<th>d/dt-Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>20</td>
<td>50</td>
<td>1,000</td>
<td>5,000</td>
</tr>
</tbody>
</table>

Table 4.6.5: Cost function weights used in the latest iteration of the short range configuration (November 25).

The components of the chaser’s position vector in the target-centered LVLH-frame are shown in figure 4.6.7. This configuration was able to correct the inclination error much faster than the previously shown cases, with an oscillation magnitude of just 64 meters around the 80-hour mark. Later on the oscillations are reduced to mere centimeters.

Figure 4.6.7: Relative position in the LVLH-frame. Short range scenario with sensor noise and cost function 4.6.5

Plots of the distance and relative velocity for the later parts of the simulation are shown in figure
4.6.8. Due to the noise added on the measurements of the position and velocity states, the real distance fluctuates between 0 and 35 meters even in steady-state. The relative velocity between the two objects reached up to 3.5 cm/s in the shown time span. These results indicate that we can get very close to the target even with inaccurate GPS-measurements, but also show the need for a more accurate estimation of the relative motion once we get to this point.

Figure 4.6.8: Distance and relative velocity between the chaser and the target. Short range scenario with sensor noise and cost function 4.6.5

Actuation usage is shown in figure 4.6.9. The high weights on the position states, coupled with the spiky noise pattern, means that the thrusters never reach a zero-output steady-state. The z-thruster, which is normal to the plane of motion, is only used sparingly in the later half of the simulation, never exceeding 1 µN after the 120-hour mark. A total of 16.83 Ns are supplied from the thrusters over the entire simulation.
Figure 4.6.9: Actuation usage. Short range scenario with sensor noise and cost function 4.6.5

These results indicate that even in the short range scenario, the controller has the ability to overcome sensor noise and get within a few meters of the target. This is closer than we can realistically go without an on-board camera or other way to accurately measure and estimate the motion between the objects, as the need for precision becomes much greater when the actual docking maneuver is to be executed.
5 Conclusion

5.1 Discussion of the results

This thesis has focused on simulating long range phasing maneuvers in LEO, using a realistic model of the environment and a small CubeSat-class spacecraft. The early work focused on using 1 mN thrusters, which is well within the range of the cold-gas propulsion systems currently being developed [12]. The tests in chapter 3.5.2 showed that there is a minimum requirement on the length of the prediction horizon for being able to rendezvous with the target, and that further increases in general leads to a faster settling time and lower fuel consumption. Above a certain point however (N=500 in that case), the performance may actually get worse, and the controller completely loses stability if the prediction horizon is increased even further. Later tests in chapter 3.5.3 then showed that the length of the prediction horizon in itself does not tell the whole story, as we got significantly different results when rebalancing the sampling rate and number of shooting points. This is believed to be linked to the fact that the controller is only allowed one iteration at each sampling instant, and is therefore not guaranteed to find an optimal solution. A less complicated problem could therefore mean that the controller gets closer to an optimal solution.

It was later realized that the controller could maintain stability for much longer prediction horizons when the cost function was tuned differently, as shown in chapter 3.8, and the number of shooting points could be increased even further when tighter constraints were put on the actuators. This is probably due to the amount of possible trajectories being much lower when stronger weights and constraints are placed on the actuators, which helps the controller in finding an optimal solution. It was shown that with the new configuration, prediction horizons as long as 90 hours still provide benefits over using shorter ones. This holds true even in a noise-filled environment, as shown in chapter 4.4.3.

The later part of the work was then focused on using thrusters with a strength of 30 \( \mu \)N, just 3% of the original value, which is within the range of today’s electrical propulsion systems [12]. While the time taken to reach the target was much longer using this configuration, the total amount of actuation used was also significantly lower. As seen in table 3.8.3 however, reducing the maximum thrust values without increasing the prediction horizon or modifying the cost function is likely to increase fuel consumption, as the controller spends more time in saturation.

One of the main drawbacks of using an electrical propulsion system is, in most contexts, the low thrust values possible because of the high power consumption. For this MATMPC-based controller however, the low thrust values lead to increased stability which in turn enables a longer prediction horizon. The longer prediction horizons then allow us to tune the weights so as to promote fuel efficiency over a shorter mission time.

While electrical propulsion systems don’t have the technological maturity level that cold-gas systems do, the higher specific impulse that can be obtained allows the spacecraft to perform a lot more maneuvering within its lifetime. This allows it to reach targets that are further away, and also means that less fuel will expended when re-orbiting the debris itself. If the mission is carried out in such a way that the chaser is able to survive, it might even be possible to perform multiple consecutive rendezvous maneuvers with different debris objects.

The exact amount of fuel that corresponds to the actuation used in the various scenarios will depend on the specific propulsion system used, as there are large differences even within the various groups. As an example, the SiEPS previously mentioned has a specific impulse of 1,150 s [10]. The relation between thrust, specific impulse, and mass consumption is described by the equation

\[
F_{thrust} = g_0 I_{sp} \dot{m}
\]
Where $\dot{m}$ is the mass flow rate in kg/s and $g_0$ is the standard gravity (9.81 m/s$^2$). The total mass change $\Delta m$ for a given maneuver can therefore be written as

$$\Delta m = \frac{T_{TOT}}{(g_0I_{sp})}$$

(75)

Where $T_{TOT}$ is the total thrust action in Ns provided over the maneuver. The roughly 25 Ns consumed in the closed-loop long range phase thus corresponds to around 2.2 grams of propellant. Using the VACCO cold gas thruster instead, with a specific impulse of 46 s, the same amount of actuation corresponds 55.4 grams of propellant. The cold gas configurations presented in this thesis did however use much more actuation than this, due to the different tuning and prediction horizons, giving the electrical configuration an even greater advantage. The latest iteration of the short range configuration uses approximately 1.5 grams of propellant, but a significant part of this is spent on inclination correction. The sensor noise also makes the controller use actuation when in steady-state, in a fruitless attempt to correct the perceived errors.

The performance of the controller was shown to be virtually unaffected by the perturbing forces that can be expected in the LEO-environment, except for when heavy drag is present. Simulation data points to altitudes a bit above 100 km to be safe, but since atmospheric density varies heavily over time these values should only be considered as rough estimates. In the higher portions of LEO, which are currently the main orbits considered for CubeSat missions, the controller can be expected to perform almost as well as in a noise-less scenario.

As discussed in the literature review, pointing errors of a few degrees can be expected if inexpensive sensors are used for attitude determination. The test results indicate that this should only play a minor effect on the performance of the controller during the orbit phasing maneuver, and once the satellite is closer to its target the on-board cameras should provide much more accurate measurements of their relative motion. Further optimization of the cost function may reduce the noise sensitivity of the attitude-related states further.

The controller has also shown that it is able to handle white noise applied to its position and velocity measurements, with uncertainties of up to several meters and mm/s, respectively. Since the errors in GPS measurements typically lie within a few meters, these results seem acceptable. It must be kept in mind that this controller runs at a very slow sampling rate, and if the sensors are able to take continuous measurements between the sampling instants the noise pattern may become a lot smoother than the one presented in 4.4.1.

In the early tests a vast majority of the fuel consumption occurred in the long range phase; this can be seen in table 3.6.4 for example. Later tests showed a much higher relative consumption in the short range phase, even though the majority is still spent in the long range phase. One explanation for this is the inclination error that occurs in the long range phase; an example of this can be seen in figure 4.4.16. This is caused by the low relative weights put on the position and velocity states, which makes this small out-of-plane component of the position error have an inconsequential effect on the cost function. This can be solved by putting a high weight on the position errors of the LVLH-frame z-axis, but since this is a non-linear combination of controller states (as it operates in the ECI-frame) the resulting cost function will be more complex.

It should be noted that optimizing the weights for a longer prediction horizon was a time consuming process, due to the long simulation times required. With an average computation time of 0.5 seconds per 100 second-step the total computation time over a 1,000,000 second simulation (277.7 hours) is 5,000 seconds. The simulation time is then further increased by calculations for reference generation and the perturbation parameters, and the simulation of the chaser and target. This makes it difficult to compare many cases in multiple scenarios, and it is therefore believed that the fuel efficiency can still be improved significantly.
The long computation times may also be the limiting factor when implementing this controller in an embedded system. It is difficult to estimate how the computation times obtained here correlates to a real life implementation; on the one hand the embedded system should be much more optimized than a Windows PC running MATLAB, but on the other hand a CubeSat may only have access to a processor running on a few hundred MHz, as opposed to the Core i5-4670K @ 3.40 GHz with 16 GB DDR3 1600 MHz RAM used in this work.

5.2 Comparison with an ideal Hohmann maneuver

As noted in the introduction, a spacecraft using conventional thruster technology could perform the orbit phasing maneuver using a two-impulse Hohmann transfer. It may therefore be useful to compare the obtained results with this technique to get some measure of how fuel-efficient the controller really is. Assuming a circular orbit with an altitude of 300 km \( (a_1 = 6,678) \), the orbital period can be calculated as

\[
T_1 = 2\pi \sqrt{\frac{a_1^3}{\mu}} = 5,431 \quad (76)
\]

The orbital period of the phasing orbit depends on the phase difference between the starting and target orbits, as well as how many orbital periods the maneuver is allowed to last for. With a phase difference of 10 degrees, a total of 150.86 seconds (1/36 of an orbital period) must be gained. The required orbital time of the phasing orbit can be calculated as

\[
T_2 = T_1 - \frac{150.86}{n} \quad (77)
\]

Where \( n \) is the number of revolutions. From this the required semi-major axis of the phasing orbit can be calculated as

\[
a_2 = \left( \frac{\sqrt{\mu T_2}}{2\pi} \right)^{2/3} \quad (78)
\]

The angular momentum of each orbit can be found through the formula

\[
h = \sqrt{2\mu \frac{r_ar_p}{r_a + r_p}} \quad (79)
\]

Where \( r_a \) and \( r_p \) are the apogee and perigee for the respective orbit. For the original circular orbit, they are both equal to \( a \). For the transfer orbit, assuming it is lower than the original orbit, they can be calculated as

\[
r_a = a_1 \quad (80)
\]

\[
r_p = a_2 - r_a \quad (81)
\]
The $\Delta V$ required to transfer into the phasing orbit is then

$$\Delta V = \frac{h_2}{a_1} - \frac{h_1}{a_1}$$

(82)

To get back into the original orbit, the same momentum must then be applied in the opposite direction. To convert the $\Delta V$ into Ns, the value should be multiplied by the mass of the spacecraft, in our case 4 kg. The plot below shows the total actuation needed for a Hohmann maneuver as a function of the time allowed for the maneuver. Using as much actuation as required in the long range phase of our closed loop simulations, roughly 23.6 Ns, a Hohmann transfer could be completed in just over 36 hours. Alternatively, using the roughly 250 hours needed for our long range phase, the Hohmann technique would require less than 3.5 Ns. While these figures represent an ideal scenario, where all thrust is instantaneous and no perturbations of any kind are present, it is still clear that the heavy restrictions put on the actuators do come with a significant cost in either fuel efficiency or time taken to reach the target.
Figure 5.2.1: The actuation required for an Ideal Hohmann transfer, for a phase difference of 10 degrees in a 600 km altitude circular orbit.

5.3 Future improvements

5.3.1 Accurate modeling of the actuators

This project has assumed that there are six thrusters mounted on the chaser, and that they are aligned perfectly with the axes of the chaser’s body-frame. Tests have been performed on this setup, using various levels of thruster strength, to provide some insight into how the capabilities of the actuators influence the overall performance. To get a full working design, there are several more things that must be considered:

- The amount of power that can be used by the thrusters will depend on the solar panel and battery configuration of the CubeSat. It might not be feasible to fire several thrusters at once at maximum capacity. To conserve power during times in the Earth’s shadow, it might be possible to heavily restrict thruster usage if the chaser already has a “good” trajectory.

- In reality, the thrusters are often tilted to make room for other equipment and instruments.
This means that they provide torque to the spacecraft when fired. Using enough of them, it is possible to perform both attitude and orbital control at once. For example, VACCO’s Reaction Control Propulsion Module, designed for occupying the middle segment of a 3U CubeSat, contains eight 25 mN cold gas thrusters. [19] This should not be difficult to implement in MATMPC, but may require extensive testing to maintain fuel efficiency.

- When a spacecraft fires its thrusters, fuel is ejected to create a net force in the desired direction. While the amount of fuel required heavily depends on the type and model of the thruster, it does inevitably lead to a decrease in mass of the spacecraft. The results indicate that this is mainly an issue when using cold gas thrusters, as the mass consumption is just a few grams when performing the phasing maneuvers using electrical thrusters.
- As noted in [1], any real mission will experience some uncertainty with regards to the force provided by the thrusters at any point in time. This is both due the hardware of the thrusters themselves, as well as the result of any error in the estimation of the orientation of the satellite.

### 5.3.2 Adding a low-level controller

[15] describes how an NMPC controller designed in MATMPC can be used for generating a reference trajectory that another, simpler, controller must then follow. With the addition of interpolation between the controllers they were able to run at different sampling rates, with the low-level local controller having a rate five times as high as the NMPC. This method reduces the issues of long computational times, and was also shown to increase the noise-rejection capabilities.

It is likely that a similar strategy could be used for this application to solve the noted issues of long computation times. It is also possible that this would enable us to generate an even more optimal trajectory, as it may sometimes be affordable to use an even longer prediction horizon and let the NMPC do a few iterations of its calculations before passing the updated values to the local controller.

### 5.3.3 Docking with the target

This thesis has investigated the ability of a CubeSat to perform orbital phasing maneuvers and then minimize the distance to a set target. However, we have not looked at the final phases of the rendezvous maneuver where the actual docking occurs. To approach the target in a safe way, we must take into account the rotation of both chaser and target. Even if the relative velocity of the two objects is close to zero, a significant exchange of force will occur if they touch each other without first synchronizing their attitude motion.

For this to be possible, we must be able to identify the attitude related parameters of the target using the sensors mounted on the chaser. Once the spin axis of the target has been identified, the chaser should position itself so that it can approach the target along the vector aligned with this spin axis. If the chaser is then able to match the targets angular velocity along said axis, the target will appear to be non-rotating from the perspective of the chaser-mounted camera. This means that we should be able to match the attitude related parameters very accurately, and only need to make sure that the translational approach can be performed smoothly.

To make the chaser follow the approach vector, we must generate position reference values based on both the position and angular velocity of the target. The work done up until now has directly used the current and predicted position of the target as the reference value, but now the reference position must be placed at a slowly decreasing distance from the target along its rotational axis. This will require the chaser to remain in a somewhat unstable orbit, since there is no natural orbit fulfilling these requirements. It is therefore expected that the thrusters will be used constantly during this operation, and it must be ascertained that they can provide enough force for the chaser to remain in this unstable orbit until the docking is completed.
If the target in question has a tumbling motion, something that is to be expected for an uncontrollable object, the mathematics involved for synchronizing the rotational motion between the objects are rather complex. This is a well-studied topic; one algorithm is shown in [18] for example.

As previously discussed, the docking phase of the maneuver can already be performed using existing control strategies, and the need for a non-linear control strategy and a long prediction horizon decreases as the objects get closer to each other. However, the indicated noise-rejection capabilities of the controller presented here, and the potential for creating a seam-less transition between the modes of operation, means that it might be interesting to investigate this phase with MATMPC as well.

5.4 Issues encountered

One of the greatest issues is the computational time required for the simulations. It was noted in chapter 3.7 that the controller became much faster after switching to a different solver, but in the later phases of the development every simulation still took several hours to compute. The main reason for this was the reduction in thruster strength from 1 mN to just 30 µN; this increases the settling time of the system which in turn requires a longer prediction horizon to optimize behavior.

This meant that every test had to be planned carefully, to extract as useful data as possible, since simulating a few different cases for a given scenario could take an entire day. The long simulation times also make it difficult to fine-tune the controller parameters, as results from multiple test scenarios are necessary to completely evaluate how any change affects the controller’s behavior. It was found that plotting some of the states and the controller’s internal predictions while the simulations were running could often be useful to get an early indication of controller performance, and simulations that did not show promise could therefore be aborted at an early stage. Plotting during the simulation does however lead to slightly increased simulation times.

Properly tuning the controller parameters was a time consuming process in itself. It is apparent that different weights on the cost function are required for each different phase of the rendezvous maneuver, as the values that provide good fuel efficiency also give rise to a larger steady state error. It is likely that some kind of scaling cost function, that makes the controller more liberal with its thrusting actions as it gets closer, would result in a more efficient behavior. To avoid having to manually find parameters that work in all situations, advanced techniques such as Derivative Free Optimization could be employed. The sampling rate and prediction horizon requirements also depend on the relative states of the chaser and target, and some dynamic scaling of these parameters is likely necessary for optimal behavior. This should be possible to implement in MATMPC, but will require some additional algorithms to transfer the controller’s shooting points between different sampling rates.

As pointed out several times, the design of the controller became very much an iterative process, as all the different parameters affected each other in many unforeseen ways. The results in 3.5.18 seemed to indicate that a prediction horizon longer than 60,000-80,000 seconds did not provide much benefit. But when the controller was tuned less aggressively, and took longer time to reach the target, gains were seen at lengths far beyond this point. It should again be emphasized that the longest prediction horizons used here, of over 300,000 seconds, only worked when the thrusters were constrained to the micro-Newton range. More available thrust means that many more trajectories through state space become possible, so it makes sense that strict constraints make the controller calculations easier. Controller stability is also increased when greater weights are put on the actuation cost, as this too limits large changes in thrust values that drastically alter the predicted trajectories.
References


A Appendix A: Matlab Codes

A.1 Main file (Closed-loop Simulation)

```matlab
% MATMPC: orbits - ORBITal and attitude Simulator & controller
% Main file
% Orbital parameters are defined here
% Contains a closed-loop system
% Chaser and Target are both simulated
% Reference values are generated on-line
% Sensor noise is added to system outputs
%
clear mex; close all;clc;
clear
%% MATMPC Configuration
addpath([pwd,'/nmmpc']);
addpath([pwd,'/model_src']);
addpath([pwd,'/mex_core']);
addpath(genpath([pwd,'/data']));
cd data;
if exist('settings','file')==2
    load settings
cd ..
else
    cd ..
    error('No setting data is detected!');
end
Tf = 1000000; % simulation time
settings.Tf = Tf;
Ts = settings.Ts; % Sampling time
Ts_st = settings.Ts_st; % Shooting interval
s = settings.s; % number of integration steps per interval
nx = settings(nx); % No. of states
nu = settings.nu; % No. of controls
ny = settings.ny; % No. of outputs (references)
nyN = settings.nyN; % No. of outputs at terminal stage
np = settings.np; % No. of parameters (on-line data)
cc = settings.nc; % No. of constraints
ncN = settings.ncN; % No. of constraints at terminal stage
nbx = settings.nbx;

%% Constraints
settings.thrusterForce = .03; % maximum thruster force (mN)
settings.torqueForce = .05; % maximum torque (uNm)
settings.dThrustMax = .1; % maximum rate of change thruster force (mN/s)
settings.dTorqueMax = 1e-2; % maximum rate of change torque (uNm/s)
settings.maxAngVel = 5e-3; % maximum angular velocity (rad/s)

%% Orbit parameters
% Chaser initial conditions
settings.chaser.semiMajorAxis = 6498.0;
settings.chaser.eccentricity = 0.0;
settings.chaser.inclination = pi/6;
settings.chaser.trueAnomaly = -pi/18;
settings.chaser.argumentofPerigee = 0;
settings.chaser.RAAN = 0;

% Target initial conditions
settings.target.semiMajorAxis = 6498.0;
```

settings.target.eccentricity = 0;
settings.target.inclination = pi/6;
settings.target.trueAnomaly = 0;
settings.target.argumentofPerigee = 0;
settings.target.RAAN = 0;

%% solver configurations
N = 2000; % No. of shooting points
settings.N = N;
N2 = 8;
settings.N2 = N2; % No. of horizon length after partial condensing (N2=1
means full condensing)
opt.integrator='ERK4'; % 'ERK4', 'IRK3', 'ERK4-CASADI'
opt.hessian='gauss_newton'; % 'gauss_newton'
opt.condensing='no'; % 'default_full', 'no', 'blasfeo_full', 'partial_condensing'
opt.qpsolver='hpipm_sparse';
opt.hotstart='no'; % 'yes', 'no' (only for qpoases)
opt.shifting='yes'; % 'yes', 'no' % if objective function is linear least square
opt.ref_type=2; % 0-time invariant, 1-time varying (no preview), 2-time varying (preview)

% Start date
year = 2020;
month = 1;
day = 1;
hour = 12;
minute = 0;
second = 0;
JulianDay = calculateJulianDay(year,month,day,hour,minute,second);

%% Initialize Data
[input, targetInitStates] = InitDataRot_tracking_SM(settings);

%% Initialize Solvers
mem = InitMemory(settings, opt, input);

%% Simulation
mem.iter = 1; time = 0.0;
state_sim = [input.x0];
target_sim = [targetInitStates];
controls_MPC = [input.u0];
y_sim = [];
constraints = [];
CPT = [];
ref_traj = [];
input.y(11:16,:) = 0;
target_output.x = targetInitStates';

while time(end) < Tf
  % update sun and moon positions
  positionMoon = lunar_position(JulianDay);
  positionSun = solar_position(JulianDay);

  % LOS to sun
  shadowChaser(mem.iter) = getShadowFunction([positionSun' state_sim(end,1:3) +10000]);
  shadowTarget(mem.iter) = getShadowFunction([positionSun' target_sim(end,1:3) +10000]);

  % atmospheric density
atmosDensityChaser = getAtmoDensity(norm(state_sim(end,1:3))*1000);
atmosDensityTarget = getAtmoDensity(norm(target_output.x(1:3))*1000);

% measurement of Chaser states
measuredStates = state_sim(end,:);
measuredStates = addChaserNoise(measuredStates);

% measurement of Target states
targetEstimation = [target_output.x(1:3)*1000, target_output.x(4:6)];
targetEstimation = addTargetNoise(targetEstimation);

% reference generation
[refPosx, refPosy, refPosz, refVelx, refVely, refVelz] = posRefGen(targetEstimation(1:3)', targetEstimation(4:6)', Ts, N);
input.y(1:6,1:N) = [refPosx, refPosy, refPosz, refVelx, refVely, refVelz]';
input.yN(1:6) = input.y(1:6,end);
[quaternionRef, quatref3d, chaserPosition3d, targetPosition3d, realquat3d] = estimateRotationSM(input, N, measuredStates);
input.y(7:10,:) = quaternionRef(1:N,:)
input.yN(7:10) = quaternionRef(N,:);

data.REF(mem.iter,1:10)= input.y(1:10,1); % save current ref

% call the NMPC solver
input.x0 = measuredStates';
[output, mem]=mpc_nmpcsolver(input, settings, mem, opt);

% obtain the solution and update the data
switch opt.shifting
  case 'yes'
    input.x=[output.x(:,2:end),output.x(:,end)];
    input.u=[output.u(:,2:end),output.u(:,end)];
    input.lambda=[output.lambda(:,2:end),output.lambda(:,end)];
    input.mu=[output.mu(n+1:end);output.mu(end-n+1:end)];
    input.mu_x=[output.mu_x(nbx+1:end);output.mu_x(end-nbx+1:end)];
    input.mu_u=[output.mu_u(nu+1:end);output.mu_u(end-nu+1:end)];
  case 'no'
    input.x=output.x;
    input.u=output.u;
    input.lambda=output.lambda;
    input.mu=output.mu;
    input.mu_x=output.mu_x;
    input.mu_u=output.mu_u;
end

% collect the statistics
cpt=output.info.cpuTime;
tshooting=output.info.shootTime;
tcond=output.info.condTime;
tqp=output.info.qpTime;
OptCrit=output.info.OptCrit;

% Simulate system dynamics
sim_input.x = state_sim(end,:).';
sim_input.u = output.u(:,1); % DONT TOUCH
sim_input.p = input.od(:,1)';
chaser_input.p = [positionMoon, (positionSun)', shadowChaser(mem.iter),
atmosDensityChaser];
target_input.p = [positionMoon, (positionSun)', shadowTarget(mem.iter),
atmosDensityTarget];
% chaser_input.p = 0;
% target_input.p = 0;
target_input.x = target_output.x;

% simulate chaser
xf=full( Simulate_system_chaserSM('Simulate_system_chaserSM', sim_input.x, sim_input.u, chaser_input.p) );
% xf=full( Simulate_system('Simulate_system', sim_input.x, sim_input.u, chaser_input.p) );

% simulate target
target_output.x = full( Simulate_system_targetSM('Simulate_system_targetSM', target_input.x, 0, target_input.p) );
% target_output.x = full ( Simulate_system_target_noiseless('Simulate_system_target_noiseless', target_input.x, 0, target_input.p) );

% Collect outputs
y_sim = [y_sim; full(h_fun('h_fun', xf, sim_input.u, sim_input.p))];
% Collect constraints
constraints=[constraints; full( path_con_fun('path_con_fun', xf, sim_input.u, sim_input.p) )];

% store the optimal solution and states
controls_MPC = [controls_MPC; output.u(:,1)'];
state_sim = [state_sim; xf'];
target_sim = [target_sim; target_output.x'];

% go to the next sampling instant
nextTime = mem.iter*Ts;
mem.iter = mem.iter+1;
disp(['current time:' num2str(nextTime) 'CPT:' num2str(cpt) 'SHOOTING: ' num2str(tshooting) 'COND:' num2str(tcond) 'QP: ' num2str(tqp) 'Opt:' num2str(OptCrit) 'SQP IT:' num2str(output.info.iteration_num)]);

% update sim time and Julian Date
time = [time nextTime];
JulianDay = JulianDay + time(end)/(24*3600);
CPT = [CPT; cpt, tshooting, tcond, tqp];
end

if strcmp(opt.qpsolver, 'qpoases')
    qpoases_sequence('c', mem.warm_start);
end
clear mex;
disp(['Average CPT:', num2str(mean(CPT(2:end,:),1))]);
disp(['Maximum CPT:', num2str(max(CPT(2:end,:)))]);

% Draw2;

---

A.2 Main file (Open-loop Simulation)

% MATMPC: orbits - ORBITal and attitude Simulator & controller
% Main file (open-loop)
% Orbital parameters are defined here
% Contains an open-loop system
% Chaser is simulated
% Target is pre-generated

155
clear mex; close all; clc;

%% Configuration (complete your configuration here...)
addpath([pwd,'/nmpc']);
addpath([pwd,'/model_src']);
addpath([pwd,'/mex_core']);
addpath(genpath([pwd,'/data']));

cd data;
if exist('settings','file')==2
    load settings
    cd..
else
    cd..
    error('No setting data is detected!');
end

Tf = 500000;  % simulation time
settings.Tf = Tf;
Ts = settings.Ts;  % Sampling time
Ts_st = settings.Ts_st;  % Shooting interval
s = settings.s;  % number of integration steps per interval
nx = settings.nx;  % No. of states
nu = settings.nu;  % No. of controls
ny = settings.ny;  % No. of outputs (references)
nyN = settings.nyN;  % No. of outputs at terminal stage
np = settings.tp;  % No. of parameters (on-line data)
nc = settings.nc;  % No. of constraints
ncN = settings.ncN;  % No. of constraints at terminal stage
nbx = settings.nbx;

%% Orbit config
settings.refGen = 'with perturbations';  % 'with perturbations', 'without perturbations'
settings.shootingPointInit = 'with perturbations';  % 'with perturbations', 'none'
settings.thrusterForce = 1;  % maximum thruster strength in mN
settings.torqueForce = .5;  % Maximum torque in uNm
settings.dThrustMax = .1;
settings.dTorqueMax = 1e-1;
settings.maxAngVel = 5e-3;
settings.minDist = 7.760;  % min dist to Earth CoM, Mm

%% Orbit parameters
% Chaser initial conditions
settings.chaser.semiMajorAxis = 7800.0;
settings.chaser.eccentricity = 0.00;
settings.chaser.inclination = 0;
settings.chaser.trueAnomaly = -pi/10;
settings.chaser.argumentofPerigee = 0;
settings.chaser.RAAN = 0;

% Target initial conditions
settings.target.semiMajorAxis = 7800;
settings.target.eccentricity = 0.00;
settings.target.inclination = 0;
settings.target.trueAnomaly = 0;
settings.target.argumentofPerigee = 0;
settings.target.RAAN = 0;

%% solver configurations
N = 800;  % No. of shooting points
settings.N = N;
N2 = 8;
settings.N2 = N2;  % No. of horizon length after partial condensing (N2=1 means full condensing)
opt.integrator='ERK4';  % 'ERK4', 'IRK3', 'ERK4-CASADI'
opt.hessian='gauss_newton';  % 'gauss_newton'
opt.condensing='default_full';  %'default_full','no','blasfeo_full','
partial_condensing'
opt.qpsolver='qpoases';
opt.hotstart='no';  %'yes','no' (only for qpoases)
opt.shifting='yes';  %'yes','no'
opt.lin_obj='yes';  %'yes','no'  % if objective function is linear least square
opt.ref_type=2;  % 0-time invariant, 1-time varying(no preview), 2-time varying (preview)

%% available qpsolver
% 'qpoases' (for full condensing)
% 'qpoases_mb' (for full condensing+moving block)
% 'quadprog_dense' (for full condensing)
% 'hpipm_sparse' (set opt.condensing='no!')
% 'hpipm_pcond' (set opt.condensing='no')
% 'ipopt_dense' (for full condensing)
% 'ipopt_sparse' (set opt.condensing='no')
% 'ipopt_partial_sparse' (set opt.condensing='partial_condensing'; only for state
  and control bounded problems)
% 'osqp_sparse' (set opt.condensing='no')
% 'osqp_partial_sparse' (set opt.condensing='partial_condensing')

%% Initialize Data (all users have to do this)
[input, data] = InitDataRot_tracking(settings);

set(gcf, 'PaperPositionMode', 'auto');
% in order to make matlab to do not "cut" latex-interpreted axes labels
set(gca, 'Units', 'normalized',...
  % 'Position',[0.15 0.2 0.75 0.7]);

%% Initialize Solvers (only for advanced users)
mem = InitMemory(settings, opt, input);

%% Simulation (start your simulation...)
mem.iter = 1; time = 0.0;
state_sim= [input.x0]
controls_MPC = [input.u0]
y_sim = [];
constraints = [];
CPT = [];
ref_traj = [];
input.y(7:10,:) = data.REF(1:N,7:10)';
while time(end) < Tf
  % the reference input.y is a ny by N matrix
  % the reference input.yN is a nyN by 1 vector
  switch opt.ref_type
    case 0  % time-invariant reference
input.y = repmat(data.REF',1,N);
input.yN = data.REF(1:nyN)';
case 1 % time - varying reference (no reference preview)
 input.y = repmat(data.REF(mem.iter,:)',1,N);
 input.yN = data.REF(mem.iter,1:nyN)';
case 2 %time - varying reference (reference preview)
 input.y(1:6,:) = data.REF(mem.iter:mem.iter+N-1,1:6)';
 input.yN(1:6) = data.REF(mem.iter+N,1:6)';
 input.y(11:16,:) = 0;
input.y(7:10,:)= quaternionRef(1,:)';
input.yN(7:10) = quaternionRef(N,:);
end
% obtain the state measurement
measuredStates=state_sim(end,:);
input.x0 = state_sim(end,:)';
% call the NMPC solver
[output, mem]=mpc_nmpcsolver(input, settings, mem, opt);
% obtain the solution and update the data
switch opt.shifting
 case 'yes'
 input.x=[output.x(:,2:end),output.x(:,end)];
 input.u=[output.u(:,2:end),output.u(:,end)];
 input.lambda=[output.lambda(:,2:end),output.lambda(:,end)];
 input.mu=[output.mu(nc+1:end);output.mu(end-nc+1:end)];
 input.mu_x=[output.mu_x(nb+1:end);output.mu_x(end-nbx+1:end)];
 input.mu_u=[output.mu_u(nu+1:end);output.mu_u(end-nu+1:end)];
 case 'no'
 input.x=output.x;
 input.u=output.u;
 input.lambda=output.lambda;
 input.mu=output.mu;
 input.mu_x=output.mu_x;
 input.mu_u=output.mu_u;
end
% collect the statistics
cpt=output.info.cpuTime;
tshooting=output.info.shootTime;
tcond=output.info.condTime;
tqp=output.info.qpTime;
OptCrit=output.info.OptCrit;
% Simulate system dynamics
sim_input.x = state_sim(end,:).';
sim_input.u = output.u(:,1); % DONT TOUCH
sim_input.p = input.od(:,1)';
xf=full( Simulate_system('Simulate_system', sim_input.x, sim_input.u, sim_input.p) ) ;
% Collect outputs
y_sim = [y_sim; full(h_fun('h_fun', xf, sim_input.u, sim_input.p))'];
% Collect constraints
189 constraints=[constraints; full(path_con_fun('path_con_fun', xf, sim_input.u , sim_input.p ))];
190
% store the optimal solution and states
191 controls_MPC = [controls_MPC; output.u(:,:1)];
192 state_sim = [state_sim; xf];
193
% go to the next sampling instant
194 nextTime = mem.iter*Ts;
195 mem.iter = mem.iter+1;
196 disp(['current␣time:' num2str(nextTime) 'U_UU_CPT:' num2str(cpt) 'U_UU_SHOOTING: ' num2str(tshooting) 'U_UU_COND:' num2str(tcond) 'U_UU_QP:' num2str(tqp) 'U_UU_0pt:' num2str(OptCrit) 'U_UU_SQP_IT:' num2str(output.info.iteration_num )]);
197
198 time = [time nextTime];
199
200 if strcmp(opt.qpsolver, 'qpoases')
201 qpOASES_sequence( 'c', mem.warm_start);
202 end
203 clear mex;
204
205 disp(['Average␣CPT:␣' num2str(mean(CPT(2:end,:),1))]);
206 disp(['Maximum␣CPT:␣' num2str(max(CPT(2:end,:)))]);

A.3 Simulated system (chaser)

%---------------------------------------------------------%
% MATMPC: orbits - ORBITal and attitude Simulator & controller
%---------------------------------------------------------%
% Accurate model of the system including orbital perturbations:
% J2, drag, lunar and solar gravity, solar radiation pressure
% %
% % Assumptions:
% % 3U spacecraft
% % Canonball model
% % Thrusters do not produce torque or consume mass
% % Constant solar activity
% % Only gravity gradient torque is considered for attitude perturbations
% % No protruding solar panels
%---------------------------------------------------------%

% Dimensions
nx=19; % No. of states
nu=6; % No. of controls
ny=16; % No. of outputs
nyN=10; % No. of outputs at the terminal point
np=8; % No. of model parameters
nc=0; % No. of general constraints
ncN=0; % No. of general constraints at the terminal point
nbx = 9; % No. of bounds on states
nbu = 6; % No. of bounds on controls

% state and control bounds
nbx_idx = [7,8,9,14,15,16, 17, 18, 19]; % indices of states which are bounded
nbu_idx = [1,2,3,4,5,6]; % indices of controls which are bounded

159
%% create variables
import casadi.*

states = SX.sym('states',nx,1);
controls = SX.sym('controls',nu,1);
params = SX.sym('paras',np,1);
refs = SX.sym('refs',ny,1);
refN = SX.sym('refs',nyN,1);
Q = SX.sym('Q',ny,ny);
QN = SX.sym('QN',nyN,nyN);

%% Dynamics
mu = 398600; % gravity constant Earth
mu_moon = 4903;
RE = 6378; % Earth radius (km)
J2 = 1082.63e-6; % Earth non-symmetry constant
m = 4; % chaser mass (kg)
A = 0.3; % chaser area (m^2)
CR = 1.5; % radiation pressure coefficient (1 <= CR <= 2)
S = 1367; % Solar constant (W/m^2)
c = 3e8; % speed of light (m/s)
angVel_earth = 72.9211e-6; % earth angular velocity (rad/s)
CD = 1; % drag coefficient

% moments of inertia
Ix=0.03333;
Iy=0.006666;
Iz=0.03333;

% position ECI (Mm)
xpos=states(1);
ypos=states(2);
zpos=states(3);

% velocity ECI (km/s)
xvel=states(4);
yvel=states(5);
zvel=states(6);

% thruster force in body frame
Tx=states(7);
Ty=states(8);
Tz=states(9);

% quaternions (measure rotation from initial state to current state)
q4=states(10);
q1=states(11);
q2=states(12);
q3=states(13);

% angular velocity body frame
wx=states(14);
wy=states(15);
wz=states(16);

% Torque applied in body frame
Torquex=states(17);
Torquey=states(18);
Torquez=states(19);

% thruster force derivative
dTx=controls(1);
dT=controls(2);
dTz=controls(3);

% derivative of torque applied
dTorquex=controls(4);
dTorquey=controls(5);
dTorquez=controls(6);

% Moon and sun positions
lunarPosx = params(1);
lunarPosy = params(2);
lunarPosz = params(3);
solarPosx = params(4);
solarPosy = params(5);
solarPosz = params(6);

sunLOS = params(7); % photons from sun can reach object (true or false)

atmosphericDensity = params(8);

% convert Mm to km
xposkm=1000*xpos;
yposkm=1000*ypos;
zposkm=1000*zpos;
r=sqrt(xposkm^2+yposkm^2+zposkm^2); % distance to Earth center of mass in km

% relative distances (moon, sun, spacecraft)
distMoonEarth = norm([lunarPosx,lunarPosy,lunarPosz]);
distSunEarth = norm([solarPosx,solarPosy,solarPosz]);
distMoonSC = sqrt((lunarPosx-xposkm)^2 + (lunarPosy-yposkm)^2 + (lunarPosz-zposkm)^2);
distSunSC = sqrt((solarPosx-xposkm)^2 + (solarPosy-yposkm)^2 + (solarPosz-zposkm)^2);

% acceleration caused by moon (in ECI frame)
xaccLunarGrav = mu_moon*( (lunarPosx-xposkm)/distMoonSC^3 - lunarPosx/distMoonEarth^3 )/1000;
yaccLunarGrav = mu_moon*( (lunarPosy-yposkm)/distMoonSC^3 - lunarPosy/distMoonEarth^3 )/1000;
zaccLunarGrav = mu_moon*( (lunarPosz-zposkm)/distMoonSC^3 - lunarPosz/distMoonEarth^3 )/1000;

% acceleration caused by sun (in ECI frame)
xaccSolarGrav = mu_sun*( (solarPosx-xposkm)/distSunSC^3 - solarPosx/distSunEarth^3 )/1000;
yaccSolarGrav = mu_sun*( (solarPosy-yposkm)/distSunSC^3 - solarPosy/distSunEarth^3 )/1000;
zaccSolarGrav = mu_sun*( (solarPosz-zposkm)/distSunSC^3 - solarPosz/distSunEarth^3 )/1000;

% solar radiation pressure
SRPx = -sunLOS * S/c * CR * A/m * (solarPosx-xposkm)/distSunSC/1e6;
SRPy = -sunLOS * S/c * CR * A/m * (solarPosy-yposkm)/distSunSC/1e6;
SRPz = -sunLOS * S/c * CR * A/m * (solarPosz-zposkm)/distSunSC/1e6;

% relative velocity to Earth atmosphere
relVel_SC_earth_x = xvel + angVel_earth*yvel;
relVel_SC_earth_y = yvel - angVel_earth*xvel;
relVel_SC_earth_z = zvel;
relVel_SC_earth = norm([relVel_SC_earth_x,relVel_SC_earth_y,relVel_SC_earth_z]);

% acceleration from drag
\[
\begin{align*}
\text{drag}_x &= \frac{-1}{2} \times \text{atmosphericDensity} \times \text{relVel}_{SC\_earth} \times (CD \times A/m) \times \text{relVel}_{SC\_earth\_x} / 1000; \\
\text{drag}_y &= \frac{-1}{2} \times \text{atmosphericDensity} \times \text{relVel}_{SC\_earth} \times (CD \times A/m) \times \text{relVel}_{SC\_earth\_y} / 1000; \\
\text{drag}_z &= \frac{-1}{2} \times \text{atmosphericDensity} \times \text{relVel}_{SC\_earth} \times (CD \times A/m) \times \text{relVel}_{SC\_earth\_z} / 1000; \\
\end{align*}
\]

\% thrusters (body frame) -> thrusters (ECI)
\[
\begin{align*}
\text{Tx}_{ECI} &= (1 - 2*(-q2)^2 - 2*(-q3)^2) \times \text{Tx} + (2*(-q1)*(-q2) + 2*(-q2)*q4) \times \text{Ty} + (2*(-q1)*(-q3) - 2*(-q2)*q4) \times \text{Tz}; \\
\text{Ty}_{ECI} &= (2*(-q1)*(-q2) - 2*(-q3)*q4) \times \text{Tx} + (1 - 2*(-q1)^2 - 2*(-q3)^2) \times \text{Ty} + (2*(-q2)*(-q3) + 2*(-q1)*q4) \times \text{Tz}; \\
\text{Tz}_{ECI} &= (2*(-q1)*(-q3) + 2*(-q2)*q4) \times \text{Tx} + (2*(-q2)*(-q3) - 2*(-q1)*q4) \times \text{Ty} + (1 - 2*(-q1)^2 - 2*(-q2)^2) \times \text{Tz}; \\
\end{align*}
\]

\% quaternions rate of change
\[
\begin{align*}
q_{4\_dot} &= (-wx*q1 - wy*q2 - wz*q3)/2; \quad \% \text{scalar} \\
q_{1\_dot} &= (wx*q4 + wz*q2 - wy*q3)/2; \\
q_{2\_dot} &= (wy*q4 - wz*q1 + wx*q3)/2; \\
q_{3\_dot} &= (wz*q4 + wy*q1 - wx*q2)/2; \\
\end{align*}
\]

\% gravity gradient torque
\[
\begin{align*}
\text{gravGrad}_x &= 3*mu/r^3 \times (I_z - I_y) \times (2*q2*q3 - 2*q1*q4) \times (1 - 2*q1^2 - 2*q2^2); \\
\text{gravGrad}_y &= -3*mu/r^3 \times (I_z - I_x) \times (2*q1*q3 + 2*q2*q4) \times (1 - 2*q1^2 - 2*q2^2); \\
\text{gravGrad}_z &= -3*mu/r^3 \times (I_x - I_y) \times (2*q1*q3 + 2*q2*q4) \times (2*q2*q3 - 2*q1*q4); \\
\end{align*}
\]

\% angular acceleration
\[
\begin{align*}
wx\_dot &= ((\text{Torquex}/1e6 + \text{gravGrad}_x) - (I_z-I_y)*wx*wz)/(I_x); \\
wy\_dot &= ((\text{Torquey}/1e6 + \text{gravGrad}_y) - (I_z-I_z)*wx*wx)/(I_y); \\
wz\_dot &= ((\text{Torquez}/1e6 + \text{gravGrad}_z) - (I_y-I_x)*wx*wx)/(I_z); \\
\end{align*}
\]

\% total acceleration
\[
\begin{align*}
xvel\_dot &= -mu*xposkm/r^3 + 15/2*J2*mu*RE^2/r^7*xposkm*zposkm^2 - 3/2*J2*mu*RE^2/r^5*xposkm + \text{xaccSolarGrav} + \text{SRPx} + \text{xaccLunarGrav} + \text{Tx}_{ECI}/(1e6*m) + \text{drag}_x; \\
yvel\_dot &= -mu*yposkm/r^3 + 15/2*J2*mu*RE^2/r^7*yposkm*zposkm^2 - 3/2*J2*mu*RE^2/r^5*yposkm + \text{yaccSolarGrav} + \text{SRPy} + \text{yaccLunarGrav} + \text{Ty}_{ECI}/(1e6*m) + \text{drag}_x; \\
zvel\_dot &= -mu*zposkm/r^3 + 15/2*J2*mu*RE^2/r^7*zposkm^3 - 9/2*J2*mu*RE^2/r^5*zposkm + \text{zaccSolarGrav} + \text{SRPz} + \text{zaccLunarGrav} + \text{Tz}_{ECI}/(1e6*m) + \text{drag}_x; \\
\end{align*}
\]

\% define states derivate vector
\[
\begin{align*}
\text{x}\_dot &= \begin{bmatrix} xvel/1000; yvel/1000; zvel/1000; xvel\_dot; yvel\_dot; zvel\_dot; dTx; dTy; dTz; q4\_dot; q1\_dot; q2\_dot; q3\_dot; wx\_dot; wy\_dot; wz\_dot; dTorquex; dTorquey; dTorquez \end{bmatrix}; \\
\text{xdot} &= \text{SX}\_\text{sym}(\text{xdot}', nx, 1); \\
\text{impl}_{f} &= \text{xdot} - \text{x}\_dot; \\
\end{align*}
\]

%% Objectives and constraints
%% objectives
\[
\begin{align*}
\text{h} &= \begin{bmatrix} \text{xpos}; \text{ypos}; \text{zpos}; \text{xvel}; \text{yvel}; \text{zvel}; \text{q4}; \text{q1}; \text{q2}; \text{q3}; \text{dTx}; \text{dTy}; \text{dTz}; \text{dTorquex}; \text{dTorquey}; \text{dTorquez} \end{bmatrix}; \\
\text{hN} &= \text{h}(1:nyN); \\
\end{align*}
\]

%% general constraints (not used)
\[
\begin{align*}
\text{general}_\text{con} &= []; \\
\text{general}_\text{con}\_N &= []; \\
\end{align*}
\]

%% NMPC sampling time [s]
\[
\text{Ts} = 10; \quad \% \text{simulation sample time} \\
\text{Ts}\_\text{st} = 10; \quad \% \text{shooting interval time} \\
\]

%% build casadi function (don't touch)
\[
\begin{align*}
\text{h}\_\text{fun} &= \text{Function}(\text{h}\_\text{fun}', \{\text{states}, \text{controls}, \text{params}\}, \{\text{h}\}, \{\text{states}', \text{controls}', \text{params}'\}) \{\text{h}'\}; \\
\end{align*}
\]
A.4 Simulated system (target)

%---------------------------------------------------%
% MATMPC: orbits - ORBITal and attitude Simulator & controller
%---------------------------------------------------%
% Accurate model of the system including orbital perturbations:
% J2, drag, lunar and solar gravity, solar radiation pressure
% Used to model the target
%---------------------------------------------------%
% Assumptions:
% Dead 3U spacecraft
% Canonball model
% Constant solar activity
% Only gravity gradient torque is considered for attitude perturbations
% No protruding solar panels
%---------------------------------------------------%
% Options
includePerturbations = 'yes'; % 'no', 'yes'
%---------------------------------------------------%
% Dimensions
nx=13; % No. of states
nu=0; % No. of controls
ny=0; % No. of outputs
nyN=0; % No. of outputs at the terminal point
np=8; % No. of model parameters
nc=0; % No. of general constraints
ncN=0; % No. of general constraints at the terminal point
nbx = 0; % No. of bounds on states
nbu = 0; % No. of bounds on controls
% state and control bounds
nbx_idx = []; % indexes of states which are bounded
nbu_idx = []; % indexes of controls which are bounded
%---------------------------------------------------%
% create variables
import casadi.*
states = SX.sym('states',nx,1);
controls = SX.sym('controls',nu,1);
params = SX.sym('params',np,1);
refs = SX.sym('refs',ny,1);
refN = SX.sym('refN',nyN,1);
Q = SX.sym('Q',ny,ny);
QN = SX.sym('QN',nyN,nyN);
%---------------------------------------------------%
% Dynamics
mu = 398600; % gravity constant
mu_moon = 4903;
mu_sun = 132.709e9;
RE = 6378; % Earth radius km
J2 = 1082.63e-6; % Earth non-symmetry constant
\[ m = 4; \quad \% \text{chaser mass kg} \]
\[ A = 0.3; \quad \% \text{chaser area (m}^2) \]
\[ CR = 1.5; \quad \% \text{radiation pressure coefficient (1 <= CR <= 2)} \]
\[ S = 1367; \quad \% \text{Solar constant (W/m}^2) \]
\[ c = 3e8; \quad \% \text{speed of light (m/s)} \]
\[ \text{angVel} \_\text{earth} = 72.9211e-6; \quad \% \text{earth angular velocity (rad/s)} \]
\[ CD = 1; \quad \% \text{drag coefficient} \]
\[ \% \text{moments of inertia} \]
\[ I_x = 0.03333; \]
\[ I_y = 0.006666; \]
\[ I_z = 0.03333; \]
\[ \% \text{position ECI (Mm)} \]
\[ x_{\text{pos}} = \text{states}(1); \]
\[ y_{\text{pos}} = \text{states}(2); \]
\[ z_{\text{pos}} = \text{states}(3); \]
\[ \% \text{velocity ECI (km/s)} \]
\[ x_{\text{vel}} = \text{states}(4); \]
\[ y_{\text{vel}} = \text{states}(5); \]
\[ z_{\text{vel}} = \text{states}(6); \]
\[ \% \text{quaternions (measure rotation from initial state to current state)} \]
\[ q_4 = \text{states}(7); \]
\[ q_1 = \text{states}(8); \]
\[ q_2 = \text{states}(9); \]
\[ q_3 = \text{states}(10); \]
\[ \% \text{angular velocity body frame} \]
\[ w_x = \text{states}(11); \]
\[ w_y = \text{states}(12); \]
\[ w_z = \text{states}(13); \]
\[ \% \text{Moon and sun positions} \]
\[ \text{lunarPos}_x = \text{params}(1); \]
\[ \text{lunarPos}_y = \text{params}(2); \]
\[ \text{lunarPos}_z = \text{params}(3); \]
\[ \text{solarPos}_x = \text{params}(4); \]
\[ \text{solarPos}_y = \text{params}(5); \]
\[ \text{solarPos}_z = \text{params}(6); \]
\[ \text{sunLOS} = \text{params}(7); \quad \% \text{photons from sun can reach object (1 or 0)} \]
\[ \text{atmosphericDensity} = \text{params}(8); \]
\[ \% \text{convert Mm to km} \]
\[ x_{\text{pos}} \text{km} = 1000 \times x_{\text{pos}}; \]
\[ y_{\text{pos}} \text{km} = 1000 \times y_{\text{pos}}; \]
\[ z_{\text{pos}} \text{km} = 1000 \times z_{\text{pos}}; \]
\[ r = \sqrt{x_{\text{pos}} \text{km}^2 + y_{\text{pos}} \text{km}^2 + z_{\text{pos}} \text{km}^2}; \quad \% \text{distance to Earth center of mass in km} \]
\[ \text{distMoonEarth} = \text{norm}([\text{lunarPos}_x, \text{lunarPos}_y, \text{lunarPos}_z]); \]
\[ \text{distSunEarth} = \text{norm}([\text{solarPos}_x, \text{solarPos}_y, \text{solarPos}_z]); \]
\[ \text{distMoonSC} = \sqrt{(\text{lunarPos}_x - x_{\text{pos}} \text{km})^2 + (\text{lunarPos}_y - y_{\text{pos}} \text{km})^2 + (\text{lunarPos}_z - z_{\text{pos}} \text{km})^2}; \]
\[ \text{distSunSC} = \sqrt{(\text{solarPos}_x - x_{\text{pos}} \text{km})^2 + (\text{solarPos}_y - y_{\text{pos}} \text{km})^2 + (\text{solarPos}_z - z_{\text{pos}} \text{km})^2}; \]
\[ \text{xaccLunarGrav} = \mu_{\text{moon}} \times ((\text{lunarPos}_x - x_{\text{pos}} \text{km})/\text{distMoonSC}^3 - \text{lunarPos}_x/\text{distMoonEarth}^3)/1000; \]
\[yaccLunarGrav = \mu_{moon} \times \frac{(lunarPosy-yposkm)/distMoonSC^3 - lunarPosy/distMoonEarth^3}{1000};\]
\[zaccLunarGrav = \mu_{moon} \times \frac{(lunarPosz-zposkm)/distMoonSC^3 - lunarPosz/distMoonEarth^3}{1000};\]
\[xaccSolarGrav = \mu_{sun} \times \frac{(solarPosx-xposkm)/distSunSC^3 - solarPosx/distSunEarth^3}{1000};\]
\[yaccSolarGrav = \mu_{sun} \times \frac{(solarPosy-yposkm)/distSunSC^3 - solarPosy/distSunEarth^3}{1000};\]
\[zaccSolarGrav = \mu_{sun} \times \frac{(solarPosz-zposkm)/distSunSC^3 - solarPosz/distSunEarth^3}{1000};\]
\[SRPx = -sunLOS \times S/c \times CR \times A/m \times \frac{(solarPosx-xposkm)/distSunSC}{1e6};\]
\[SRPy = -sunLOS \times S/c \times CR \times A/m \times \frac{(solarPosy-yposkm)/distSunSC}{1e6};\]
\[SRPz = -sunLOS \times S/c \times CR \times A/m \times \frac{(solarPosz-zposkm)/distSunSC}{1e6};\]
\[relVel_SC_earth_x = xvel + angVel_earth \times yvel;\]
\[relVel_SC_earth_y = yvel - angVel_earth \times xvel;\]
\[relVel_SC_earth_z = zvel;\]
\[relVel_SC_earth = \sqrt{relVel_SC_earth_x^2 + relVel_SC_earth_y^2 + relVel_SC_earth_z^2};\]
\[dragx = -1/2 \times atmosphericDensity \times relVel_SC_earth \times (CD*A/m) \times \frac{xvel}{1000};\]
\[dragy = -1/2 \times atmosphericDensity \times relVel_SC_earth \times (CD*A/m) \times \frac{yvel}{1000};\]
\[dragz = -1/2 \times atmosphericDensity \times relVel_SC_earth \times (CD*A/m) \times \frac{zvel}{1000};\]
\[\% \ \text{quaternions \ rate \ of \ change}\]
\[q4_dot = (-wx*q1 - wy*q2 - wz*q3)/2; \ \% \ \text{scalar}\]
\[q1_dot = (wx*q4 + wz*q2 - wy*q3)/2;\]
\[q2_dot = (wy*q4 - wz*q1 + wx*q3)/2;\]
\[q3_dot = (wz*q4 + wy*q1 - wx*q2)/2;\]
\[\% \ \text{gravity \ gradient \ torque}\]
\[gravGradx = 3*mu/r^3*(I_z - I_y)*(2*q2*q3 - 2*q1*q4)*(1 - 2*q1^2 - 2*q2^2);\]
\[gravGrady = -3*mu/r^3*(I_z - I_x)*(2*q1*q3 + 2*q2*q4)*(1 - 2*q1^2 - 2*q2^2);\]
\[gravGradz = -3*mu/r^3*(I_x - I_y)*(2*q1*q3 + 2*q2*q4)*(2*q2*q3 - 2*q1*q4);\]
\[\% \ \text{angular \ acceleration}\]
\[wx_dot = ((gravGradx) - (I_z-I_y)*wx*wz)/(I_x);\]
\[wy_dot = ((gravGrady) - (I_x-I_z)*wz*wx)/(I_y);\]
\[wz_dot = ((gravGradz) - (I_y-I_x)*wx*wy)/(I_z);\]
\[\% \ \text{objectives \ and \ constraints}\]
\[h = [0];\]
\[hN = h(1:nyN);\]
% general inequality constraints
% general_con = (xpos^2 + ypos^2 + zpos^2) - minDist^2;
% general_con_N = (xpos^2 + ypos^2 + zpos^2) - minDist^2;
general_con = [];
general_con_N = [];

%% NMPC sampling time [s]
Ts = 10; % simulation sample time
Ts_st = 10; % shooting interval time

%% build casadi function (don't touch)
h_fun=Function('h_fun', {states,controls,params}, {h},{'states','controls','params'},{'h'});
hN_fun=Function('hN_fun', {states,params}, {hN},{'states','params'},{'hN'});
path_con_fun=Function('path_con_fun', {states,controls,params}, {general_con},{'states','controls','params'},{'general_con'});
path_con_N_fun=Function('path_con_N_fun', {states,params}, {general_con_N},{'states','params'},{'general_con_N'});

A.5 Controller dynamics

%--- ORBITal and attitude Simulator & controller
% Simpified model of the system used for the controller
% Earth is treated as point mass + J2
%
% Assumptions:
% 3U spacecraft
% Thrusters do not produce torque or consume mass
% Constant solar activity
% Only gravity gradient torque is considered for attitude perturbations
% No protruding solar panels
%-----------------------------------------------%

% Options
includePerturbations = 'yes'; '% no', 'yes'

% Dimensions
nx=19; % No. of states
nu=6; % No. of controls
ny=16; % No. of outputs
nyN=10; % No. of outputs at the terminal point
np=0; % No. of model parameters
nc=0; % No. of general constraints
ncN=0; % No. of general constraints at the terminal point
nbx = 9; % No. of bounds on states
nbu = 6; % No. of bounds on controls

% state and control bounds
nbx_idx = [7,8,9,14,15,16, 17, 18, 19]; % indexes of states which are bounded
nbu_idx = [1,2,3,4,5,6]; % indexes of controls which are bounded

% create variables
import casadi.*
states = SX.sym('states',nx,1);
controls = SX.sym('controls',nu,1);
params = SX.sym('paras',np,1);
refs = SX.sym('refs',ny,1);
refN = SX.sym('refs',nyN,1);
Q = SX.sym('Q',ny,ny);
QN = SX.sym('QN',nyN,nyN);

%% Dynamics

mu=398600; % gravity constant Earth
RE=6378; % Earth radius km
J2=1082.63e-6; % Earth non-symmetry constant
m=4; % chaser mass kg

% moments of inertia
Ix=0.03333;
Iy=0.006666;
Iz=0.03333;

% position ECI (Mm)
xpos=states(1);
ypos=states(2);
zpos=states(3);

% velocity ECI (km/s)
xvel=states(4);
yvel=states(5);
zvel=states(6);

% thruster force in body frame
Tx=states(7);
Ty=states(8);
Tz=states(9);

% quaternions (measure rotation from initial state to current state)
q4=states(10);
q1=states(11);
q2=states(12);
q3=states(13);

% angular velocity body frame
wx=states(14);
wy=states(15);
wz=states(16);

% Torque applied in body frame
Torquex=states(17);
Torquey=states(18);
Torquez=states(19);

% thruster force derivative
dTx=controls(1);
dTy=controls(2);
dTz=controls(3);

dTorquex=controls(4);
dTorquey=controls(5);
dTorquez=controls(6);

% thrusters (body frame) -> thrusters (ECI)
TxECI = (1 - 2*(-q2)^2 - 2*(-q3)^2)*Tx + (2*(-q1)*(-q2) + 2*(-q3)*q4)*Ty + (2*(-q1)*(-q3) - 2*(-q2)*q4)*Tz;
\[ Ty_{ECI} = (2(-q_1)(-q_2) - 2(-q_3)q_4)T_x + (1 - 2(-q_1)^2 - 2(-q_3)^2)T_y + (2(-q_2)(-q_3) + 2(-q_1)q_4)T_z; \]
\[ Tz_{ECI} = (2(-q_1)(-q_3) + 2(-q_2)q_4)T_x + (2(-q_2)(-q_3) - 2(-q_1)q_4)T_y + (1 - 2(-q_1)^2 - 2(-q_2)^2)T_z; \]

% convert Mm to km
xposkm=1000*xpos;
yposkm=1000*ypos;
zposkm=1000*zpos;
r=sqrt(xposkm^2+yposkm^2+zposkm^2); % distance to Earth center of mass in km

% quaternions rate of change
q_4_dot=(-wx*q_1 - wy*q_2 - wz*q_3)/2;
q_1_dot=(wx*q_4 + wz*q_2 - wy*q_3)/2;
q_2_dot=(wy*q_4 - wz*q_1 + wx*q_3)/2;
q_3_dot=(wz*q_4 + wy*q_1 - wx*q_2)/2;

% gravity gradient torque
gravGradx = 3*mu/r^3*(I_z - I_y)*(2*q_2*q_3 - 2*q_1*q_4)*(1 - 2*q_1^2 - 2*q_2^2);
gravGrady = -3*mu/r^3*(I_z - I_x)*(2*q_1*q_3 + 2*q_2*q_4)*(1 - 2*q_1^2 - 2*q_2^2);
gravGradz = -3*mu/r^3*(I_x - I_y)*(2*q_1*q_3 + 2*q_2*q_4)*(2*q_2*q_3 - 2*q_1*q_4);

% angular acceleration
wx_dot=((Torquex/1e6 + gravGradx) - (I_z-I_y)*wy*wz)/(I_x);
wy_dot=((Torquey/1e6 + gravGrady) - (I_x-I_z)*wz*wx)/(I_y);
wz_dot=((Torquez/1e6 + gravGradz) - (I_y-I_x)*wx*wy)/(I_z);

% total acceleration
xvel_dot = -mu*xposkm/r^3 + 15/2*J_2*mu*RE^2/r^7*xposkm*zposkm^2 - 3/2*J_2*mu*RE^2/r^5*xposkm + TxECI/(1e6*m);
yvel_dot = -mu*yposkm/r^3 + 15/2*J_2*mu*RE^2/r^7*yposkm*zposkm^2 - 3/2*J_2*mu*RE^2/r^5*yposkm + TyECI/(1e6*m);
zvel_dot = -mu*zposkm/r^3 + 15/2*J_2*mu*RE^2/r^7*zposkm^3 - 9/2*J_2*mu*RE^2/r^5*zposkm + TzECI/(1e6*m);

% define states derivate vector
\[ \dot{x}_d = \frac{\text{SX}\_\text{sym}'(\text{xdot}',nx,1)}{\text{impl}_f = \text{xdot} - \dot{x}_d}; \]

% Objectives and constraints
% objectives
\[ h = [\text{xpos};\text{ypos};\text{zpos};\text{xvel};\text{yvel};\text{zvel};\text{q_4};\text{q_1};\text{q_2};\text{q_3};\text{dT_x};\text{dT_y};\text{dT_z}; \]
\[ \text{dTorquex};\text{dTorquey};\text{dTorquez}]; \]
\[ hN = h(1:nyN); \]

% general inequality constraints (not used)
general_con = []; 
general_con_N = [];

% NMPC sampling time [s]
Ts = 10; % simulation sample time
Ts_st = 10; % shooting interval time

% build casadi function (don’t touch)
h_fun=Function('h_fun', {\text{states},\text{controls},\text{params}}, \{h\}, \{'\text{states}', 'controls', 'params'\}, \{'h\'});
hN_fun=Function('hN_fun', {\text{states},\text{params}}, \{hN\}, \{'\text{states}', 'params'\}, \{'hN'\});

path_con_fun=Function('path_con_fun', {\text{states},\text{controls},\text{params}}, \{\text{general_con},\{'\text{states}', 'controls', 'params'\}, \{'general_con'\}});
A.6 Position reference generator

MATMPC: orbits - ORBITal and attitude Simulator & controller

- Generates reference values for position and velocity states
- over the prediction horizon based on the targets current position and velocity

- INPUTS : [targetPosition x, targetPosition y, targetPosition z] (km),
  [targetVelocity x, targetVelocity y, targetVelocity z] (kms/s)
  samplingRate (s)
  numberOfShootingPoints (integer)

- OUTPUTS : [refPosition x] (Mm) [1 xN]
  [refPosition y] (Mm) [1 xN]
  [refPosition z] (Mm) [1 xN]
  [refVelocity x] (Mm) [1 xN]
  [refVelocity y] (Mm) [1 xN]
  [refVelocity z] (Mm) [1 xN]

function [refPosx,refPosy,refPosz,refVelx,refVely,refVelz] = posRefGen(currentPosition, currentVelocity, Ts, N)

% Integration tolerance
ODEoptions = odeset(...
'reltol', 1.e-12, ...
'abstol', 1.e-12);

% allocate memory for reference values
refPosx = zeros(N,1);
refPosy = zeros(N,1);
refPosz = zeros(N,1);
refVelx = zeros(N,1);
refVely = zeros(N,1);
refVelz = zeros(N,1);

% Predict motion over the prediction horizon
refTimespan=0:Ts:N*Ts;
[predictionTimeSpan, predictedStates] = ode45(@integrateTargetOrbit, refTimespan, [currentPosition; currentVelocity], ODEoptions);

% Assign predicted target position/velocity as reference states
refPosx = predictedStates(1:N,1)/1000;
refPosy = predictedStates(1:N,2)/1000;
refPosz = predictedStates(1:N,3)/1000;
refVelx = predictedStates(1:N,4);
refVely = predictedStates(1:N,5);
refVelz = predictedStates(1:N,6);
refPosx=refPosx(1:N);
end

% Integration loop
function dfdt = integrateTargetOrbit(refTime,f)
  position=f(1:3);
  velocity=f(4:6);
RE = 6378;  % Earth radius
J2=1082.63e-6;  % Earth flatness around the poles
mu=398600;  % Earth gravitational parameter
r=norm(position);

position_dot = velocity;
velocity_dot=zeros(3,1);
% Treat Earth as point mass + J2
velocity_dot(1) = -mu*position(1)/r^3 + 15/2*J2*mu*RE^2/r^7*position(1)*
position(3)^2 - 3/2*J2*mu*RE^2/r^5*position(1);
velocity_dot(2) = -mu*position(2)/r^3 + 15/2*J2*mu*RE^2/r^7*position(2)*
position(3)^2 - 3/2*J2*mu*RE^2/r^5*position(2);
velocity_dot(3) = -mu*position(3)/r^3 + 15/2*J2*mu*RE^2/r^7*position(3)*
^3 - 9/2*J2*mu*RE^2/r^5*position(3);
dfdt=[position_dot; velocity_dot];

function [quatRef, quatref3d, chaserPosition3d, targetPosition3d, realquat3d] =
estimateRotationSM(input, N, measuredStates)
% ECI bases
ECIx = [1; 0; 0];
ECIy = [0; 1; 0];
ECIz = [0; 0; 1];
refquatMat=zeros(N,4);
quatError = zeros(N,4);
BODYy = zeros(3,N);  
BODYx = zeros(3,N);  
BODYz = zeros(3,N);
previousREF = input.y(7:10,1)';
chaserPosition = [measuredStates(1:3); input.x(1:3,2:end)'];
targetPosition = [input.y(1:3,:); input.yN(1:3)];
chaserVelocity = [measuredStates(4:6); input.x(4:6,2:end)'];

% Calculate DCM matrix to get a desired body frame for each point in
% the prediction horizon
for n=1:N
  % y axis points along velocity vector
  BODYy(:,n) = chaserVelocity(n,:)/norm(chaserVelocity(n,:));
  % z axis points normal to plane of motion
  BODYz(:,n) = cross(chaserPosition(n,:)',chaserVelocity(n,:)');
  BODYz(:,n) = BODYz(:,n) / norm(BODYz(:,n));
  % z axis is orthogonal to y and z
  BODYx(:,n) = cross(BODYy(:,n), BODYz(:,n));
  %
  % y = body, e=eci
  c11 = dot(BODYx(:,n),ECIx);
  c12 = dot(BODYx(:,n),ECIy);
  c13 = dot(BODYx(:,n),ECIz);
  c21 = dot(BODYy(:,n),ECIx);


A.7  Attitude reference generator
c22 = dot(BODYy(:,n),ECIy);
c23 = dot(BODYy(:,n),ECIz);
c31 = dot(BODYz(:,n),ECIx);
c32 = dot(BODYz(:,n),ECIy);
c33 = dot(BODYz(:,n),ECIz);

DCM1 = [c11 c12 c13; c21 c22 c23; c31 c32 c33];

refquatMat(n,:) = dcm2quat_mod(DCM1);

% Change sign of quaternion if that moves it closer to previous ref
quatError(1,:) = quatmultiply(refquatMat(1,:),quatconj(previousREF));
if quatError(1,1)<0
    refquatMat(1,:) = quatconj(refquatMat(1,:));
end

% Change sign of quaternion if that decreases distance between 2
% consecutive ref values
for n=2:N
    quatError(n,:) = quatmultiply(refquatMat(n,:),quatconj(refquatMat(n-1,:)));
    if quatError(n,1)<0
        refquatMat(n,:) = quatconj(refquatMat(n,:));
    end
end

targetPosition3d = [(1:N)', targetPosition(1:N,:)];
chaserPosition3d = [(1:N)', chaserPosition(1:N,:)];
quatRef3d = [(1:N)', refquatMat(1:N,:)]; % ref quat
realquat3d = [(1:N); input.x(10:13,1:N)']; % expected quat

% functions to plot during simulation
predictedDistancePlotter;
realquat3dplot;
refquatmatPlotter;

quatRef = refquatMat;

A.8 Init data Long Range Closed-loop

% Initialization
function [input, targetInitStates] = InitDataRot_tracking_SM(settings)

TF = settings.Tf;  % Simulation time
Ts = settings.Ts;  % Sampling time
Ts_st = settings.Ts_st;  % Shooting interval
s = settings.s;  % number of integration steps per interval
nx = settings.nx;  % No. of states
nu = settings.nu;  % No. of controls
ny = settings.ny;  % No. of outputs (references)
nyN = settings.nyN;  % No. of outputs at terminal stage
np = settings.np;  % No. of parameters (on-line data)
cn = settings.nc;  % No. of constraints
cnN = settings.ncN;  % No. of constraints at terminal stage
N = settings.N;  % No. of shooting points
nbx = settings.nbx;  % No. of state constraints
nbu = settings.nbu;  % No. of control constraints
nbu_idx = settings.nbu_idx;
% constraints
thrusterForce = settings.thrusterForce;
torqueForce = settings.torqueForce;
dThrustMax = settings.dThrustMax;
dTorqueMax = settings.dTorqueMax;
maxAngVel = settings.maxAngVel;

% integrator tolerance for initial shooting points
ODEoptions = odeset(...
    'reltol', 1.e-12, ...
    'abstol', 1.e-12);

% generate starting position and velocity from orbit parameters
chaserAngularMomentum = getAngularMomentum(settings.chaser.semiMajorAxis,
    settings.chaser.eccentricity);
chaserCOE = [chaserAngularMomentum, settings.chaser.eccentricity, settings.
    chaser.RAAN, ...
    settings.chaser.inclination, settings.chaser.argumentofPerigee, settings.
    chaser.trueAnomaly];
[startPos,startVel]=sv_from_coe(chaserCOE);

% body frame starts aligned with ECI frame
initQuat(1:4) = [1 0 0 0];

% initial states and controls
input.x0 = [startPos(1)/1000;startPos(2)/1000;startPos(3)/1000;startVel(1);
    startVel(2);startVel(3);0;0;0;initQuat';0;0;0;0;0;0];
input.u0 = zeros(nu,1);

para0 = [0];

% weighting matrices
posCost = 7e-5;
velCost = 7e-5;
scalarQuatCost = 2e-1; % improved: 2e1 -> 2e-1
vectorQuatCost = 2e-1;
dThrustCost = 5e6;
dTorqueCost = 1e2;

Q=diag([posCost*ones(1,3) velCost*ones(1,3) scalarQuatCost vectorQuatCost*ones
    (1,3) dThrustCost*ones(1,3) dTorqueCost*ones(1,3)]);
QN=Q(1:nyN,1:nyN);

% upper and lower bounds for states (=nbx)
lb_x = [-thrusterForce; -thrusterForce; -thrusterForce; -maxAngVel; -maxAngVel;
    -maxAngVel; -torqueForce; -torqueForce; -torqueForce];
ub_x = [thrusterForce; thrusterForce; thrusterForce; maxAngVel; maxAngVel;
    maxAngVel; torqueForce; torqueForce; torqueForce];

% upper and lower bounds for controls (=nbu)
lb_u = [-dThrustMax; -dThrustMax; -dThrustMax; -dTorqueMax; -dTorqueMax;
    -dTorqueMax];
ub_u = [dThrustMax; dThrustMax; dThrustMax; dTorqueMax; dTorqueMax; dTorqueMax];

% upper and lower bounds for general constraints (=nc)
lb_g = [];
ub_g = [];
lb_gN = [];
ub_gN = [];

% prepare the data
input.lb = repmat(lb_g,N,1);
input.ub = repmat(ub_g,N,1);
input.lb = [input.lb;lb_gN];
input.ub = [input.ub;ub_gN];
lbu = -inf(nu,1);
ubu = inf(nu,1);
for i=1:nbu
    lbu(nbu_idx(i)) = lb_u(i);
    ubu(nbu_idx(i)) = ub_u(i);
end
input.lbu = repmat(lbu,1,N);
input.ubu = repmat(ubu,1,N);
input.lbx = repmat(lb_x,1,N);
input.ubx = repmat(ub_x,1,N);

% shooting point init
shootingPointTimes = 0:Ts_st:N*Ts_st;
chaserInitStates = [startPos,startVel];
[shootingPointTime,x] = ode45(@getShootingPoints, shootingPointTimes,
    chaserInitStates, ODEoptions); % integrate over prediction horizon

% %
for i=1:N
    p(i,:) = x(:,i);
end

u = repmat(input.u0,1,N); % initialize all controls with the same initial
cpl
para = repmat(para0,1,N+1); % initialize all parameters with the same initial
para
input.x=x; % states and controls of the first N stages (nx by N+1
matrix)
input.od=para; % states of the terminal stage (nu by N vector)
input.W=Q; % on-line parameters (np by N+1 matrix)
input.WN=QN; % weights of the first N stages (ny by ny matrix)
input.lambda=zeros(nx,N+1);
input.mu=zeros(N*nc+ncN,1);
input.mu_u = zeros(N*nu,1);
input.mu_x = zeros(N*nbx,1);

% Reference generation
targetAngularMomentum = getAngularMomentum(settings.target.semiMajorAxis,
    settings.target.eccentricity);
targetCOE = [targetAngularMomentum, settings.target.eccentricity, settings.
    target.RAAN, ...
    settings.target.inclination, settings.target.argumentofPerigee, settings.
    target.trueAnomaly];

[targetStartPos, targetStartVel] = sv_from_coe(targetCOE);
targetStartPos = targetStartPos/1000;
targetStartQuat = [1 0 0 0];
targetStartAngVel = [0 0 0];
targetInitStates = [targetStartPos, targetStartVel, targetStartQuat,
    targetStartAngVel];

function dfdt = getShootingPoints(shootingPointTime,f)
    position=f(1:3);
velocity=f(4:6);
RE = 6378;
J2=1082.63e-6;
mu=398600;
r=norm(position);

position_dot = velocity;
velocity_dot=zeros(3,1);
velocity_dot(1) = -mu*position(1)/r^3 + 15/2*J2*mu*RE^2/r^5*position(1)*position(3)^2 - 3/2*J2*mu*RE^2/r^5*position(1);
velocity_dot(2) = -mu*position(2)/r^3 + 15/2*J2*mu*RE^2/r^5*position(2)*position(3)^2 - 3/2*J2*mu*RE^2/r^5*position(2);
velocity_dot(3) = -mu*position(3)/r^3 + 15/2*J2*mu*RE^2/r^5*position(3)^3 - 9/2*J2*mu*RE^2/r^5*position(3);
dfdt=[position_dot; velocity_dot];

function measuredChaserStates = addChaserNoise(realChaserStates)

%% Adds white noise to the output states of the chaser
% no noise
measuredChaserStates = realChaserStates;

% position noise (Mega-meters)
measuredChaserStates(1) = realChaserStates(1)+randn*5/1e6;
measuredChaserStates(2) = realChaserStates(2)+randn*5/1e6;
measuredChaserStates(3) = realChaserStates(3)+randn*5/1e6;

% velocity noise (km/s)
measuredChaserStates(4) = realChaserStates(4)+randn*5/1e6;
measuredChaserStates(5) = realChaserStates(5)+randn*5/1e6;
measuredChaserStates(6) = realChaserStates(6)+randn*5/1e6;

% quaternion noise (normalize to maintain unity length)
measuredChaserStates(10:13) = measuredChaserStates(10:13) + randn(4)/500;
measuredChaserStates(10:13) = measuredChaserStates(10:13)/norm(measuredChaserStates(10:13));

% angular velocity noise (rad/s)
measuredChaserStates(14:16) = realChaserStates(14:16) + randn(1,3)/5e4;

end

function measuredTargetStates = addTargetNoise(realTargetStates)

%% Adds white noise to the output states of the chaser
% position noise (km)
measuredTargetStates(1) = realTargetStates(1)+randn*5/1e3;
measuredTargetStates(2) = realTargetStates(2)+randn*5/1e3;
measuredTargetStates(3) = realTargetStates(3)+randn*5/1e3;

% velocity noise (km/s)
measuredTargetStates(4) = realTargetStates(4)+randn*5/1e6;
measuredTargetStates(5) = realTargetStates(5)+randn*5/1e6;
measuredTargetStates(6) = realTargetStates(6)+randn*5/1e6;

end
A.11 Calculate Julian Date

```matlab
function JD=calculateJulianDay(year,month,day,hour,minute,second)
% Calculates initial Julian Day number based on starting date.

% year day and month
JD1=367*year-floor((7*(floor((month+9)/12)))/4)+floor(275*month/9)+day+1721013.5;
% hour minute and second
UT=hour+minute/60+second/3600;
% add contributions
JD=JD1+UT/24;
```

B Appendix B: Simulink Plot

Figure B.0.1 shows the Simulink structure used to represent the reference frames graphically. This structure uses data from any closed-loop simulation, and it is therefore possible to see how the frames move with respect to each other during the simulation. Note that "Target.rotation" only controls the rotation of the LVLH-axes. The target itself is represented as a sphere. The "Refquat.rotation" shows the reference direction for the body-frame y-axis, i.e. where the camera should be pointing.