Model-based Flight Control for a VTOL Aircraft with Independently Tilting Rotors

J.M. de Jong

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Model-based Flight Control for a VTOL Aircraft with Independently Tilting Rotors

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

J.M. de Jong

February 7, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by and partially carried out at Avy. Their cooperation is hereby gratefully acknowledged.

Abstract

This thesis reports on the research and design of a real-time Time-Varying Model Predictive Control (TVMPC) scheme to stabilize a tilt-rotor aircraft with four independently tilting rotors. The aircraft design stems from a prototype system constructed by the drone technology start-up Avy.

First, a motivation is presented for implementation of model-based flight control techniques. Consequently, TVMPC is introduced as a middle ground between Adaptive Model Predictive Control (MPC) and fully nonlinear MPC, capturing model variation over the control horizon while retaining the computational efficiency of quadratic optimization algorithms. In a simulation of a simple nonlinear system, TVMPC outperforms fixed-model and adaptive MPC using only a small set of extra models.

Secondly, the tilt-rotor system model is developed. Models for rotor torque and thrust responses and stepper motor movement are introduced and calibrated using experiments, leading to a full nonlinear model of 21 states, 8 control inputs and 8 box constraints. Further envisioned experimental work and its preparations such as aerodynamic testing and tracked constrained flight are briefly mentioned.

The implementation of TVMPC is tested in simulation. Five scenarios compare the effectiveness of time-varying control versus adaptive and fixed-model MPC, with TVMPC not outperforming fixed-model control in horizontal vehicle orientations, but improving on oscillatory behavior shown by adaptive control. TVMPC outperforms the other two controllers in a past 90 degree pitch scenario (partial backflip).

In parallel, the setup of Robot Operating System (ROS) in conjunction with a Hardware-in-the-Loop (HIL) experiment is presented to test tilt-rotor control in real-time. In closed loop, the system was stabilized for limited degrees of freedom using a combination of PID controllers, validating the framework and providing more opportunities for research.

It is concluded that TVMPC improves on the predictions of adaptive MPC, but is prone to oscillatory behavior. Only in highly nonlinear situations, fixed-model MPC is outperformed. For real-time, embedded results, more research still needs to be performed.

Table of Contents

1 Introduction **1**
- 1-1 Historical context . . . 1
- 1-2 Avy One . . . 2
- 1-3 Previous control solutions . . . 3
- 1-4 Thesis outline . . . 4

2 Time-varying MPC **5**
- 2-1 The case for model-based control . . . 5
- 2-2 TVMPC as a middle ground . . . 7
- 2-3 From model to model-based controller . . . 7
  - 2-3-1 Linearizing a non-linear dynamical system . . . 7
  - 2-3-2 Discretizing a continuous state-space system . . . 8
  - 2-3-3 Writing the state update equation in matrix form . . . 9
  - 2-3-4 MPC control in matrix form . . . 10
  - 2-3-5 QP formulation . . . 11
- 2-4 Motivating examples . . . 13

3 Model identification **17**
- 3-1 Rotor model . . . 17
  - 3-1-1 System description . . . 17
  - 3-1-2 Four-parameter combined model . . . 20
  - 3-1-3 RPM sensor calibration . . . 21
  - 3-1-4 Throttle curve measurement . . . 23
  - 3-1-5 Thrust and torque coefficient measurement . . . 25
  - 3-1-6 Dynamic response . . . 27
- 3-2 Stepper motor model . . . 29
  - 3-2-1 System description . . . 29
  - 3-2-2 Stepper Rate Limit Modeling . . . 31
- 3-3 Full system model . . . 33
  - 3-3-1 Equations of Motion . . . 34

4 Hardware-in-the-loop 37

4-1 Tiltrotor set-up . . . 37

4-1-1 System sketch . . . 37

4-1-2 Serial interface . . . 38

4-1-3 Command board . . . 39

4-1-4 Sensor board . . . 39

4-2 Simulator set-up . . . 39

4-2-1 ROS Architecture . . . 40

4-2-2 Filtering and observing . . . 40

4-2-3 Internal simulator . . . 42

4-2-4 Controllers . . . 42

4-2-5 Visualization . . . 43

4-2-6 User Interface . . . 43

5 Results 47

5-1 Simulation . . . 47

5-1-1 Yaw-right . . . 48

5-1-2 Forward and upward flight . . . 51

5-1-3 Forward and upward flight 2 . . . 53

5-1-4 Pitch-up . . . 56

5-1-5 Partial backflip . . . 58

5-2 Hardware-in-the-loop . . . 60

5-2-1 Altitude control . . . 60

5-2-2 Forward velocity control . . . 61

5-2-3 Yaw rate control . . . 61

6 Conclusions 63

6-1 TVMPC as a linear extension of adaptive MPC . . . 63

6-2 TVMPC on a tilt-rotor drone . . . 63

6-3 Hardware-in-the-loop . . . 64

6-4 Topics for further research . . . 64

7 Future work 67

7-1 Flight arena . . . 67

7-2 Optical sensors . . . 68

7-3 Wireless control box . . . 70

7-4 Wind tunnel testing . . . 70

A Mathematical derivations and definitions 73

A-1 Time-varying MPC . . . 73

A-1-1 State space discretization . . . 73

A-1-2 Prediction matrix entries . . . 74

B Coordinate framework **77**

B-1 Reference frames 77

B-1-1 Euler angle representation 78

Bibliography **81**

Glossary **85**

List of Acronyms 85

List of Symbols 86

Index **91**

List of Figures

1-1 From left to right: Harrier Jump Jet, Yak-38, V-22 Osprey and F-35B 2
1-2 A render of the preliminary Avy One design 2
1-3 Classical Cascaded Control with Inner and Outer loops 3

2-1 Principle of Model Predictive Control in a discrete scheme 6
2-2 Left: Step scenario with $R = 0.1$. Right: Step scenario with $R = 0.3$. 14
2-3 Sine wave tracking with $N_P = 30$ and extended control limits for fixed, adaptive and full time-varying MPC 14
2-4 Sine wave tracking with $N_P = 30$ and extended control limits for time-varying MPC with reduced model variation 15
2-5 Post-0.5s RMS tracking error 15

3-1 From left to right: Multistar 5008 KV330, 30A AfroESC and Quanum 18x5.5 propeller 20
3-2 Left: Effect of parameter a on the the step response. Right: Effect of parameter b on the the step response. 21
3-3 Eagle Tree RPM sensor 22
3-4 Left: Sensor timing results and difference with expectation. Right: Sensor linearity analysis 23
3-5 PWM board input values to RPM curves 24
3-6 ESC PWM to RPM curves 25
3-7 RPM and thrust analysis 26
3-8 Thrust coefficient analysis 26
3-9 Torque coefficient analysis 27
3-10 User prompt during rise time analysis 28
3-11 Fitted rise times for a variety of target throttle levels 28
3-12 Fitting the overshoot parameter to load cell data 29

3-13 Left: perspective render of the T3 CAD design with a single rotor. Right: Constructed T3 experiment in a coaxial testing configuration 30
3-14 Left: Arduino UNO CNC Shield. Right: Nema 17 stepper motor 30
3-15 Time-series evolution of the stepper motor angle 32

4-1 Plan view of the hardware-in-the-loop system 38
4-2 ROS overview of all active nodes and topics 41
4-3 rViz visualization window displaying live position, thrust forces and tilt angles . . 43
4-4 Navigation Tab 44
4-5 PID and MPC Tuning Tab 45

5-1 Body velocities, angles and control inputs for fixed-model MPC in a yaw-right scenario 48
5-2 Body velocities, angles and control inputs for adaptive MPC in a yaw-right scenario 49
5-3 Body velocities, angles and control inputs for TVMPC in a yaw-right scenario . . 49
5-4 Body velocities, angles and control inputs for fixed-model MPC in a lift-off scenario 51
5-5 Body velocities, angles and control inputs for adaptive MPC in a lift-off scenario 52
5-6 Body velocities, angles and control inputs for TVMPC in a lift-off scenario 52
5-7 Body velocities, angles and control inputs for TVMPC in a lift-off scenario 53
5-8 Body velocities, angles and control inputs for TVMPC in a lift-off scenario 54
5-9 Body velocities, angles and control inputs for TVMPC in a lift-off scenario 54
5-10 Left: V_x, V_z and θ for a heavily rotor penalized liftoff scenario. Right: V_x, V_z and θ for a lightly rotor penalized liftoff scenario. 55
5-11 Body velocities, angles and control inputs for fixed-model MPC in a pitch-up scenario 56
5-12 Body velocities, angles and control inputs for adaptive MPC in a pitch-up scenario 57
5-13 Body velocities, angles and control inputs for TVMPC in a pitch-up scenario . . 57
5-14 Body velocities, angles and control inputs for fixed-model MPC in a partial backflip scenario 58
5-15 Body velocities, angles and control inputs for adaptive MPC in a partial backflip scenario 59
5-16 Body velocities, angles and control inputs for TVMPC in a partial backflip scenario 59
5-17 Real-time HIL altitude control on a restricted system 60
5-18 Filtered RPM sensor values during the hover test 61
5-19 Real-time HIL forward velocity control on a restricted system 62
5-20 Real-time HIL yaw rate control on a restricted system 62

7-1 Left: Sensor track on the X axis of the flight experiment. Right: Partially constructed flight arena 68
7-2 Relative linear position pattern for a 10-sensor tracking device - flat surface solution 69
7-3 COPS (Cheap Optical Position Sensor) - linear version 69
7-4 Relative linear position pattern for a 10-sensor tracking device - freely moving axis solution 70
7-5 COPS (Cheap Optical Position Sensor) - rotating version 70

7-6 Wireless control box comprising 8 toggle switches 71
7-7 Experimental wind tunnel in the machining hall of Avy 71

B-1 Body and world reference frames 77

List of Tables

2-1 Optimization result at the end of the simulation, for $R = 0.3$ 13
2-2 RMS tracking errors for the sine wave tracking case 16

3-1 Arming frequency and signal limits for all ESC's 23
3-2 PWM to RPM models 24
3-3 Average throttle to RPM model 25
3-4 Thrust and torque model fits 26
3-5 Rotor positions and spinning directions (1 = clockwise, -1 is counterclockwise) . 34
3-6 Virtual throttle mapping function coefficients 36

5-1 MPC state penalty weights 48
5-2 MPC control penalty weights 48
5-3 Step reference signal for the yaw-right scenario 48
5-4 Step reference signal for the lift-off scenario 51
5-5 MPC state penalty weights 53
5-6 MPC control penalty weights 53
5-7 Step reference signal for the pitch-up scenario 56
5-8 MPC control penalty weights 58
5-9 Co-simulation parameters 60

7-1 Bottom plane configurations for the flight arena, drone orientations (world frame) and corresponding degrees of freedom (in body frame) for the suspended system 68

“Aircraft have all these unnecessary things like tails and rudders and elevators - not needed. Just gimbal the electric fan. For some weird reason gimbaling motors is normal in rockets and not in aircraft. Why not?”

— *Elon Musk*

Chapter 1

Introduction

This chapter serves as an introduction to Vertical Takeoff and Landing (VTOL) aircraft, the control challenges relating to flight control of such an aircraft and the motivation for this thesis work. First, a short historical context of VTOL aircraft is given in Section 1-1, connecting previous work with recent unmanned efforts. The Avy One, which is subject of this thesis, is presented in Section 1-2. In Section 1-3 the control solutions are laid out. Finally, the structure of the thesis is outlined in Section 1-4.

1-1 Historical context

VTOL aircraft are a type of aircraft that combines the flight envelopes of helicopters and regular jets, integrating hovering capability with long-range flight. This hybrid design eliminates the need for lengthy runways, because the aircraft can take off and land on any unprepared flat surface with the required minimal dimensions.

Although the first VTOL designs were drawn as early as 1843, it would take until the development of jet engines before a sufficient power to weight ratio could be obtained to have liftoff. In the decades after World War II, a number of designs was flown, with varying degrees of success [1]. In those days, the majority of research and innovation in these types of aircraft was driven by military demand, as airfields and landing strips that are damaged during conflict could otherwise limit the much needed transportation capabilities. As of today, only four designs culminated in serial production: the English Hawker Harrier Jump Jet (1967), the Soviet Yakovlev Yak-38 (1970), the American Bell Boeing V-22 Osprey (1989) and lastly, the grossly delayed and over budget F-35B (2015) from American Lockheed Martin Aeronautics (Figure 1-1).

More recently, VTOL has caught the eye of private institutions that are considering commercial applications in data collection, personal transport and (urban) deliveries. With the low cost and miniaturization of electric components and the infancy stage of official regulations, the unmanned industry has seen an increase in various VTOL Unmanned Aerial Vehicle (UAV)'s. There have been successful flights of unmanned VTOL systems of different

Figure 1-1: From left to right: Harrier Jump Jet, Yak-38, V-22 Osprey and F-35B

designs, such as tiltrotors (Quantum Systems TRON, 2014), tiltwings (Lilium, 2017), dual systems (ALTI Transition, 2016) and tailsitters (Atmos Marlyn, 2017). For a full explanation and other examples for these categories, please consult [2, 3]. The unmanned system which is the subject of this thesis, is introduced next in Section 1-2.

1-2 Avy One

Another new company entering the VTOL UAV field is the startup Avy, where work for this thesis was performed for dealing with the control challenges that arise in stabilizing a tiltrotor drone with four independently tilting rotors. The motivation for this design was to develop an unmanned system for humanitarian purposes, such as medicine delivery, wildlife monitoring or rescue operations. The envisioned control system needs to be future-proof, easily adapted in case the components or aerodynamical design have to change, and an be all-in-one solution for the full flight envelope.

A physical demonstration prototype, called the Avy One, is shown on the cover page and rendered in Figure 1-2. Apart from the forward tilting rotors and joined wing design for higher flight efficiency [4], other features include an electrical propulsion system and the possibility to 'swap' payloads to adapt the airframe to different missions. This also increases the direct applicability of the system to an ever-changing market as a 'one design fits most'-type frame can used for different customer requests.

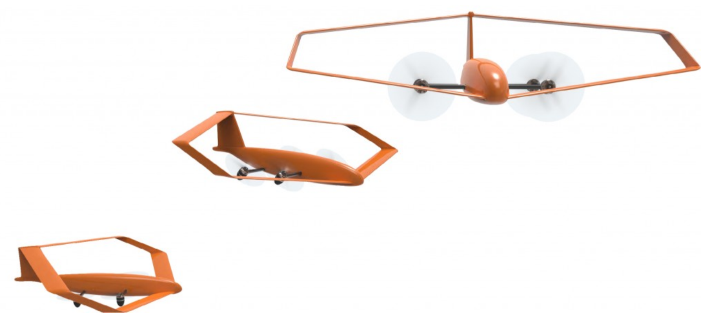


Figure 1-2: A render of the preliminary Avy One design

Some of the control challenges are immediately visible upon inspecting the render of Figure 1-2. There is the multivariable aspect of assigning inputs to 8 control channels - 4 rotor speeds and 4 tilt angles -, the non-linear aspect of the variation of rotor thrust and torque directions and the aerodynamic influences of joined wings and rotor wakes. Techniques to deal with these control challenges are discussed in Section 1-3.

1-3 Previous control solutions

Classically, cascaded control, comprising inner and outer Proportional Integral Derivative (PID) loops is applied in automatic flight control. For example, flap angle or engine speed controllers are lower level systems that receive prescribed input from a higher level navigation controller. The output of these systems are forces, which are used for position and attitude control - the so-called outer loop. The separation of these loops is based on the principle of timescale separation: if the inner loop dynamics are regulated fast enough, the outer loop control, comprising slower dynamics, can treat the inner closed loop as a simple stabilized system.

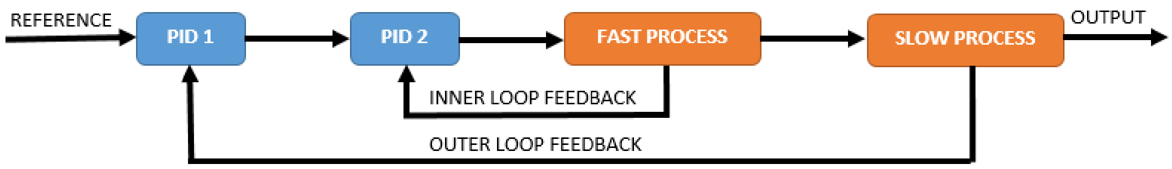


Figure 1-3: Classical Cascaded Control with Inner and Outer loops

However, limitations of classical PID control, not necessarily in cascaded form, are that the controllers are unaware of physical limits, and are tuned to a specific linear system description.

In reality, state and control limits need to be respected, such as altitude and throttle constraints. Hitting control limits in aggressive flight maneuvers or control failure scenarios causes a simple PID controller integrator to windup, slowing the correction during overshoot. Additional tuning is required to prevent windup scenarios [5].

The linear system description is also not generally applicable over the full flight envelope of a VTOL aircraft. The transition of lift from the vertical propulsion system to the wings introduces nonlinear changes to the system, requiring different input-output behaviour from an Flight Control System (FCS) and/or human operator. Tilting rotors add further nonlinearities as the control effectiveness in different degrees of freedom is highly affected by the direction of thrust.

In the case of PID or other linear controllers, it is a tedious work of interpolating and switching between differently tuned local controllers to stabilize the system around a sufficient amount of operating points to cover the flight envelope. Additionally, well-tuned controllers for possible failure modes need to be generated, such as engine failure or stuck control surfaces. This technique of controller switching is called gain scheduling, and has for example successfully been applied to F-16 flight control by switching between controllers based on Mach number and/or altitude [6].

Few details are available on how control systems deal with the additional degrees of freedom in full-scale VTOL tilt-rotors, as many previous efforts are military and commercial in nature. Integrated solutions in drones in academic research include the aforementioned PID-based gain scheduling [7], optimal control-based techniques such as Linear Quadratic Regulator (LQR) control [8], Model Predictive Control (MPC) [9] and even some more exotic setups such as neural networks in combination with model inversion [10]

Overall, two general cases are possible.

The nacelle and/or rotor tilt angles can be...

> ...obtained from a human controller, who commands the tilt angles by hand. The desired angles are then fed to an angle controller and other algorithms that adapt the system response.
>
> ...obtained from a non-human high-level control system in a loop, and given to an angle controller. The reference angles are calculated or optimized given higher level reference signals such as pitch angle, forward speed and vertical speed.

In the case of a medium-sized drone with fast tilting dynamics, and multiple individual angles to be controlled, this mostly argues for the second case, where tilt angle is prescribed by a high-level FCS.

An augmented MPC approach is selected for investigation in this thesis, for which the motivation and mathematics are introduced in Chapter 2.

1-4 Thesis outline

The introduction is concluded with an overview of the remaining thesis chapters.

In Chapter 2 a case is made for MPC as the future-proof flight control technique of choice. An extension to the classical implementation of MPC is proposed to deal with non-linear system changes, by varying the prediction model based on not only the current state (adaptive MPC), but also on previously predicted states. This prediction model variation attempts to approach fully non-linear implementations of MPC while retaining the computational efficiency of Quadratic Programming (QP) optimization.

Chapter 3 introduces and describes the physical models required to describe the flight dynamics of a simplified tiltrotor system. It also contains results from the experimental setups that were designed and used to acquire model parameters. The chapter concludes with a 21-state state space description of the experimental system and an outlook on further experimental work.

Chapter 4 documents the experimental system. First, the external hardware setup is presented, along with the connections and code that interface with the virtual simulator. Secondly, the Robot Operating System (ROS) implementation of the simulator along with a Graphical User Interface (GUI) and visualization is laid out.

Chapter 5 contains the results of off-line Time-Varying Model Predictive Control (TVMPC) flight tests in different flight scenarios, in comparison with fixed-model and adaptive MPC. On the real-time co-simulation setup, simple PID control was implemented to validate the framework, providing more opportunities for research.

Chapter 6 concludes the thesis and Chapter 7 provides ideas and recommendations for further research.

The thesis also contains a number of Appendices which hold information on mathematical modeling, as well as a list of symbols and a glossary.

Chapter 2

Time-varying MPC

In this chapter, a case is made for a model-based flight control technique called Model Predictive Control (MPC) and its ideal, but computationally heavy fully non-linear implementation (Section 2-1). Finally, a middle ground between a static linear MPC and fully non-linear MPC controller is presented as Time-Varying Model Predictive Control (TVMPC) (Section 2-2) and mathematically described (Section 2-3).

2-1 The case for model-based control

It is envisioned that over the long term, aircraft control will switch to model-based, optimization-type methods, that incorporate a mathematical description of the system dynamics, input, state and output constraints and provide more fault tolerance [11]. Such innovation in manned systems is often slow due to regulations and extensive testing procedures, but the quick-moving drone industry has repeatedly proven be a suitable test bed for these types of control [12]. The challenge here is to adhere to weight and power limits while at the same time being able to compute the optimized control decision in time.

These contradictory requirements are nowadays already being met by a small number of minicomputers, which is set to increase in the years to come. They are suited to be carried on board small unmanned systems and capable of demanding computational tasks such as vision processing. For this thesis, the NVIDIA Jetson TX2 was purchased, a system that has previously been applied in unmanned systems successfully [13, 14].

The proposed optimization-type control technique is MPC, originating in slow-moving process control. MPC, in the general sense, employs a multivariable system description in state space form to generate step-ahead predictions for the state trajectory, and attempts to match this prediction with a desired trajectory by optimizing the input signal, subject to any number of constraints on input, state and output using a quadratic penalty. In the classical sense, the system description is Linear Time Invariant (LTI) and the constraints are linear matrix inequalities, allowing for direct computation of the predicted trajectory and optimization gradients. The resulting Quadratic Programming (QP) optimization problem can hence be solved efficiently. The benefits of MPC are a near-perfect overlap with the main limitations of Proportional Integral Derivative (PID): it is multivariable, predictive, takes coupling effects into account, and is designed to handle constraints.

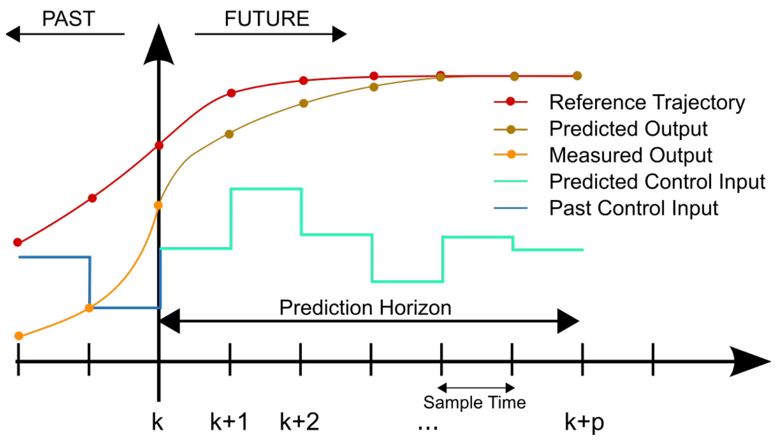


Figure 2-1: Principle of Model Predictive Control in a discrete scheme

Integrating MPC by varying the model descriptions in time is a straightforward solution to deal with the changing system dynamics of a non-linear system. It can be implemented in various ways.

A first extension is to switch between different locally linearized models (which can already be precomputed) based on the current state. This is the Gain-scheduling equivalent of MPC.

A second extension of LTI-MPC is the use of a locally linearized LTI model computed in real time at every time step, given current state information. This technique has been introduced as Adaptive MPC, and eliminates the need to pre-specify the different models [15].

Alternatively, the use of an exogenous (state independent) parameter to select LTI models for control is called Linear Parameter Varying (LPV) control. Feasibility and stability of LPV-MPC has been proven in [16], under conditions similar to LTI-MPC in the case of imperfect models [17].

A second-order time variation effect however, namely the variation of system dynamics and constraints over the prediction horizon is still not taken into account by either extension. It is perfectly possible, that a model linearized around the current state, significantly deviates from a model linearized around a predicted state.

Imagine a lane control system for an automated vehicle in a cornering maneuver, which predicts that within the time horizon, the vehicle tire will hit an icy surface. The next time step, it will change the model used at the predicted time of reaching the icy surface by decreasing the surface friction parameter. Using the new prediction model, it arrives at a new optimized steering command, possibly increasing the steering angle before hitting the icy patch to be able to make the corner. The variation of the prediction model may be a better solution than keeping the high-friction model for the full horizon until parameter changes are detected.

2-2 TVMPC as a middle ground

To integrate this second-order effect, future models that are likely to be visited during the closed-loop evolution of the system need to be determined, either during optimization or pre-optimization.

During optimization Recalculating future models during trajectory optimization steps, which is significantly more often than control steps, quickly launches the control engineer into the field of Non-linear Model Predictive Control (NMPC). This results in optimization routines that are non-quadratic, and goes hand in hand with exponential and unpredictable increases in computational effort. Especially in the time-critical field of flight control, the number of computations needed to obtain a stabilizing control input needs to be limited.

Fortunately, there are other ways to add prediction model variation to an MPC controller while maintaining the QP structure.

Pre-optimization Should the variation of the prediction model be known pre-optimization, supplying the MPC controller with precomputed plant models significantly improves the system response and still results in a QP [15].

In this thesis, the prediction model is varied based on the trajectory prediction result of the previous time step. Linearization points are selected along the trajectory and used to create new LTI prediction models in between these points. It is assumed that with sufficiently accurate models, at least for a short time horizon, the previous state trajectory prediction is close to the real future trajectory in a closed-loop setting and may be used to anticipate model variation.

The main research question is whether such an implementation of adaptive MPC is stable in practice, and if so, how many and which linearization points along the horizon suitably need to be selected.

2-3 From model to model-based controller

2-3-1 Linearizing a non-linear dynamical system

As the controller is model-based, the controller design starts with a mathematical description of the full non-linear dynamical system. For the sake of defining the research scope, the system description is limited to systems for which the state derivative can be described by a continuously differentiable function in the state and control vectors, as

$$\dot{x} = f(x, u), \tag{2-1}$$

where x is a state vector of dimension N_S, u is control input vector of dimension N_C, $\dot{x}$ is the first derivative of system state x with respect to time, and $f(.,.)$ is a state evolution function continuously differentiable in the elements of x and control vector u.

It is common to approach such non-linear system descriptions by their linearization, which are generally a good approximation for short time intervals, or, if $f(.,.)$ is only weakly dependent on x and u, for longer periods of time. By a first-order Taylor expansion, we can obtain

$$\dot{x} = f(x_L, u_L) + \frac{\partial f(x,u)}{\partial x}\Big|_{\substack{x=x_L \\ u=u_L}}(x - x_L) + \frac{\partial f(x_L,u)}{\partial u}\Big|_{\substack{x=x_L \\ u=u_L}}(u - u_L) + \mathcal{O}(x^2, u^2, xu), \tag{2-2}$$

where x_L and u_L are linearization points of the state and control vectors, respectively, and $\mathcal{O}$ is the error term with elements of second order and higher. Ignoring the error term, the notation can be simplified to

$$\dot{x} = A_C x + B_C u + C_C, \tag{2-3}$$

with

$$\begin{aligned} A_C &= \frac{\partial f(x,u)}{\partial x}\Big|_{\substack{x=x_L \\ u=u_L}} \\ B_C &= \frac{\partial f(x_L,u)}{\partial u}\Big|_{\substack{x=x_L \\ u=u_L}} \\ C_C &= f(x_L, u_L) - A_C x_L - B_C u_L. \end{aligned}$$

The terms A_C and B_C are commonly referred to as continuous-time state matrix and continuous-time control matrix, and C_C can be considered an offset vector including non-equilibrium effects.

2-3-2 Discretizing a continuous state-space system

Implementation in a digital controller requires a discrete-time system description. This is commonly done by difference equations for the current state x_k depending on previous state and control x_{k-1} and u_{k-1} at a specific time k, where the sampling time between times k and $k-1$ is equal to τ. Using the derivation of Appendix A-1-1, we can write

$$x_{k+1} = A_D x_k + B_D u_k + C_D, \tag{2-4}$$

with

$$\begin{aligned} A_D &= e^{A_C \tau}, \\ B_D &= A_C^{-1}(A_D - I)B_C, \\ C_D &= A_C^{-1}(A_D - I)C_C, \end{aligned} \tag{2-5}$$

where A_D, B_D and C_D are discrete-time equivalents of the state matrix, control matrix and offset vector. If A_C is non-invertible, a Taylor approximation can be used (Appendix A-1-1).

2-3-3 Writing the state update equation in matrix form

Now that we have established the discrete-time state update equation for a single timestep for a linearized system, we can extend the equation to a multi-timestep update. This facilitates computation and optimization in the MPC setting, as we are continuously calculating and optimizing multiple step ahead predictions. From this point onwards, it is understood that all state and control matrices refer to discrete-time matrices corresponding to a predetermined sampling time τ.

Let a system be updated using different linearized state space models at every time step, generating the following sequence of equations:

$$\begin{aligned} x_{k+1} &= C_{L_1} + A_{L_1} x_k + B_{L_1} u_k \\ x_{k+2} &= C_{L_2} + A_{L_2} x_{k+1} + B_{L_2} u_{k+1} \\ x_{k+3} &= C_{L_3} + A_{L_3} x_{k+2} + B_{L_3} u_{k+2} \\ x_{k+4} &= \dots \end{aligned} \tag{2-6}$$

where A_{L_i}, B_{L_i} and C_{L_i} represent a sequence of state space models.

The state evolution can be expressed using the initial state and control vector by replacing all x_{k+i} for $i \in \mathcal{N}, i \geq 2$. For the first two substitutions, this becomes

$$\begin{aligned} x_{k+2} &= C_{L_2} + A_{L_2} C_{L_1} + A_{L_2} A_{L_1} x_k + A_{L_2} B_{L_1} u_k + B_{L_2} u_{k+1}, \\ x_{k+3} &= C_{L_3} + A_{L_3} C_{L_2} + A_{L_3} A_{L_2} C_{L_1} + A_{L_3} A_{L_2} A_{L_1} x_k + A_{L_3} A_{L_2} B_{L_1} u_k + A_{L_3} B_{L_2} u_{k+1} + B_{L_3} u_{k+2}. \end{aligned} \tag{2-7}$$

A linear pattern starts to appear. By defining (for $k = 0$) the vectors and matrices

$$\tilde{X} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_{N_P} \end{bmatrix}, \quad \tilde{U} = \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ u_{N_P-1} \end{bmatrix}, \quad \tilde{C}_L = \begin{bmatrix} C_{L_1} + A_{L_1} x_k \\ C_{L_2} \\ \dots \\ C_{L_{N_P}} \end{bmatrix},$$

$$\tilde{A} = \begin{bmatrix} I & 0 & 0 & \dots & 0 \\ A_{L_2} & I & 0 & \dots & 0 \\ A_{L_3} A_{L_2} & A_{L_3} & I & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ A_{L_{N_P}} \dots A_{L_2} & A_{L_{N_P}} \dots A_{L_3} & A_{L_{N_P}} \dots A_{L_4} & \dots & I \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} B_{L_1} & 0 & \dots & 0 \\ 0 & B_{L_2} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & B_{L_{N_P}} \end{bmatrix},$$

with N_P being the prediction horizon, $\tilde{X}$ the stacked vector of predicted states, $\tilde{U}$ the stacked vector of future control inputs, $\tilde{C}$ the stacked vector of offset vectors, $\tilde{A}$ the stacked state prediction matrix and $\tilde{B}$ the stacked control prediction matrix, the total state prediction can be written in a condensed fashion, as

$$\tilde{X} = \tilde{A}\tilde{C}_L + \tilde{A}\tilde{B}\tilde{U}. \tag{2-8}$$

It can immediately be observed that the prediction is still a linear function of the control inputs, stacked in vector $\tilde{U}$.

Identical models

Special attention is paid to the case where the same model is used for multiple horizon steps. This reduces the number of unique entries in the matrix, speeding up the problem formulation as the matrix can be filled more efficiently. The copying of entries is computationally less costly than recalculating a matrix multiplication.

Suppose a 6-step ahead prediction is carried out using only two models, where the model is switched exactly halfway along the prediction. We would obtain the following $\tilde{A}$ matrix,

$$\tilde{A} = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ A_{L_1} & I & 0 & 0 & 0 & 0 \\ A_{L_1}^2 & A_{L_1} & I & 0 & 0 & 0 \\ A_{L_2}A_{L_1}^2 & A_{L_2}A_{L_1} & A_{L_2} & I & 0 & 0 \\ A_{L_2}^2A_{L_1}^2 & A_{L_2}^2A_{L_1} & A_{L_2}^2 & A_{L_2} & I & 0 \\ A_{L_2}^3A_{L_1}^2 & A_{L_2}^3A_{L_1} & A_{L_2}^3 & A_{L_2}^2 & A_{L_2} & I \end{bmatrix},$$

where the number of unique sub-matrices (not including zeros and identity) is reduced from 15 to 11. In case of a 12-step ahead prediction with a model switch halfway, the number of unique sub-matrices is reduced from 66 to 41. A formula to determine the total number of matrices to compute, depending on the number of models and their respective horizons, is determined in Appendix A-1-2. The ratio of unique entries to total matrix entries can be shown to approach 0.5 when the model switch is placed exactly halfway (see Appendix A-1-2), but is still quadratic in N_P.

2-3-4 MPC control in matrix form

Now, following traditional MPC notation, let Q_k and R_k respectively be a state weighting matrix and control weighting matrix at time k, and consider simply bounded linear and matrix constraints. Let $\tilde{X}_{\text{ref}}$ be the stacked vector of state reference values and let $\tilde{Q}$ and $\tilde{R}$ be defined as the block diagonalizations of the Q_k and R_k, respectively, over the prediction horizon. Then the MPC problem is formulated as a quadratic programming problem in the form of

$$\min_{\tilde{X},\tilde{U}} (\tilde{X} - \tilde{X}_{\text{ref}})^T \tilde{Q} (\tilde{X} - \tilde{X}_{\text{ref}}) + \tilde{U}^T \tilde{R} \tilde{U}, \text{ subject to}$$

$$\tilde{X} = \tilde{A}\tilde{C}_L + \tilde{A}\tilde{B}\tilde{U}.$$

$$\begin{aligned} \tilde{h}_{xl} &\leq \tilde{H}_x \tilde{X} \leq \tilde{h}_{xu} \\ \tilde{s}_{xl} &\leq \tilde{X} \leq \tilde{s}_{xu} \\ \tilde{h}_{ul} &\leq \tilde{H}_u \tilde{U} \leq \tilde{h}_{uu} \\ \tilde{s}_{ul} &\leq \tilde{U} \leq \tilde{s}_{uu} \end{aligned}$$

with

$$\tilde{X}_{\text{ref}} = \begin{bmatrix} x_{\text{ref}_1} \\ x_{\text{ref}_2} \\ ... \\ x_{\text{ref}_{N_P}} \end{bmatrix}, \quad \tilde{h}_{xl} = \begin{bmatrix} h_{xl_1} \\ h_{xl_2} \\ ... \\ h_{xl_{N_P}} \end{bmatrix}, \quad \tilde{h}_{xu} = \begin{bmatrix} h_{xu_1} \\ h_{xu_2} \\ ... \\ h_{xu_{N_P}} \end{bmatrix}, \quad \tilde{h}_{ul} = \begin{bmatrix} h_{ul_1} \\ h_{ul_2} \\ ... \\ h_{ul_{N_P}} \end{bmatrix}, \quad \tilde{h}_{uu} = \begin{bmatrix} h_{uu_1} \\ h_{uu_2} \\ ... \\ h_{uu_{N_P}} \end{bmatrix},$$

$$\tilde{s}_{xl} = \begin{bmatrix} s_{xl_1} \\ s_{xl_2} \\ ... \\ s_{xl_{N_P}} \end{bmatrix}, \quad \tilde{s}_{xu} = \begin{bmatrix} s_{xu_1} \\ s_{xu_2} \\ ... \\ s_{xu_{N_P}} \end{bmatrix}, \quad \tilde{s}_{ul} = \begin{bmatrix} s_{ul_1} \\ s_{ul_2} \\ ... \\ s_{ul_{N_P}} \end{bmatrix}, \quad \tilde{s}_{uu} = \begin{bmatrix} s_{uu_1} \\ s_{uu_2} \\ ... \\ s_{uu_{N_P}} \end{bmatrix},$$

$$\tilde{Q} = \begin{bmatrix} Q_1 & 0 & ... & 0 \\ 0 & Q_2 & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & Q_{N_P} \end{bmatrix}, \tilde{R} = \begin{bmatrix} R_1 & 0 & ... & 0 \\ 0 & R_2 & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & R_{N_P} \end{bmatrix},$$

$$\tilde{H}_x = \begin{bmatrix} H_{x_1} & 0 & ... & 0 \\ 0 & H_{x_2} & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & H_{x_{N_P}} \end{bmatrix}, \tilde{H}_u = \begin{bmatrix} H_{u_0} & 0 & ... & 0 \\ 0 & H_{u_1} & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & H_{u_{N_P-1}} \end{bmatrix},$$

where $\tilde{h}_{xl}$, $\tilde{h}_{xu}$, $\tilde{h}_{ul}$, $\tilde{h}_{uu}$ are stacked lower and upper state and control bounds for matrix constraints defined by stacked matrix constraint matrices $\tilde{H}_x$ and $\tilde{H}_u$ for state and control respectively, and $\tilde{s}_{xl}$, $\tilde{s}_{xu}$, $\tilde{s}_{ul}$, $\tilde{s}_{uu}$ are stacked simple state and control bounds.

2-3-5 QP formulation

The implementation of an MPC controller in an embedded system requires a real-time QP solver. Such solvers exist for various platforms and differ in input formulation, programming language, underlying algorithm and licensing. For this thesis, the solver qpOases [18] was used because of its efficiency, free license, and existing interface with Robot Operating System (ROS).

This solver can not take QP problems with equality constraints, but only of the following form:

$$\min_z \frac{1}{2} z^T H z + z^T g, \text{ subject to}$$
$$c_{AL} \leq A\, z \leq c_{AU},$$
$$c_L \leq z \leq c_U,$$

where z is the optimization vector, H the Hessian, g a bias vector, c_{AL}, c_{AU}, c_L and c_U vector lower and upper bounds and A a matrix constraint.

Use substitutions for the following optimization variable independent vectors:

$$\begin{aligned}\tilde{C} &= \tilde{A}\tilde{C}_L,\\ \tilde{C}_R &= \tilde{C} - \tilde{X}_{\text{ref}}.\end{aligned}$$

Substituting the state update equation in the cost function and constraints, we can derive that

$$\begin{aligned}J &= (\tilde{X} - \tilde{X}_{\text{ref}})^T\tilde{Q}(\tilde{X} - \tilde{X}_{\text{ref}}) + \tilde{U}^T\tilde{R}\tilde{U}\\ &= (\tilde{C}_R + \tilde{A}\tilde{B}\tilde{U})^T\tilde{Q}(\tilde{C}_R + \tilde{A}\tilde{B}\tilde{U}) + \tilde{U}^T\tilde{R}\tilde{U}\\ &= \tilde{C}_R^T\tilde{Q}\tilde{C}_R + \tilde{C}_R^T\tilde{Q}\tilde{A}\tilde{B}\tilde{U} + \tilde{U}^T\tilde{B}^T\tilde{A}^T\tilde{Q}\tilde{C}_R + \tilde{U}^T\tilde{B}^T\tilde{A}^T\tilde{Q}\tilde{A}\tilde{B}\tilde{U} + \tilde{U}^T\tilde{R}\tilde{U}\\ &= \tilde{C}_R^T\tilde{Q}\tilde{C}_R + 2\tilde{U}^T\tilde{B}^T\tilde{A}^T\tilde{Q}\tilde{C}_R + \tilde{U}^T(\tilde{B}^T\tilde{A}^T\tilde{Q}\tilde{A}\tilde{B} + \tilde{R})\tilde{U}\end{aligned}$$

where the first term may be omitted in the optimization, as it is a constant. Rescaling with a factor $\frac{1}{2}$ does not influence the location of the optimum, and hence we obtain for the qpOases formulation that

$$\begin{aligned}H &= \tilde{B}^T\tilde{A}^T\tilde{Q}\tilde{A}\tilde{B} + \tilde{R},\\ g &= \tilde{U}^T\tilde{B}^T\tilde{A}^T\tilde{Q}\tilde{C}_R.\end{aligned}$$

For the constraints, we arrive at

$$\begin{aligned}c_{AL} &= \begin{bmatrix}\tilde{h}_{xl} - \tilde{H}_x\tilde{C}\\ \tilde{s}_{xl} - \tilde{C}\\ \tilde{h}_{ul}\end{bmatrix},\\ c_{AU} &= \begin{bmatrix}\tilde{h}_{xu} - \tilde{H}_x\tilde{C}\\ \tilde{s}_{xu} - \tilde{C}\\ \tilde{h}_{uu}\end{bmatrix},\\ A &= \begin{bmatrix}\tilde{H}_x\tilde{A}\tilde{B}\\ \tilde{A}\tilde{B}\\ \tilde{H}_u\end{bmatrix},\\ c_L &= \tilde{s}_{ul},\\ c_U &= \tilde{s}_{uu}.\end{aligned}$$

It turns out that calculating the Hessian and bias vector are the largest contributors to computational time for the problem setup. It should be noted that a significant speed reduction can be obtained by exploiting the symmetrical structure of the problem, for example by computing only the lower half of a symmetrical matrix, and filling the remaining entries by mirroring the solution. Several functions are implemented to this end in the Eigen [19] library, as well as accelerations for multi-threading.

2-4 Motivating examples

The proposed time-varying MPC controller is compared to non-varying (fixed model) and adaptive implementations on a simple non-linear system with one state variable and one control input.

The dynamical equation governing the system is described by (2-9). Both the state and control dependence are non-linear. We have

$$\dot{x} = x^2 + u^3 + u, \tag{2-9}$$

where x is the state and u the control input. The model is linearized using a sample time of $\tau = 0.025$, and the constraints on state and input of (2-10) are introduced.

$$\begin{aligned} -5 \leq x \leq 5 \\ -2 \leq u \leq 2 \end{aligned} \tag{2-10}$$

All three controllers are first evaluated in a simple step scenario where the system initializes at $x = 1$ and needs to be controlled to $x = 2$. State and control weights are 10 and 0.1 respectively. The control horizon is set at $N_P = 4$. The fixed-model MPC is linearized around $(x_L, u_L) = (2, -1.3)$, which is close to the equilibrium condition around the target state. The adaptive MPC uses the current state and control value, and the time-varying implementation supplements that with previously predicted state and control values over the full horizon. For the first iteration the time-varying controller is initialized with a repetition of the current model.

All three controllers show the capability to stabilize the system (Figure 2-2). The overshoot of the time-varying controller is smaller than that of the adaptive MPC controller.

However, the time-varying controller appears sensitive to the control weighting. When R is tripled to 0.3, it converges to a steady-state value significantly different from the other two and the reference. The future control vectors determined at the end of the simulation for all three implementations show the source of the error.

	u_0	u_1	u_2	u_3
Fixed Model	-1.3821	-1.2446	-1.0471	-0.6913
Adaptive	-1.3802	-1.2628	-1.0837	-0.7338
Time-Varying	-1.3448	-0.2771	-0.2261	-0.1272

Table 2-1: Optimization result at the end of the simulation, for $R = 0.3$

The time-varying model shows a large anticipated decrease in control, resulting in a different prediction model, with reduced control effectiveness for the last time steps. The balance between tracking accuracy and control effort now tips toward reducing control effort at the cost of a steady state error.

The controllers are further compared in a scenario where the system is expected to visit significantly different model regimes, and the controller is able to predict further ahead.

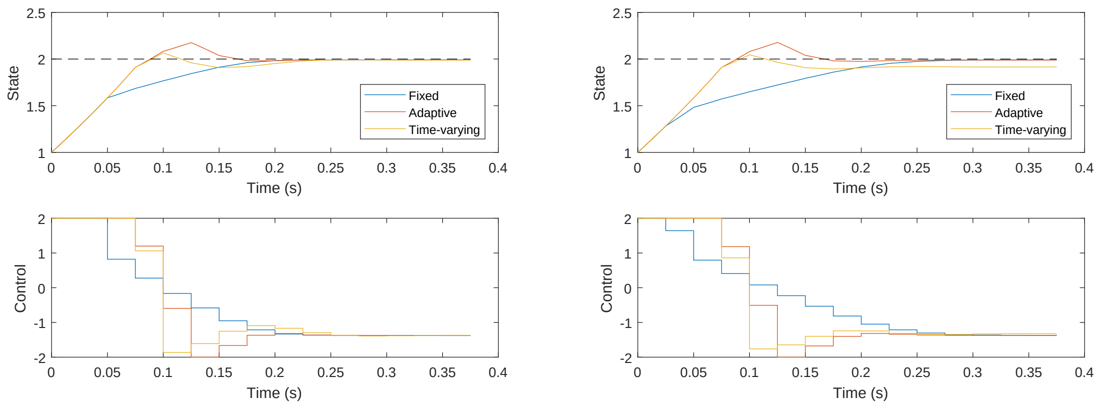


Figure 2-2: Left: Step scenario with $R = 0.1$. Right: Step scenario with $R = 0.3$.

Taking $N_P = 30$, allowing more aggressive control ($-3 \leq u \leq 3$) and using a sine wave reference signal around $x = 0$ with period 0.5 and amplitude 1, the advantage of predicting model changes is directly apparent.

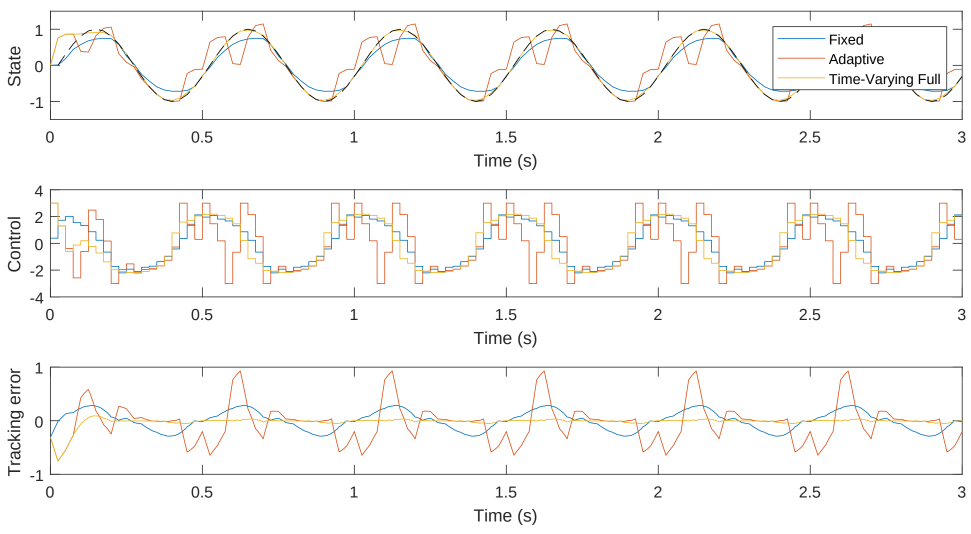


Figure 2-3: Sine wave tracking with $N_P = 30$ and extended control limits for fixed, adaptive and full time-varying MPC

This time, the time-varying MPC controller significantly outperforms the adaptive MPC controller. After a short period needed to converge to the trajectory, it follows the sine wave closely, with an Root-Mean-Square (RMS) tracking error of only 0.0910 over the full period, and 0.0197 after $t = 0.5$ (Table 2-2). The adaptive MPC controller shows adequate tracking behavior only in the region $x < 0$ over the first half of the bottom sine crest, but oscillates heavily when returning to the positive region. Its RMS tracking error equals 0.3522 (0.3582 after $t = 0.5$). The fixed-model controller performs better with an RMS error of 0.1734 after $t = 0.5$ but misses the peaks of the wave signal as it underestimates the control effectiveness.

An additional comparison is made with time-varying controllers where the model becomes constant after a set horizon value. This can be seen as an intermediate step between adaptive MPC and time-varying MPC over the full horizon. Figure 2-4 displays a controller that keeps the model constant after step 3, and a controller that keeps the model constant after step 7, in comparison with the full time-varying controller.

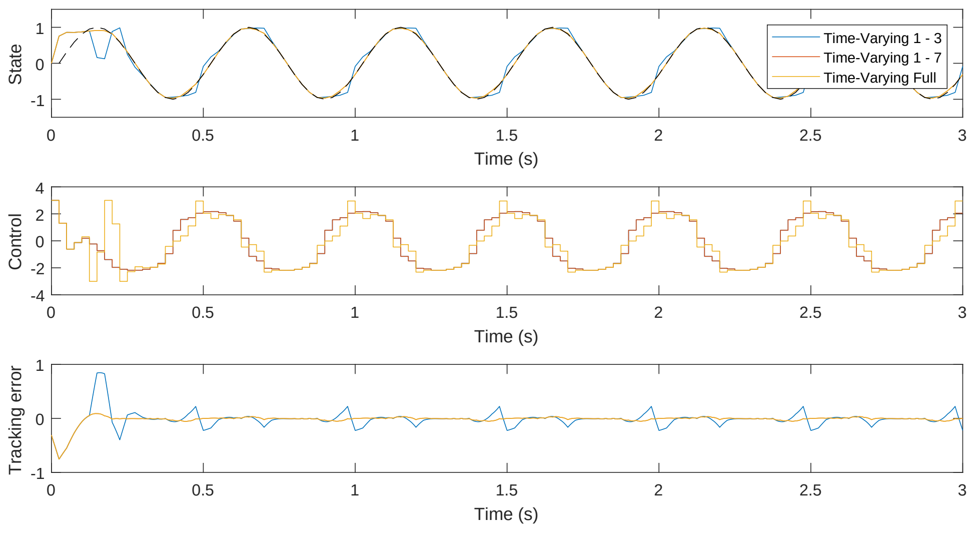


Figure 2-4: Sine wave tracking with $N_P = 30$ and extended control limits for time-varying MPC with reduced model variation

The controller implementing only three models already shows a large RMS tracking error improvement over the adaptive MPC (0.0778 after $t = 0.5$, Table 2-2). Surprisingly, the controller implementing seven models outperforms the full thirty-model time-varying controller by a post-0.5s tracking error reduction of almost 24%.

The full development of the RMS tracking error under model number increases is displayed in Figure 2-5. The seven-model controller shows the smallest tracking error overall, but the performance difference between 6 to 30 models is small.

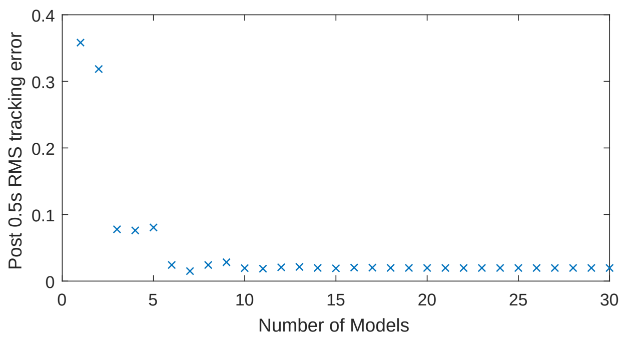


Figure 2-5: Post-0.5s RMS tracking error

Motivated by the examples above, the selection of future linearization models presents itself

MPC controller	RMSE (full data)	RMSE ($t > 0.5$)
Fixed Model	0.1742	0.1734
Adaptive (TV 1-1)	0.3522	0.3582
Time-Varying 1-2	0.3298	0.3185
Time-Varying 1-3	0.1553	0.0778
Time-Varying 1-4	0.1149	0.0761
Time-Varying 1-5	0.1190	0.0805
Time-Varying 1-6	0.0922	0.0242
Time-Varying 1-7	0.0901	0.0150
Time-Varying 1-8	0.0919	0.0242
Time-Varying 1-9	0.0933	0.0283
Time-Varying 1-10	0.0910	0.0195
Time-Varying 1-30	0.0910	0.0197

Table 2-2: RMS tracking errors for the sine wave tracking case

as a non-trivial field of study, depending on tuning parameters, prediction horizon and system description. It shows that a reduced, but relatively small set of additional linear prediction models can result in a significant improvement over non-time-varying MPC.

The thesis continues with the description and results of modeling a multivariable tilt-rotor system. The system description is non-linear and necessarily larger than the one-state system from this section, motivating the need to capture variations with as little additional models as possible, as the computational cost of matrix multiplications grows quadratically with model size.

Chapter 3

Model identification

This chapter introduces and describes the physical models required to describe the flight dynamics of a simplified tilt-rotor system, such as a rotor model (Section 3-1), stepper motor model (Section 3-2), and the full dynamical equations (Section 3-3. This section concludes with a 21-state state space description of the test system and its state and control limits.

In the respective modeling sections, results of the experimental setups that were designed and used to acquire the model parameters are presented. The final section (Section 7), describes unfinished experimental work and a large guided flight experiment which was designed but only partly built, providing opportunities for future research.

3-1 Rotor model

3-1-1 System description

Central to the modeling of the drone is an accurate model for the rotor dynamics. The rotors spin at high speeds, generated by Direct Current (DC) motors and generate thrust and torque forces. The thrust forces and torque forces combined drive the system's dynamical evolution. Higher rotational speeds are coupled to higher forces, but changes in speed introduce torque effects as well. The speeds are in turn controlled by Electronic Speed Controller (ESC), quick low-level Proportional Integral Derivative (PID) controllers that take speed commands and output current to the DC motors. All three components (DC motor, rotor and ESC) are modeled in respective subsections.

To experimentally determine the input-output behavior of the rotor system, a combined Thrust and Torque Test (T3) was built. To facilitate coaxial testing and efficiency analysis in the future, provisions for measurement of individual motor torques, rotational speeds, electrical currents and the combined thrust were made. A render and of the experiment can be seen in Figure 3-13.

DC Motor

At the core of the rotor system lies the DC motor. In the motor, the stator and rotor (or armature) contain windings that are excited by DC current, and the interaction of both magnetic fields of the stator and rotor results in rotation of the rotor. Most commonly, the stator windings are replaced by permanent magnets, and hence the DC motor is 'armature controlled'. This type of design is cheaper than designs where the stator has coil windings, but the design suffers from large currents that may damage the insulation material.

In armature controlled motors, the magnetic field may be assumed constant [20]. Under that assumption, the motor torque becomes proportional to the current by

$$T_{M,i} = K_t i_i, \tag{3-1}$$

where $T_{M,i}$ is the motor torque of motor i, K_t is the motor torque constant (assumed equal for all motors used) and i_i is the current through motor i. From Faraday's law, the back Electromagnetic Force (EMF) by the rotating armature is proportional to the angular speed by

$$e_i = K_e \omega_i = K_e \dot{\theta}_i, \tag{3-2}$$

where e_i is the back EMF of motor i, K_e is the EMF constant (assumed equal), ω_i is the rotor speed, also denoted as the first derivative of the angle θ_i. In SI units, K_t and K_e are equal and will from hereon be replaced by K.

From Kirchhoff's law and Newton's second law we arrive at the system of linear equations of

$$\begin{aligned} J_R \ddot{\theta}_i + b_f \dot{\theta}_i &= K i_i, &(3\text{-}3)\\ L \dot{i}_i + R_i i_i &= V_i - K\theta_i &(3\text{-}4)\end{aligned}$$

where J_R is the rotor inertia, b_f is the motor friction coefficient, L is the motor inductance (all assumed equal), R_i is the motor resistance and V_i is the motor voltage.

Assuming the voltage as input, and as the back EMF and hence rotor speed are measured by the Rounds Per Minute (RPM) sensors ((3-2)) the complete system can be written in state-space form as

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix}\dot{\theta}_i\\ i_i\end{bmatrix} &= \begin{bmatrix}-\frac{b_f}{J_R} & \frac{K}{J_R}\\ -\frac{K}{L} & -\frac{R_i}{L}\end{bmatrix}\begin{bmatrix}\dot{\theta}_i\\ i_i\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{L}\end{bmatrix} V_i &(3\text{-}5)\\ y &= \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}\dot{\theta}_i\\ i_i\end{bmatrix}, &(3\text{-}6)\end{aligned}$$

where system output y is the rotor speed i. A classical analysis of observability and controllability according to R. Kalman [21] reveals that the system is controllable, observable and also passively stable.

For this thesis, the Multistar 5008 KV-330 motor (Figure 3-1) was used, which has a K-value of 330 RPM/V.

The most important takeaway here however, is not the exact physical system description in motor parameters, but that the system follows a linear description. From here, the the coupling to the rotor introduces aerodynamical effects which are in general non-linear.

Rotor

The shafts of the fast-spinning DC motors are rigidly connected to the aircraft rotors, and are used to displace air to generate thrust and torque. The faster the shaft spins, the faster the rotor turns and the more air is displaced. For the sake of simplicity, only a single-rotor model out of ground effect is considered. The ground effect is an increase in thrust and torque when the rotor spins closely above a surface, leading to a cushion-like effect. However, in normal flight, the correction factor associated with the ground effect is already smaller than 1% when the flight altitude is 2.5 times larger than the rotor radius [22].

Thrust Using momentum theory [23], the thrust $T_{R,i}$ generated by a spinning rotor in steady state in free air may be modeled as

$$T_{R,i} = C_T \rho A_r r^2 \omega_i^2, \tag{3-7}$$

where A_r represents the rotor disk area, r is the rotor radius, ρ is the density of air, ω_i is the angular velocity of rotor i and C_T is the thrust coefficient. The thrust coefficient depends on several parameters of the rotor design, such as blade pitch and curvature. In practice, a combined thrust coefficient c_T is often used for modeling of a specific rotor [24], so that Eq. (3-7) reduces to

$$T_{R,i} = c_T \omega_i^2. \tag{3-8}$$

Torque Similarly, a single rotor reaction torque can be modeled using a combined coefficient as

$$Q_{R,i} = c_Q \omega^2, \tag{3-9}$$

where $Q_{R,i}$ is the rotor torque and c_Q is the combined torque coefficient, again dependent on rotor geometry and air density.

The most important takeaway here is that in steady state, we can model thrust and torque by quadratic coefficients in the rotor speed. A dynamical system is however not always in steady state. From the DC motor system, we know that for speed changes, additional torque needs to be exerted on the rotor. Hence, for an acceleration motion, we can expect thrust to follow the quadratic relation, but torque to consist of a motor component and a speed-related component.

For this thesis, the carbon fiber Quanum 18x5.5 propeller was used (Figure 3-1).

ESC system

The rotor speed control is largely taken care of by a dedicated ESC, which takes speed reference commands in the form of Pulse Width Modulation (PWM) signals and controls the motor current to obtain the desired speed without overshoot. In practice, most of these systems are essentially well-tuned, extremely fast PID controllers. For this thesis, the 30A AfroESC was used (Figure 3-1), which operates at a frequency of 400 Hz.

Figure 3-1: From left to right: Multistar 5008 KV330, 30A AfroESC and Quanum 18x5.5 propeller

ESC controllers, when turned on, are by default in a so-called "unarmed" state, where they do not accept command inputs for safety reasons. Sending a constant command signal of a specific frequency (the "arm" signal) slightly below the minimum speed reference for a short duration activates the controller. From that point onwards, the motor speed can be regulated. Loss of signal or sending a throttle signal below the minimum speed reference will disarm the ESC again.

Although PID controllers constitute linear systems, current commercial solutions like AfroESC have evolved to implement multiple non-linear advanced features, such as rotor braking, saturation limits, and throttle curves. Rotor braking aims to assist in slowing down the rotor using the coils in addition to normal aerodynamic drag. Saturation limits help prevent extremely high currents, and throttle curves can assist the user in achieving the desired responsiveness and flight behaviour, and are essentially a mapping between input signal frequency and desired rotational speed.

3-1-2 Four-parameter combined model

As the total subsystem combination of motor, rotor and controller can definitely not be considered linear, it was decided to model the system from angular speed reference to output forces as a single entity, taking one throttle input command and generating two outputs. The corresponding ESC-specific PWM frequency to a reference angular speed is computed separately by inverting a measured throttle curve.

The first output of the system is the angular speed, which does not exhibit overshoot behavior and is modeled according to a first-order response using the time constant of the system. In the real-time setup, the angular speed can be measured using RPM sensors. The thrust force can be obtained by squaring the result and multiplying it with c_T.

The second output is the angular speed with an additional overshooting effect, as a result of the rapid acceleration and deceleration of the rotating mass. It is modeled according to a same time constant of the thrust rise time and an overshoot parameter influencing the peak height of the response. The motivation behind this state space variable is that it captures the torque effect associated with a non-overshooting speed increase within the same time frame, and is easily squared and multiplied with the torque coefficient c_Q, resulting in a model with a minimum number of parameters. Further motivation comes from fits with experimental data, showing good correspondence with observed measurements (Section 3-1-6. This variable however, is not directly observable, but is estimated using a Luenberger observer, which is further described in Section 4-2-2.

The proposed system model from angular speed reference to forces is represented by

$$\begin{aligned}\begin{bmatrix}\dot{\omega}_T \\ \dot{\omega}_Q\end{bmatrix} &= \begin{bmatrix}-a & 0 \\ -b & -a\end{bmatrix}\begin{bmatrix}\omega_T \\ \omega_Q\end{bmatrix} + \begin{bmatrix}a \\ a+b\end{bmatrix}u \\ F_R &= c_T\omega_T^2 \\ Q_R &= c_Q\omega_Q^2\end{aligned} \tag{3-10}$$

where ω_T is the rotational speed, ω_Q is the rotational speed with overshooting effect and a and b are model parameters. The effect of these parameters can be seen in Fig. 3-2, with a being the inverse of the time constant of the first order system, and b significantly influencing the peak height of the overshoot.

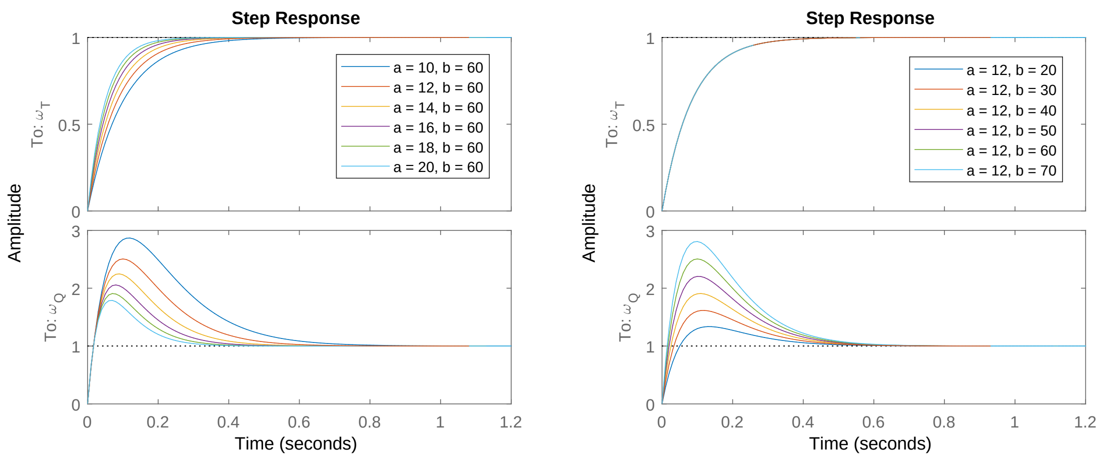


Figure 3-2: Left: Effect of parameter a on the the step response. Right: Effect of parameter b on the the step response.

3-1-3 RPM sensor calibration

Experimental Set-up

At the core of the modeling and co-simulation is the time-series measurement of rotational speed. It is necessary to determine the minimum and maximum motor speeds (and their corresponding input signals), the steady-state thrust and torque coefficients, and the dynamic response behaviour of the system.

The sensors used are simple Eagle Tree RPM sensors for brushless motors. They are lightweight sensors with a measurement range up to 8000 rpm, and only need to be calibrated for the amount of poles in the brushless motor (4). The sensor wires are attached to any two of the three motor wires, which the sensor uses to determine voltage switches associated with the movement of the magnets inside the motor.

Figure 3-3: Eagle Tree RPM sensor

The calibration is carried out using a small striped marker and a stroboscope. The marker is stuck to the brushless motor and the motor is spun from minimum to maximum speed in small steps. At every speed setting, the stroboscope is adjusted so that the marker appears to stand still. The oscilloscope frequency and timing between sensor spikes are logged to analyze the sensor linearity.

For the real-time implementation, the timing signal is filtered and analyzed for zero-speed detection, which occurs when the sensor stops sending spikes. This is motivated by the hardware-in-the-loop simulation where the system always starts from zero-speed, and possible future research into fault-tolerant control. The implementation and code is further described in Section 4-2-2.

Results

The sensor timing values are close to the expected curve for a 4-pole motor. However, the observed time differences between expected and measured timing values do not seem to follow the same relation (See left figure of Figure 3-4). The difference values appear to converge to a 40-45 μs difference, which may be attributed to the internal sensor algorithm or delays in communication or measurement by the Arduino.

In the right graph of Fig. 3-4, the timing values are converted to RPM. The same small difference is observed. Should the RPM sensor be used in a stand-alone fashion, a correction factor of 96% should be used. In this thesis, the same RPM sensor is used for mapping in all the other experiments, so correction factors need not be applied.

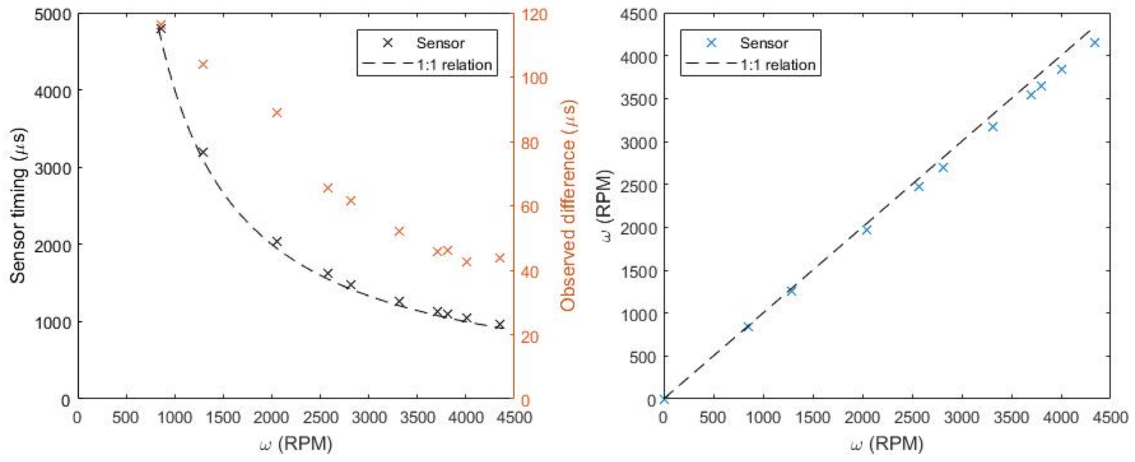


Figure 3-4: Left: Sensor timing results and difference with expectation. Right: Sensor linearity analysis

3-1-4 Throttle curve measurement

Experimental Set-up

From the between-spike timings of the sensor, the RPM values of the motor are read. For every ESC, the throttle curves (as described in Section 3-1-1) are measured by arming the ESC, increasing the input frequency in small steps, and recording the corresponding RPM values.

Results

All ESC's were previously used in flying systems, with the front left (FL), front right (FR) and rear right (RR) all having been used in one system, and the rear left (RR) in a different system.

First, the arming signals, and minimum and maximum effective signal inputs for every ESC were determined. These are mapped to PWM frequencies by an external board via Inter-Integrated Circuit (I2C), where the full duty cycle corresponds to a signal of 4095. The results are displayed in Table 3-1.

ESC	Arming Signal	Min Throttle	Max Throttle
Front Left (FL)	1300	1485	2700
Front Right (FR)	2000	2011	3400
Rear Left (RL)	2000	2130	2650
Rear Right (RR)	1750	1810	3350

Table 3-1: Arming frequency and signal limits for all ESC's

The difference in calibration settings is explained by the practical need to minimize signal interference. Having differently calibrated ESC's avoids accidentally arming the wrong motor, or signal interference when similar throttle values are sent to different motors.

ESC	x^3	x^2	x	1	R^2
Front Left (FL)	$3.2262 \cdot 10^{-6}$	-0.0225	53.5920	$-3.9902 \cdot 10^4$	0.9998
Front Right (FR)	$2.2542 \cdot 10^{-6}$	-0.0200	60.5950	$-5.8576 \cdot 10^4$	0.9997
Rear Left (RL)	$3.9594 \cdot 10^{-6}$	-0.2959	740.2501	$-6.1647 \cdot 10^5$	0.9994
Rear Right (RR)	$1.6299 \cdot 10^{-6}$	-0.0141	41.8915	$-3.8527 \cdot 10^4$	0.9998

Table 3-2: PWM to RPM models

Consequently, a full PWM to RPM mapping is created, by increasing the throttle in steps from minimum to maximum and recording the RPM value reported by the sensor. The PWM board takes inputs in the range of 0 to 4096, where 0 corresponds with a 0% duty cycle and 4096 with a 100% duty cycle. The results are visible in Figure 3-5.

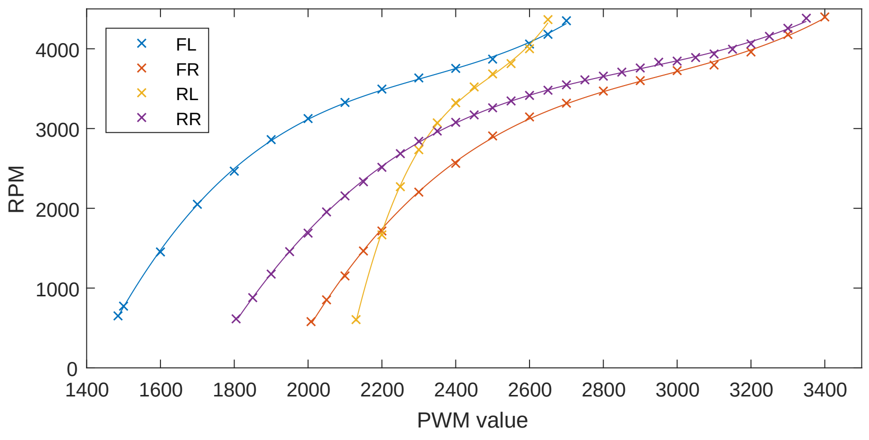


Figure 3-5: PWM board input values to RPM curves

The results show a curved throttle mapping, which is motivated by expectations of a user during manual flight and the quadratic effect of rotational speed on thrust. By implementing a throttle curve that develops like a square root, the total throttle-to-thrust map is linearized. Manual flight for an inexperienced user therefore becomes more intuitive, as the absolute thrust increase per throttle step is similar in all regions of flight. The upward slope at the end of the graph can be motivated by an 'emergency' throttle command, where a high-powered response is needed for an aggressive maneuver.

The throttle curves are modeled according to a cubic polynomial to capture the S-shape. The parameters and goodness of fit of the cubic polynomials are included in Table 3-2.

Finally, the throttle curves are modeled by re-scaling them to their respective throttle ranges. For numerical stability, the rotational speed is divided by 1000 (kRPM = kilo-RPM).

The curves show good agreement, motivating the use of a single averaged model. Its parameters are included in Table 3-3. The R^2-value of 1 is no accident - the average curve of three cubic polynomials is again a cubic polynomial.

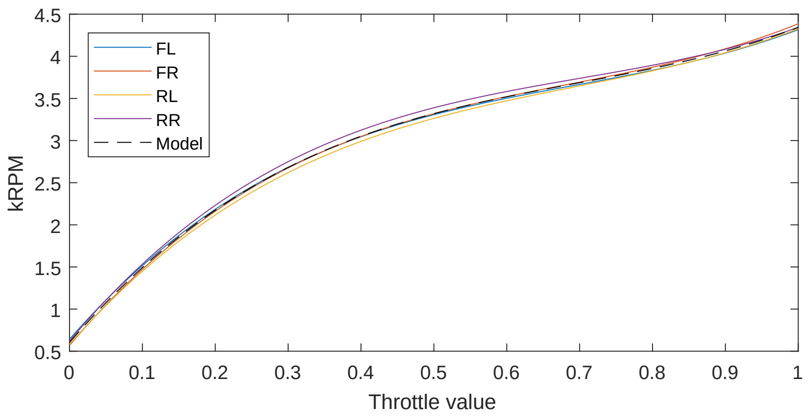


Figure 3-6: ESC PWM to RPM curves

	mean	std	x^3	x^2	x	1	R^2
Average model	0.5	0.293	0.1468	-0.2905	0.668	3.32	1

Table 3-3: Average throttle to RPM model

3-1-5 Thrust and torque coefficient measurement

Experimental Set-up

The thrust and torque coefficients are determined in a steady state measurement using respective load cells. The rotor speed is increased in small steps and allowed to settle. For every command, the corresponding data points are cut from the data set and plotted with respect to time. The first quarter of the data points are discarded as they may still include the acceleration phase. For the remaining data points, the mean and standard deviation are calculated to obtain a point in the (ω, F)-diagram. Finally, a quadratic curve is fitted in the (ω, F)-diagram.

Results

For every command step, the dynamic data show a quick convergence to a new steady state level (Figure 3-7), and warrant the use of the last three quarters of the data set for coefficient analysis. The mean and standard deviation of the rotational speed and forces are calculated and displayed in a scatter plot for verification.

Combining the measurements per level, a full thrust curve is generated. In Figure 3-8, the quadratic relationship is directly visible, save a small deviation for high rotational speed. Nevertheless, a quadratic model is deemed sufficient. An identical procedure is carried out for the torque forces, resulting in the figure and fit of Figure 3-9. The model parameters, to be used as c_T and c_Q in the four-parameter model (Equation 3-10), are reported in Table 3-4.

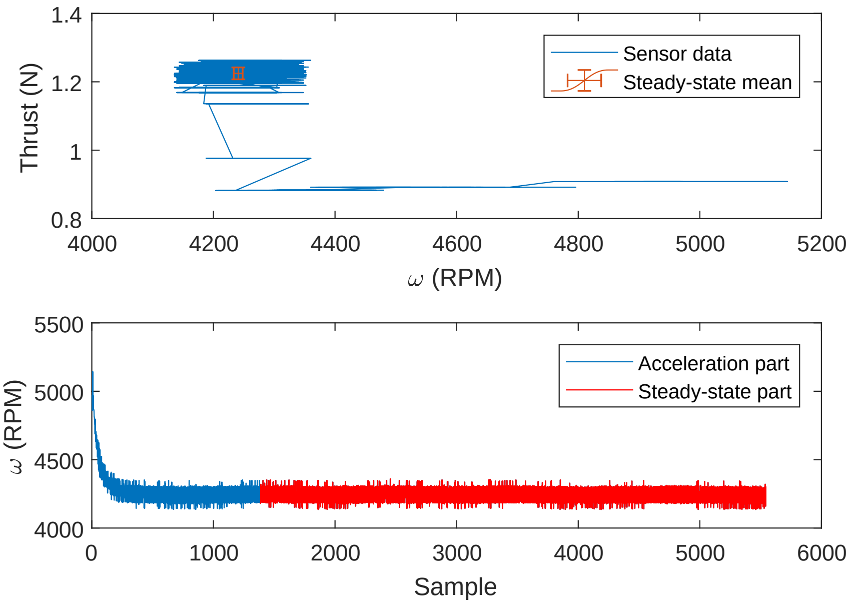


Figure 3-7: RPM and thrust analysis

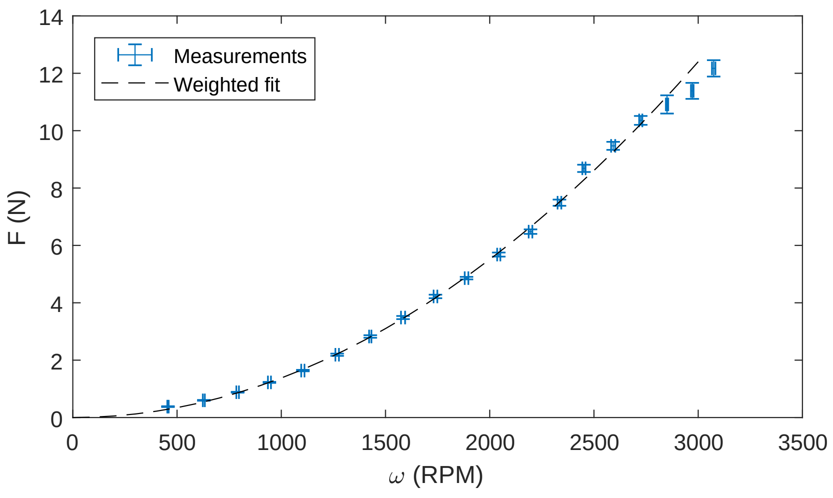


Figure 3-8: Thrust coefficient analysis

	x^2	x	1	R^2
Thrust model fit	$1.378 \cdot 10^{-6}$	0	0	0.9977
Torque model fit	$4.824 \cdot 10^{-8}$	0	0	0.9830

Table 3-4: Thrust and torque model fits

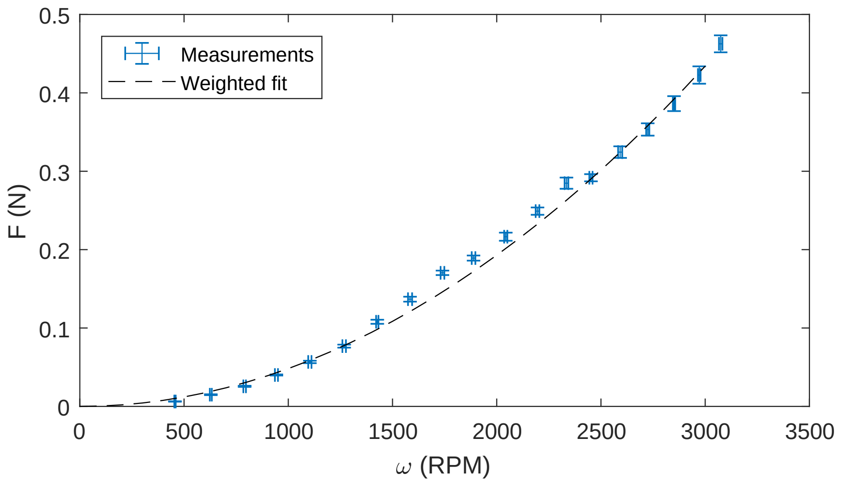


Figure 3-9: Torque coefficient analysis

3-1-6 Dynamic response

Experimental Set-up

The state space parameters are determined using a dynamic analysis of step input commands. Using the same data set, the rise time after a step command is determined for a variety of operating points by analyzing the RPM changes. From the rise time, the time constant of the system and corresponding parameter a of the four-parameter model (see Equation 3-10) is determined.

The overshoot parameter b is modeled by normalizing the load cell measurement values from 0 (before) to after (1) the step input and taking the square root of the forces (as ω_Q of the four-parameter model is later squared again). The overshoot factor is matched by varying b until the system response roughly corresponds with the observed measurements.

Results

A simple program analyzes the full data set, detects input step changes, isolates the individual responses and prompts the user to decide whether or not to include the response for rise time analysis, as some initial steps may have been required to arm the ESC and direct the motor to its minimum speed.

For a variety of throttle values (with minimum and maximum corresponding to the respective ESC in Table 3-1), the rise times are determined. The results are displayed in Figure 3-11.

For steps at a low throttle value, the rise time is significantly larger than for higher throttle values, where the rise time is seen to remain nearly constant. As a drone normally operates a throttle value around 50%, the six significantly larger rise times in the low-throttle regime are discarded for modeling. The results are displayed in Figure 3-11, with indication of the fitted mean of 150 ± 23 ms, corresponding to $a = 15$ in the four-parameter model.

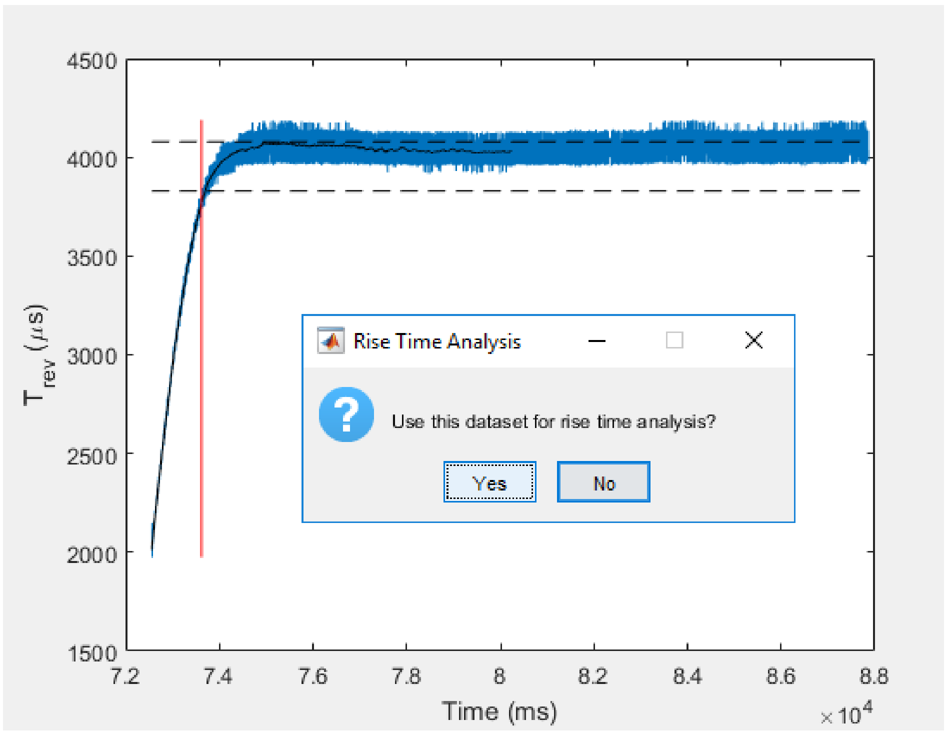


Figure 3-10: User prompt during rise time analysis

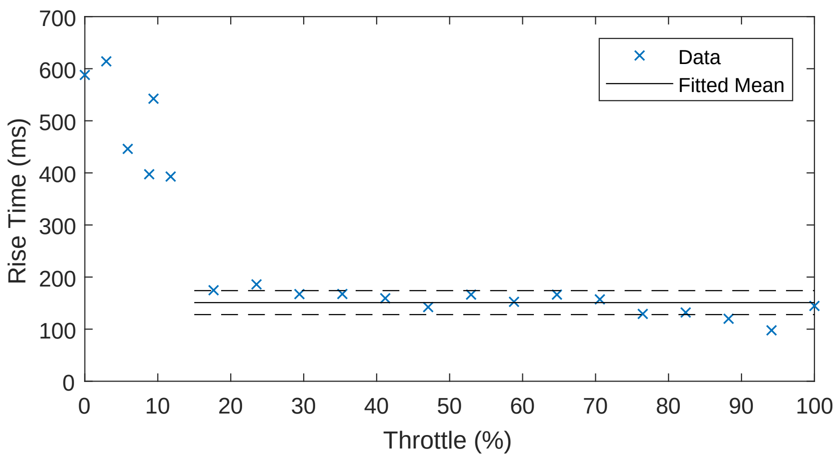


Figure 3-11: Fitted rise times for a variety of target throttle levels

The load cell measurements are delayed and averaged measurements at a rate of 10 Hz, as the function library performs internal averaging before reporting a load value. The values are however still useful to determine the overshoot percentage. As the RPM measurements are not delayed (Fig. 3-7), and thrust is assumed as a directly coupled and immediate effect of rotational speed, the load cell measurement delay can be estimated at 300 ms by matching the first order responses. The precise measurement delay is however not a relevant model parameter. The overshoot parameter is estimated at $b = 80$.

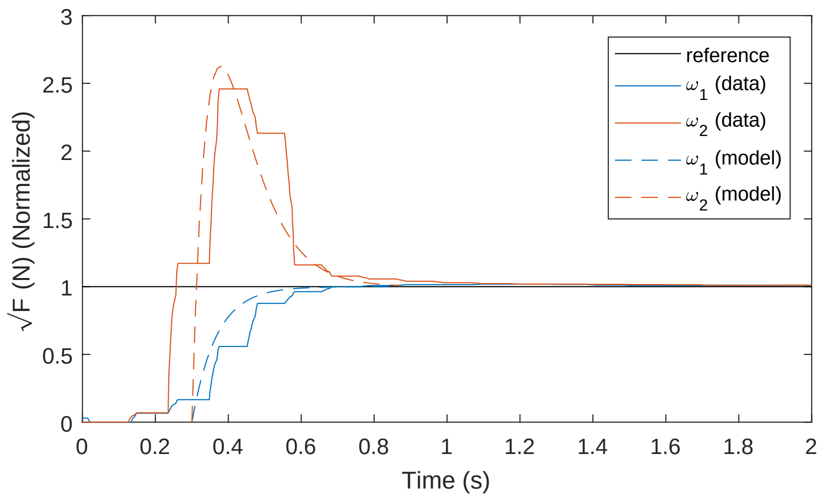


Figure 3-12: Fitting the overshoot parameter to load cell data

3-2 Stepper motor model

3-2-1 System description

For the tilting of the arms that are fitted with the propellers, stepper motors are used. Stepper motors are DC motors which can be controlled precisely in angular position and exhibit a high holding torque. They are controlled by a dedicated controller which powers the different motor coils to attract the rotor to a specific angular position in so-called steps. Using PWM signals on adjacent coils, a high resolution can be obtained. This technique is called micro-stepping.

A drawback of stepper motors is that during high acceleration or high torque movements, angular slip may occur, resulting in a discrepancy between the actual angular position and the position the motor controller has counted during its successive powering of coils. Advantageously, a stepper motor is coupled with an angular encoder to detect the angular position even in cases of slip.

The drone test-bed is fitted with four NEMA 17 stepper motors, connected to the propeller arm by means of toothed pulleys and a toothed belt. The motors are controlled by a commercially available stepper motor shield for the Arduino UNO, with a full-stepping configuration being associated with 200 steps per rotation. The shield also offers capabilities for micro-stepping up to 1/16 step size.

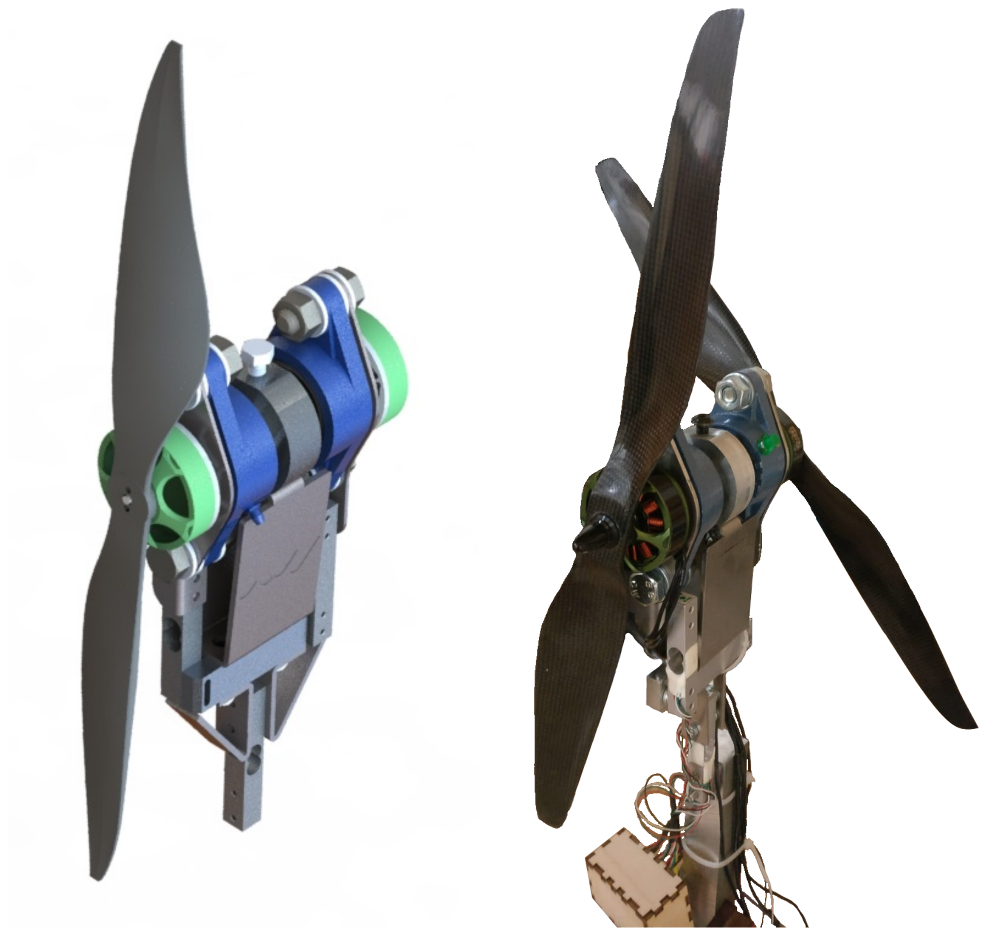


Figure 3-13: Left: perspective render of the T3 CAD design with a single rotor. Right: Constructed T3 experiment in a coaxial testing configuration

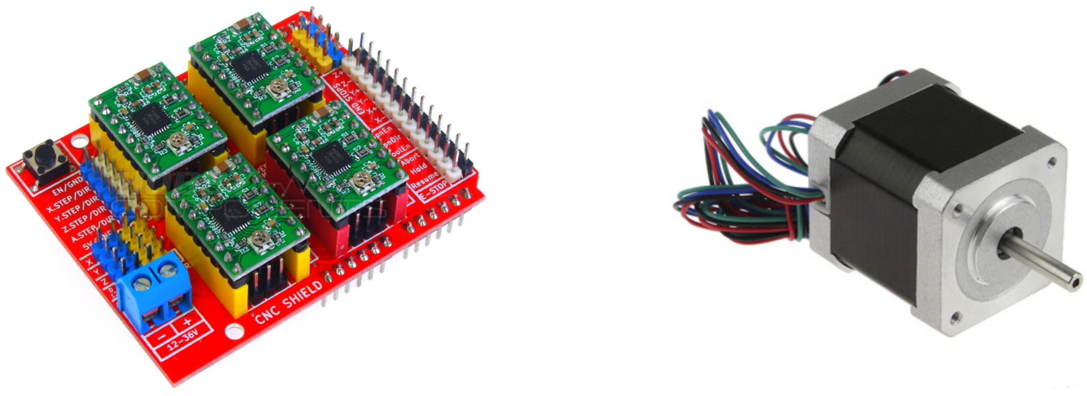


Figure 3-14: Left: Arduino UNO CNC Shield. Right: Nema 17 stepper motor

The proposed model is directly related to the code that is implemented on the Arduino. Every time step, the code checks the current position with the desired position and sends a step signal to move a single step into the direction that will minimize the error. This is repeated until the desired position is achieved. The implementation of the code is such that after the Arduino command has been sent to the shield (and while the motor controller does its work), new commands can be received via serial communication, thus constituting non-blocking control. No special features such as speed ramping have been implemented. For more information, see Section 4-1-3.

The stepper motor including motor controller is a non-linear model. Doubling the control input (reference angle) does not speed up the movement - as the step sizes are fixed. In fact, once a specific micro-stepping setting has been chosen, the only variable for consideration is the loop speed. It needs to be set as high as possible without introducing motor slip.

The angular control model can be summarized as

$$\begin{aligned} u_i &= \sigma(\delta_{i,ref}(t) - \delta_i(t)) \\ \delta_i(t+\tau_M) &= \delta_i(t) + u_i \Delta_M, \end{aligned} \tag{3-11}$$

where u_i is the control direction of motor i, $\sigma(.)$ is the sign function, $\delta_{\text{ref},i}$ is the reference angle of motor i, δ_i is the angle of motor i, τ_M is the loop time of the motor controller, and Δ_M is the angular micro-stepping size.

To implement the stepper motor model in Model Predictive Control (MPC), it is reformulated to a linear model with input box constraints. These constraints represent the maximum step size within a specific sampling time, as the motor controller may operate at faster loop rates than the high-level MPC controller. The model equals

$$\begin{aligned} \delta_{i,k+1} &= \delta_{i,k} + u_{i,k}, \text{with} \\ -\frac{\tau}{\tau_M}\Delta_M \leq u_{i,k} &\leq \frac{\tau}{\tau_M}\Delta_M. \end{aligned} \tag{3-12}$$

Technically, the model would allow for arbitrarily small input commands to be applied. In reality the motor controller only moves once the angular error is high enough. As a result, so-called chattering, jumping between two adjacent angular positions, may occur, but the effect on the propeller arm through a gear-reduced connection is neglected.

3-2-2 Stepper Rate Limit Modeling

Experimental Set-up

The stepper motor is connected to an angular encoder and a sufficiently high angular position reference input is given. From the time series development of the motor angle, the rate limit is determined.

Results

The time series shows near perfect rate-limited behavior. The slope of the graph is fitted with a straight line, resulting in a rate limit of $\frac{\Delta_M}{\tau_M} = 50.04 \pm 0.10°/\text{s}$ ($R^2 = 0.9992$).

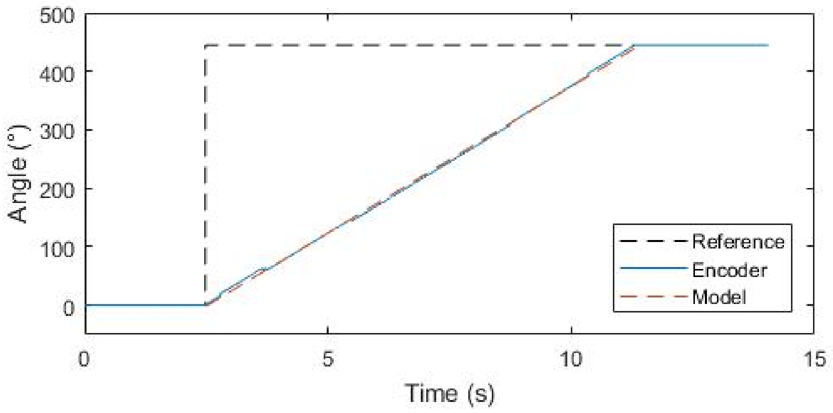


Figure 3-15: Time-series evolution of the stepper motor angle

3-3 Full system model

With the two-state description of the rotor system and the rate limits of the stepper motor in hand, the system model can be completed. The mass and inertia matrix are taken as m = 4 kg and I= 0.05I_3 (identity matrix), which is modeled after scaling up the corresponding parameters of an OS4 quadcopter system [25].

Each individual rotor model is summarized by

$$\begin{aligned}
v_1 &= \frac{u - 0.5}{0.293} \\
v_2 &= 0.1468v_1^3 - 0.2905v_1^2 + 0.668v_1 + 3.32 \\
\begin{bmatrix} \dot{\omega}_T \\ \dot{\omega}_Q \end{bmatrix} &= \begin{bmatrix} -15 & 0 \\ -80 & -15 \end{bmatrix} \begin{bmatrix} \omega_T \\ \omega_Q \end{bmatrix} + \begin{bmatrix} 15 \\ 95 \end{bmatrix} v_2 \\
T_R &= 1.378\omega_T^2 \\
Q_R &= 4.824 \cdot 10^{-2}\omega_Q^2 \\
0 &\leq u \leq 1
\end{aligned} \tag{3-13}$$

where ω_T and ω_Q have been re-scaled to kRPM units, and v_1 and v_2 are virtual throttle inputs used in the mapping. The stepper motors are modeled using a pre-determined sampling time τ according to

$$\begin{aligned}
\delta_{i,k+1} &= \delta_{i,k} + u_{M,k}, \text{with} \\
-\frac{50\pi}{180}\tau &\leq u_{M,k} \leq \frac{50\pi}{180}\tau.
\end{aligned} \tag{3-14}$$

The rotor thrust forces in the body frame are tilted using the stepper motor angles, with the convention of $\pi/2$ rad being upward and 0 rad being forward. For the total directed thrust $F_{R,B}$, we can write

$$F_{R,B} = \sum_{i=1}^{4} F_{Ri} = \begin{bmatrix} \sum_{i=1}^{4} T_{Ri} \cos \delta_i \\ 0 \\ -\sum_{i=1}^{4} T_{Ri} \sin \delta_i. \end{bmatrix} \tag{3-15}$$

The last force acting on the system is gravity. Given the tilt-rotor body orientation in roll (ϕ), pitch (θ) and yaw (ψ) coordinates (also see App. B), the gravity force $F_{g,B}$ can be expressed in the body frame as

$$F_{g,B} = 9.81 \cdot m \begin{bmatrix} -\sin(\theta) \\ -\cos(\theta)\sin(\phi) \\ \cos(\theta)\cos(\phi) \end{bmatrix}, \tag{3-16}$$

	ξ	x_B	y_B	z_B
Rotor 1	1	0.3	-0.3	0
Rotor 2	-1	0.3	0.3	0
Rotor 3	-1	-0.3	-0.3	0
Rotor 4	1	-0.3	0.3	0

Table 3-5: Rotor positions and spinning directions (1 = clockwise, -1 is counterclockwise)

and the total force in the body frame $F_{t,B}$ finally equals

$$F_{t,B} = F_{R,B} + F_{g,B}. \tag{3-17}$$

The total torque M_t on the vehicle body consists of the rotor torques Q_{Ri} and thrust-induced moments Q_{Fi}. The rotor spinning direction is indicated by ξ_i, where 1 is clockwise and -1 is counterclockwise, seen from above. The rotors are placed in a 0.6×0.6 square centered at the center of gravity (see Table 3-5).

$$\begin{aligned}
Q_{R,B} &= \sum_{i=1}^{4} Q_{Ri} = \begin{bmatrix} -\sum_{i=1}^{4} \xi_i Q_{Ri} \cos\delta_i \\ 0 \\ \sum_{i=1}^{4} \xi_i Q_{Ri} \sin\delta_i \end{bmatrix} \\
Q_{F,B} &= \sum_{i=1}^{4} Q_{Fi} = \sum_{i=1}^{4} \begin{bmatrix} x_{Bi} \\ y_{Bi} \\ z_{Bi} \end{bmatrix} \times F_{Ri} \\
M_t &= Q_{R,B} + Q_{F,B}
\end{aligned} \tag{3-18}$$

3-3-1 Equations of Motion

The dynamical equations together describe a 21-state system, with 8 inputs and 8 box constraints. The first 9 states describe the body velocity, rates and orientation.

$$
\begin{aligned}
m\dot{v}_{x,B} &= 1.378\sum_{i=1}^{4}\omega_{Ti}^2\cos\delta_i - 9.81\cdot m\sin(\theta)\\
m\dot{v}_{y,B} &= -9.81\cdot m\cos(\theta)\sin(\phi)\\
m\dot{v}_{z,B} &= -1.378\sum_{i=1}^{4}\omega_{Ti}^2\sin\delta_i + 9.81\cdot m\cos(\theta)\cos(\phi)\\
I_{xx}\dot{p} &= -4.824\cdot 10^{-2}\sum_{i=1}^{4}\xi_i\omega_{Qi}^2\cos\delta_i - 1.378\sum_{i=1}^{4}y_{Bi}\omega_{Ti}^2\sin\delta_i\\
I_{yy}\dot{q} &= 1.378\sum_{i=1}^{4}x_{Bi}\omega_{Ti}^2\sin\delta_i\\
I_{zz}\dot{r} &= 4.824\cdot 10^{-2}\sum_{i=1}^{4}\xi_i\omega_{Qi}^2\sin\delta_i - 1.378\sum_{i=1}^{4}y_{Bi}\omega_{Ti}^2\cos\delta_i\\
\dot{\phi} &= p\\
\dot{\theta} &= q\\
\dot{\psi} &= r
\end{aligned}
\tag{3-19}
$$

where I_{xx}, I_{yy} and I_{zz} are the respective moments of inertia around the roll, pitch and yaw axes, and p, q and r are the respective roll, pitch and yaw rates.

The next set of equations describes the evolution of the rotor systems. For the sake of simplicity during linearization, the inputs u_i per rotor are mapped to virtual input v_{2i} as per (3-13). For every rotor system,

$$
\begin{aligned}
\dot{\omega}_{Ti} &= -15\omega_{Ti} + 15v_{2i}\\
\dot{\omega}_{Qi} &= -80\omega_{Ti} - 15\omega_{Qi} + 95v_{2i}, \text{with}\\
0.6046 &\leq v_{2i} \leq 4.3435.
\end{aligned}
\tag{3-20}
$$

To recompute throttle input $u_i(v_{2i})$ from the optimization result, the mapping of (3-21) using the coefficients of Table 3-6 is used.

$$
\begin{aligned}
v_1(v_2) &= \sqrt[3]{c_1v_{2i} + \sqrt{(c_1v_{2i}+c_2)^2 + c_3} + c_2} + \frac{c_4}{\sqrt[3]{c_1v_{2i} + \sqrt{(c_1v_{2i}+c_2)^2+c_3} + c_2} + c_5}\\
u(v_1) &= 0.293u + 0.5
\end{aligned}
\tag{3-21}
$$

The final 4 equations describe the evolution of the stepper motors. For every motor, for a time step of τ, we have

Coefficient	Value
c_1	3.40599
c_2	-12.52
c_3	1.26565
c_4	-1.08169
c_5	0.659628

Table 3-6: Virtual throttle mapping function coefficients

$$\begin{aligned} &\delta_{i,k+1} = \delta_{i,k} + u_{M,k}, \text{with} \\ -&\frac{50\pi}{180}\tau \leq u_{M,k} \leq \frac{50\pi}{180}\tau. \end{aligned} \tag{3-22}$$

However, for accurately modeling the indirect effects of stepper angle inputs to the vehicle velocities, rates and angles, the individual discrete-time input matrices can not simply be appended to the continuous-time model. Therefore, the stepper motor was approximated by a rate-limited integrator.

During simulation, the real world position vector X_W is tracked by integrating the body velocities and orientation (see App. B).

Chapter 4

Hardware-in-the-loop

This chapter documents the real-time co-simulation system. First, the external hardware setup is presented, along with the connections and code that interface with the virtual simulator (Section 4-1). Secondly, the Robot Operating System (ROS) implementation of the simulator along with a Graphical User Interface (GUI) and visualization screen is laid out (Section 4-2).

4-1 Tiltrotor set-up

The hardware part of the co-simulation system is a fixed board which replicates input-output behavior of the drone for a subset of the total of 21 states. A total of 8 inputs can be given, through the serial interface (Section 4-1-2), one for each stepper and Electronic Speed Controller (ESC). The commands are parsed on an Arduino command board (Section 4-1-3) and sent to the respective actuators. An Arduino sensor board (Section 4-1-4) measures tilt angle and rotational speed, and returns this to the computer. As the computational power on the on-board microcontrollers is limited, tasks such as filtering and state estimation are off-loaded to the computer.

4-1-1 System sketch

The layout of the Hardware-in-the-Loop (HIL) board is motivated by the Avy Zero design, comprising four rotatable arms at the corners of the vehicle (Figure 4-1). To facilitate full rotation of the motors, the motor wiring is pulled through the arm tubes and led to the bottom side of the board. The arms are each held in place by two rotational bearings with set screws, fixing their position with respect to the board in all degrees except axial rotation.

The stepper motors are connected to the arms via toothed belts. With a 4:1 gear ratio, precise angular position control is facilitated. The stepper motor angle is tracked using a coaxial encoder, mounted directly to the motor axis.

The motor wiring and ESC's are placed on the bottom side for cable management and reduction of interference: high-current motor wires can interfere with digital signals sent to and from the top-side micro-controllers when placed directly adjacent to each other. The motor wires are joined using a four-way junction and connected to a Lithium Polymer (LiPo) battery via a safety switch.

The Rounds Per Minute (RPM) sensors directly connect to any two of each three motor wires per ESC. Their signals, and those of the coaxial encoders are read via 8 pins on the Arduino sensor board.

The control signals originate from the Computer Numerical Control (CNC) shield and Pulse Width Modulation (PWM) breakout board, both associated with the Arduino control board. The CNC shield contains four A4988 motor controllers with current limits that can be predetermined using a variable resistor. The PWM board can output up to 16 signals with a resolution of 4096 steps between zero and full duty cycle, at a rate of up to 500 Hz.

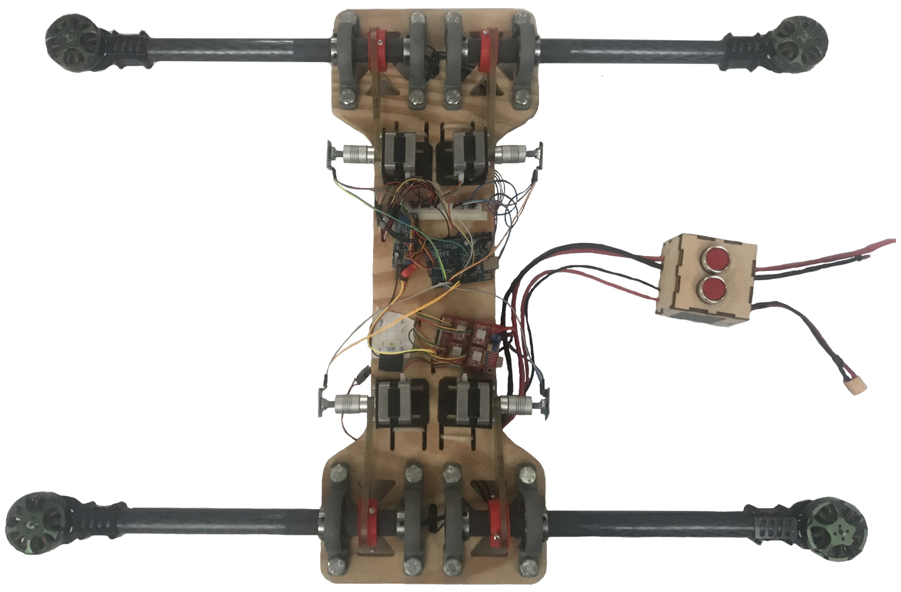


Figure 4-1: Plan view of the hardware-in-the-loop system

4-1-2 Serial interface

The existing libraries of interfacing positional commands for the CNC board, and exchanging ROS topic messages with Arduino proved too slow and heavy for implementation. A simple integer messaging system, both for the command board and sensor board was written, sending 8 integers relating to respective throttle, angle or timing values with a basic form of error reduction.

Each message is constructed in the form of "<int1, int2, int3, int4, int5, int6, int7, int8>", where the "<" and ">" symbols signal message start and end, respectively, and int1-int8 are comma-separated integer values. The message length may vary, depending on the integer value, but in both cases (commmand and sensor), will never exceed 45 characters. With a 115200 baud, and 10 bits per symbol message, the theoretical maximum messaging frequency therefore lies around 250 Hz. Realistically, time needs to be reserved for the Arduino to parse and execute the commands, and for the computer to calculate a new optimization result. In the implementation, the command messaging rate was set to 20 Hz, with the command board updating the stepper motors (and being able to listen to new incoming messages) at a rate of 125 Hz, and the sensor messaging rate was set to 40 Hz.

4-1-3 Command board

The command board listens for new control commands in the form of the aforementioned 8-integer message and splits them into ESC commands and stepper motor commands. ESC commands get sent over Inter-Integrated Circuit (I2C) as duty cycle ratios to an external PWM board capable of generating multiple high-frequency PWM signals. The stepper motor commands get sent to the CNC shield containing four A4988 motor controller chips, with each individual motor being controlled using a STEP and a DIR signal.

For stepper motors, it is essential to pause between step signals, to allow the motor to move. The code has been designed to alternate between a first listening and I2C loop segment and a second stepping segment, where the first loop is used to read and parse new commands and control the ESC's, simultaneously serving as a pause for the step commands, and the second loop is used to determine movement direction and step signals for the four individual stepper motors. Non-blocking control has been implemented, meaning that set points can be updated inbetween single steps.

4-1-4 Sensor board

The sensor board tracks the encoder positions and high-to-low signal change timings of the RPM sensors, indicating a (partial) rotation of the motor has passed. Ideally, all 12 sensor pins are interrupt pins, being able to trigger timing and increment functions based on their signal changes, but the Arduino Uno does not possess that many interrupt pins. To maximize the sampling frequency and detect signal changes as soon as possible, filtering and zero-speed detection functions are off-loaded to the computer.

The 8 integers are packed into the aforementioned message format and sent to ROS via serial connection at a rate of 40 Hz.

4-2 Simulator set-up

The software part of the co-simulation system is the ROS kernel, an open-source development platform for Linux with built-in functionality for a variety of robot-related tasks, such as communication, control, localization and visualization. The software layout of the implementation in this thesis is explained in Section 4-2-1. The data reception, filtering and state observation is explained in Section 4-2-2. The measured and observed states are supplemented with simulated states to recreate a full 21-state simulator (Section 4-2-3). The simulated states are then used to generate Proportional Integral Derivative (PID) control signals for various degrees of motion (Section 4-2-4). The motion of the system is displayed in a visualization window (Section 4-2-5), and the simulation is controlled with a GUI comprising a variety of sliders, dials and buttons (Section 4-2-6).

4-2-1 ROS Architecture

The ROS system implements a central "master", which provides a connection between programs ("nodes") and data passed between them ("topics"). Nodes can be run at instances triggered by an event, or at specific rates. As the data exchange is handled centrally, the nodes can be run largely independent from each other. They exchange data by subscribing or publishing to a topic. The advantage of this structure is that separate functions can easily be added or edited, and even run on separate hardware - as long as it communicates with the master.

The nodes are divided into workspaces - for example, code related to the mathematical calculations for simulation is placed into the "sim" space, code related to visualization is placed into the "viz" space. This is nothing but a handle, like a file folder, but it provides an intuitive separation of topics that are relevant within that space only, and the ones being passed inbetween.

An overview of all the nodes and their associated topics running in the co-simulation setup is displayed in Figure 4-2. The direction of the arrows indicates the flow direction of data. The "/HIL/sender" node is easily identified as a virtual sink - only data gets sent to it. It is the program that exports the commands over Universal Serial Bus (USB) to the command Arduino. Likewise, the "/HIL/receiver" node is the serial listener, reading sensor messages from the sensor Arduino, and passing them, via an observer and filter (Section 4-2-2), to the simulation. The virtual sink "/viz/viz_window" is a screen output, displaying the current flight situation to a user.

Nodes can be started using a launch file, providing an all-in-one shortcut to launch a complete network of programs. The launch files can also specify specific parameters to run the nodes with, such as the associated USB port, their execution rate or PID parameters. Additionally, nodes can be started and killed at run-time.

4-2-2 Filtering and observing

To maximize sampling frequency, computationally intensive tasks such as filtering, observing and zero speed detection are handled on ROS rather than on the sensor Arduino.

First, the RPM value is computed for every motor from the timing value. Extreme RPM values outside the 0-10000 range are dropped to zero. An exponential moving average filter with 5% weight to the newest data point is implemented to reduce noise as

$$\omega_{\text{f,new}} = 0.95\omega_{\text{f,old}} + 0.05\omega_{\text{new}}, \tag{4-1}$$

where ω_{new} is the most recent received RPM value in the 0-10000 range, $\omega_{\text{f,old}}$ is the previous filter value (initialized at 0) and $\omega_{\text{f,new}}$ is the new filter value, which is used as the ω_T in the state space model of Chapter 3.

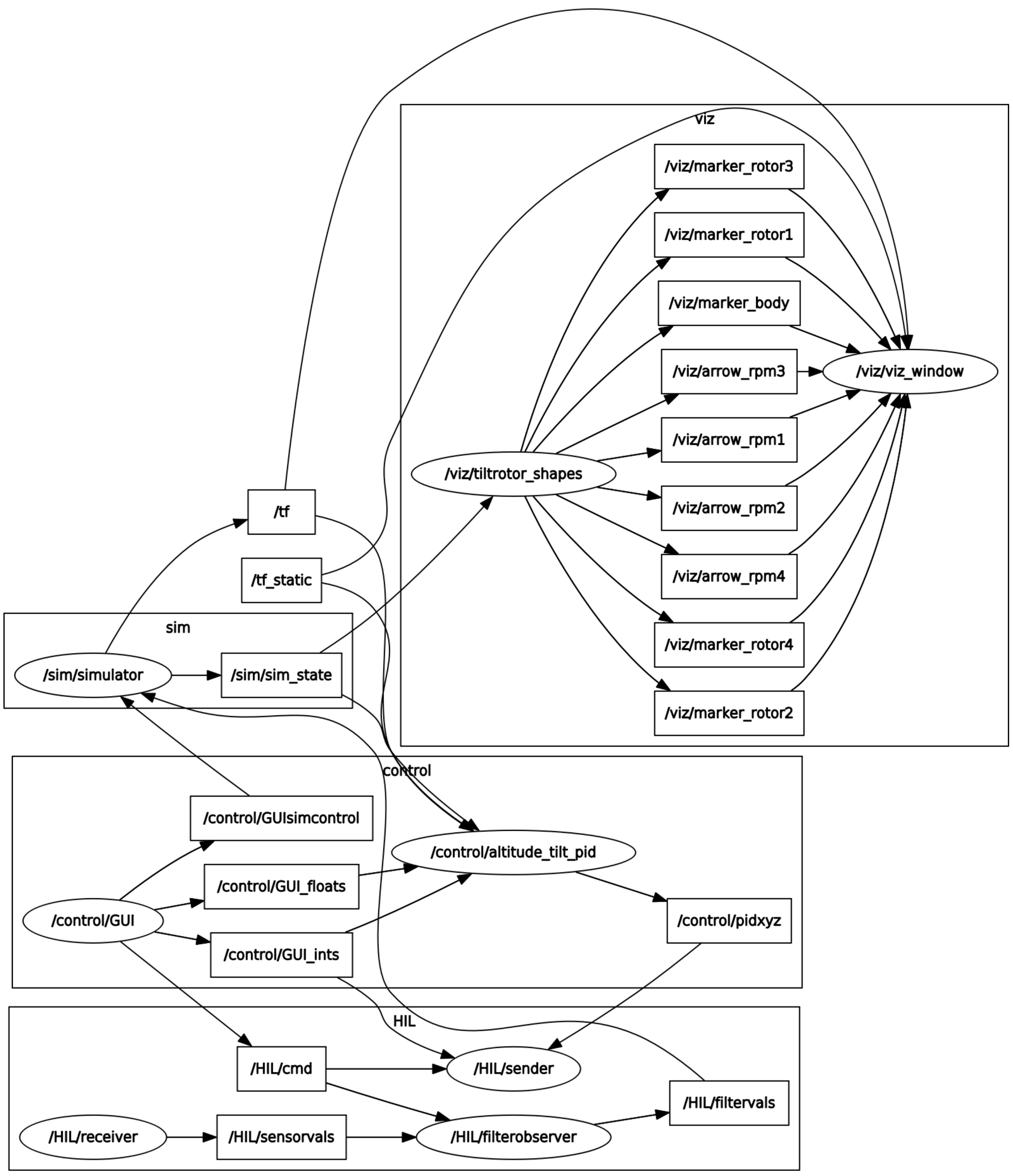


Figure 4-2: ROS overview of all active nodes and topics

When the rotor stops spinning, the sensor does not send peaks to the Arduino anymore. In this case, the last timing value gets repeated as output and the filter will not converge to 0 RPM. To accurately determine this scenario, the most recently received timing value is compared to the last. Should they be equal, a counter is increased. When the counter reaches 10, it is assumed the motor has stopped, and the corresponding RPM is set to zero. Should the timing value change again, the counter is reset to 0.

The ω_Q of the state space model is not measured directly, but is observable. Using the fitted state space, observer poles are placed at (-12, -13) and the observer is converted to a discrete-time state space implemented at 40 Hz.

The stepper motor encoder signals are not filtered.

4-2-3 Internal simulator

From the resulting ω_T, ω_Q and δ per arm of the HIL board, the evolution of the system can be simulated. Using the equations of Section 3-3-1, a state update is calculated at 40 Hz using forward Euler integration.

The simulator can be restricted in degrees of motion (except altitude) and reset using toggle signals from the control GUI. This facilitates testing and tuning separate controllers, without having to restart the complete network.

4-2-4 Controllers

The closed loop system is stabilized for limited degrees of freedom using three PID controllers. The limitations reflect the dynamics of the first envisioned dynamical flight experiment (see Chapter 7), in which the aircraft can take off, translate in one direction and perform yawing maneuvers.

1. The first PID controls altitude and outputs the same throttle reference signal to all four rotors.
2. The second PID controls forward speed and outputs the same tilt reference signal to all four steppers.
3. The third PID controls yaw rate and outputs a positive tilt reference signal to the left steppers and an equal but opposite tilt reference signal to the right steppers.

The controllers are implemented using a simple discretisation scheme at 40 Hz. Given the control gains K_P, K_I and K_D, state reference value x_{ref}, current state value x_{curr}, previous error value e_{prev} (updated iteratively, initialized at 0), and previous integrator state i_{prev} (internal state, initialized at 0) the control action is computed every time step as

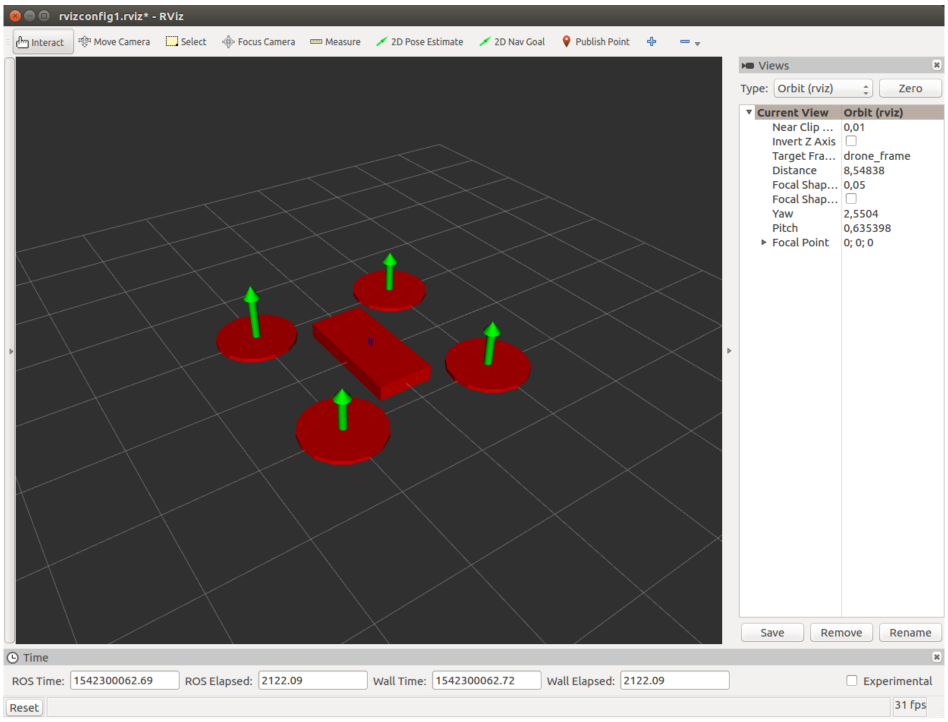
$$\begin{aligned} e_{\text{PID}} &= x_{\text{ref}} - x_{\text{curr}} \\ i_{\text{PID}} &= i_{\text{prev}} + e_{\text{PID}}/40 \\ u_{\text{PID}} &= K_P e_{\text{PID}} + K_I i_{\text{PID}} + 40 K_D (e_{\text{PID}} - e_{\text{prev}}), \end{aligned} \tag{4-2}$$

where e_{PID} is the current PID error value (which updates e_{prev}), i_{PID} is the current PID integral state and u_{PID} is the computed PID control action.

Integrator states for all three controllers can be individually reset to 0 using three buttons on the user interface (Section 4-2-6).

4-2-5 Visualization

Using the simulation state, transfer functions and tilt angles are published to a visualization node. The visualization node places boxes and cylinders, representing the tilt-rotor system in a 3D space, which is displayed using rViz. Additionally, live directed thrust forces are displayed using arrows originating at the respective rotor centers.



Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0,01
Invert Z Axis	☐
Target Fra...	drone_frame
Distance	8,54838
Focal Shap...	0,05
Focal Shap...	☐
Yaw	2,5504
Pitch	0,635398
Focal Point	0; 0; 0

ROS Time: 1542300062.69 ROS Elapsed: 2122.09 Wall Time: 1542300062.72 Wall Elapsed: 2122.09

31 fps

Figure 4-3: rViz visualization window displaying live position, thrust forces and tilt angles

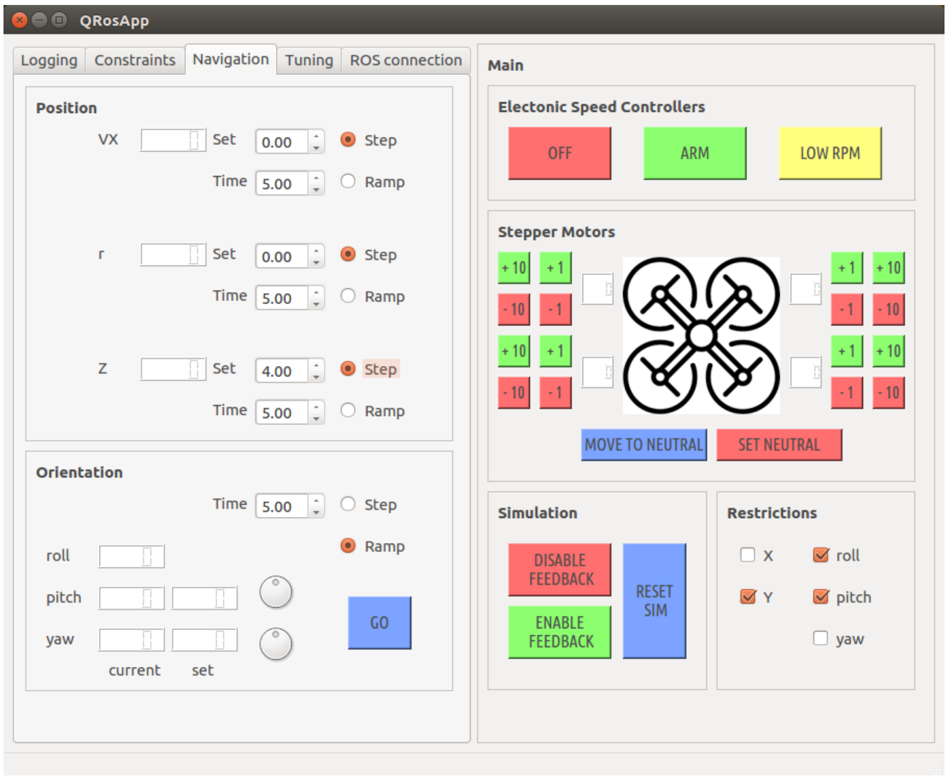
4-2-6 User Interface

The user interface is a source node for a test engineer to interact with. It contains safety buttons ("ARM/OFF"), calibration buttons to center the stepper motors, simulation control toggles and a feedback control switch. The most important buttons are colorized to indicate their behavior (green is activation or increase, red is deactivation or decrease). These buttons are always visible in the right part of the window, as they need to be accessed regularly, but also quickly, in case of emergencies.

The left part of the window contains several clickable tabs. The most important tabs are the Navigation Tab (Figure 4-4) and the Tuning Tab (Figure 4-5).

The Navigation Tab contains input boxes for forward speed, yaw rate and altitude. A new reference level can be set for each variable, and the decision to change it using a step command or a ramp (linear) command spread out over a time interval. The new levels are published to the controller at the push of the blue "GO" button.

The Tuning Tab contains input boxes for the altitude, forward speed and yaw rate PID controllers, along with individual integrator reset buttons and an overall parameter publish button ("SET"). A provision for Model Predictive Control (MPC) tuning weights has also been implemented.

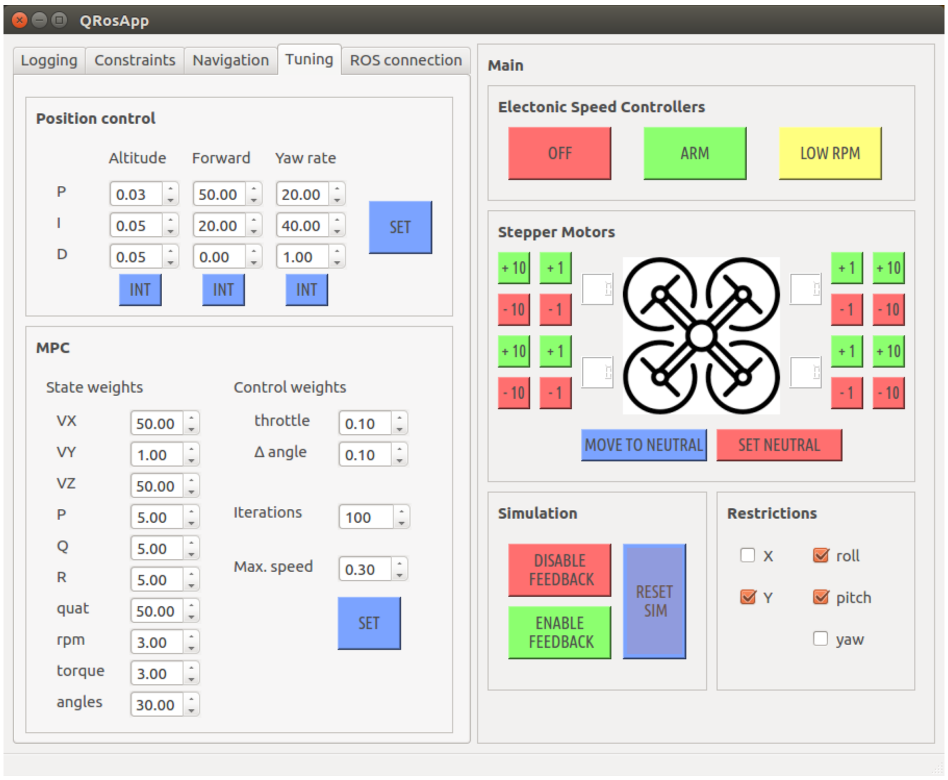


Position
VX Set 0.00 [✓ Step] Time 5.00 Ramp
r Set 0.00 [✓ Step] Time 5.00 Ramp
z Set 4.00 [✓ Step] Time 5.00 Ramp

Orientation
Time 5.00 Step [✓ Ramp]
roll
pitch
yaw

Restrictions
X [✓ Y] [✓ roll] [✓ pitch] yaw

Figure 4-4: Navigation Tab



Position control

	Altitude	Forward	Yaw rate
P	0.03	50.00	20.00
I	0.05	20.00	40.00
D	0.05	0.00	1.00

MPC

State weights	
VX	50.00
VY	1.00
VZ	50.00
P	5.00
Q	5.00
R	5.00
quat	50.00
rpm	3.00
torque	3.00
angles	30.00

Control weights	
throttle	0.10
Δ angle	0.10
Iterations	100
Max. speed	0.30

Electonic Speed Controllers

Stepper Motors

Simulation

Restrictions

☐ X ☑ roll
☑ Y ☑ pitch
☐ yaw

Figure 4-5: PID and MPC Tuning Tab

Chapter 5

Results

The time-varying Model Predictive Control (MPC) controller is simulated on the full 21-state tilt-rotor model and compared to adaptive and fixed-model implementations in Section 5-1. Five flight scenarios are presented: a forward and upward flight (twice), a yaw-right command, and a partial backflip. Every scenario is initialized by a mid-air hover condition.

5-1 Simulation

The controller is simulated using a time step of 0.025, corresponding to a 40 Hz control frequency. The prediction horizon is $N_P = 5$ and state weights are as in Tables 5-1 and 5-2, unless indicated otherwise. The weights are motivated by penalizing velocity and orientation errors over the other states, and penalizing input-associated states from an energy perspective - rotor action is costly whereas stepper angles constitute no change in energy usage. A stepper motor is namely always energized, irrespective of its orientation.

The time-varying controller uses three models along the horizon, motivated by the example of Section 2-4.

Five scenarios are simulated. The first scenario, a yaw-right command, highlights the influence of rotor tilting (Section 5-1-1). The second and third scenarios are related to a combined liftoff - forward motion command, where both body pitching and stepper action can be used to achieve reference tracking (Sections 5-1-2 and 5-1-3). The fourth scenario is a pitch-up command, an orientation change regular drones can not attain without holding their position (Section 5-1-4). The final scenario is an extension of the pitch-up command, where the vehicle is commanded to keep pitching up until instability occurs or flying completely upside down (Section 5-1-5).

As an outer-loop position tracking controller was not implemented, the scenarios do not include roll angle commands. Moreover, in roll-only scenarios, the performance of a tilt-rotor can be expected to be similar to normal non-tilting quadrotor systems.

Q	v_x	v_y	v_z	p	q	r	ϕ	θ	ψ	ω_{Ti}	ω_{Qi}	δ_i
Weight	10	10	10	0.1	0.1	0.1	5	5	5	0.5	0.5	0

Table 5-1: MPC state penalty weights

R	v_{2i}	u_{Mi}
Weight	0.02	0.02

Table 5-2: MPC control penalty weights

5-1-1 Yaw-right

The controller is given the step reference signal of Table 5-3, to indicate a 20 degree yaw command to the right.

	v_x	v_y	v_z	p	q	r	ϕ	θ	ψ	ω_{Ti}	ω_{Qi}	δ_i
x_{ref}	0	0	0	0	0	0	0	0	0.3491	2.6670	2.6670	1.5708

Table 5-3: Step reference signal for the yaw-right scenario

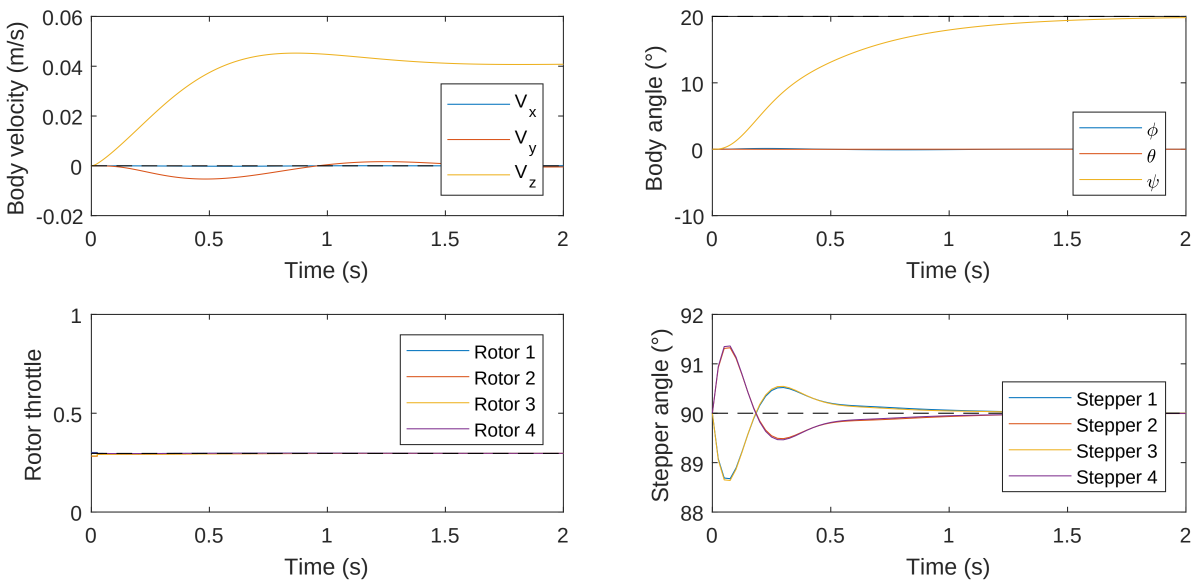


Figure 5-1: Body velocities, angles and control inputs for fixed-model MPC in a yaw-right scenario

The difference in weighting between rotor action and stepper action results in a stepper-heavy maneuver. As visible in Figure 5-1, the rotor input remains virtually on the equilibrium command, whereas the left stepper motors start tilting forward at the start of the simulation, and the right stepper motors tilt backward. The initial action is larger than the correction to slow the yaw rate. During the maneuver, the vehicle however loses some vertical thrust, resulting in a slow descent rate of 4 cm/s.

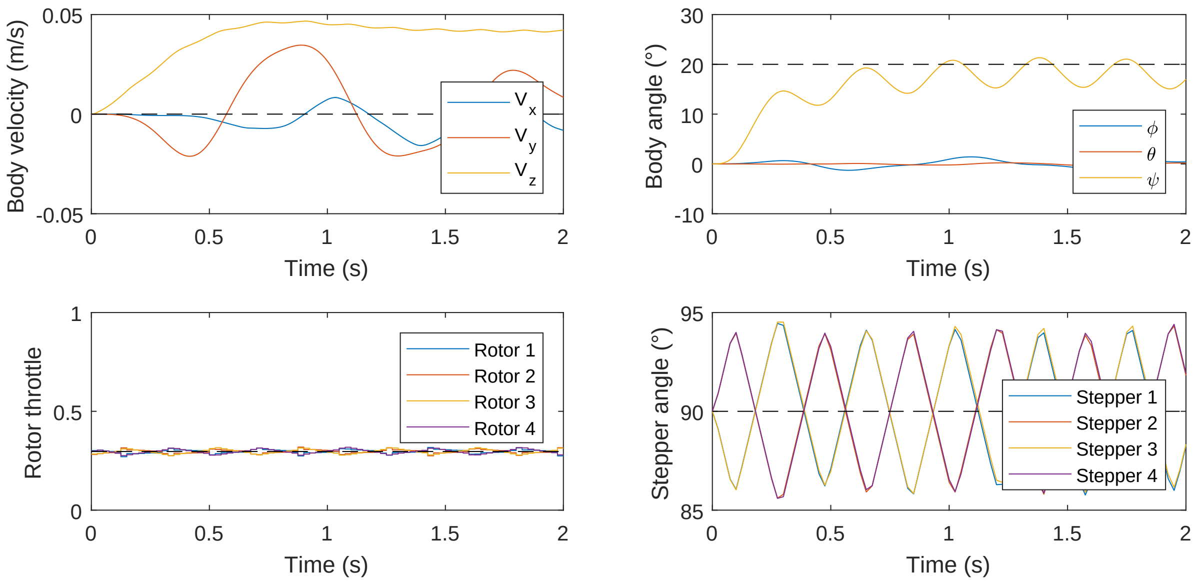


Figure 5-2: Body velocities, angles and control inputs for adaptive MPC in a yaw-right scenario

The adaptive controller shows a similar response, but overuses the stepper motors, tilting them to maximum angles 3 times as large as the fixed-model controller. The prediction model variation by relinearizing around the current stepper angles are likely to blame, as yaw changes or body speed changes do not contribute to a differently linearized model.

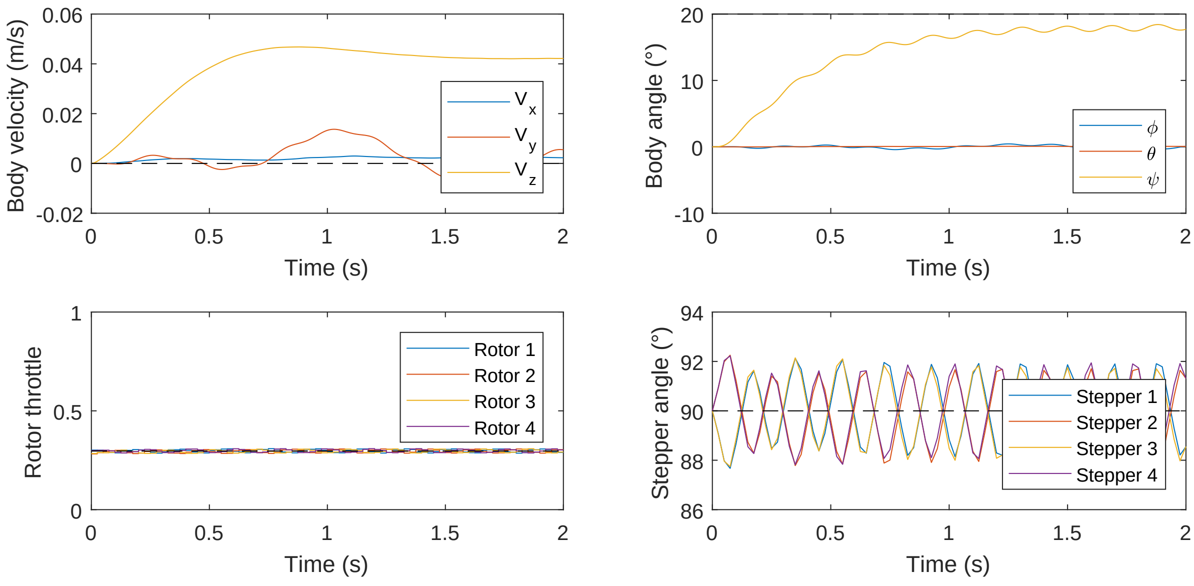


Figure 5-3: Body velocities, angles and control inputs for TVMPC in a yaw-right scenario

The time-varying controller achieves a more satisfactory response, reducing the stepper action and associated oscillations. The same underlying effect of prediction model variation effects through the stepper angle changes is present, especially as the model of the first step is identical to that of the adaptive controller. Varying the subsequent models according to previously predicted development however succeeds to mitigate the oscillatory effects of Figure 5-2.

5-1-2 Forward and upward flight

The controller is given the step reference signal of Table 5-4. The negative vertical body velocity reference of -0.5 m/s corresponds with upward motion in the world reference frame.

	v_x	v_y	v_z	p	q	r	ϕ	θ	ψ	ω_{Ti}	ω_{Qi}	δ_i
x_{ref}	0.5	0	-0.5	0	0	0	0	0	0	2.6670	2.6670	1.5708

Table 5-4: Step reference signal for the lift-off scenario

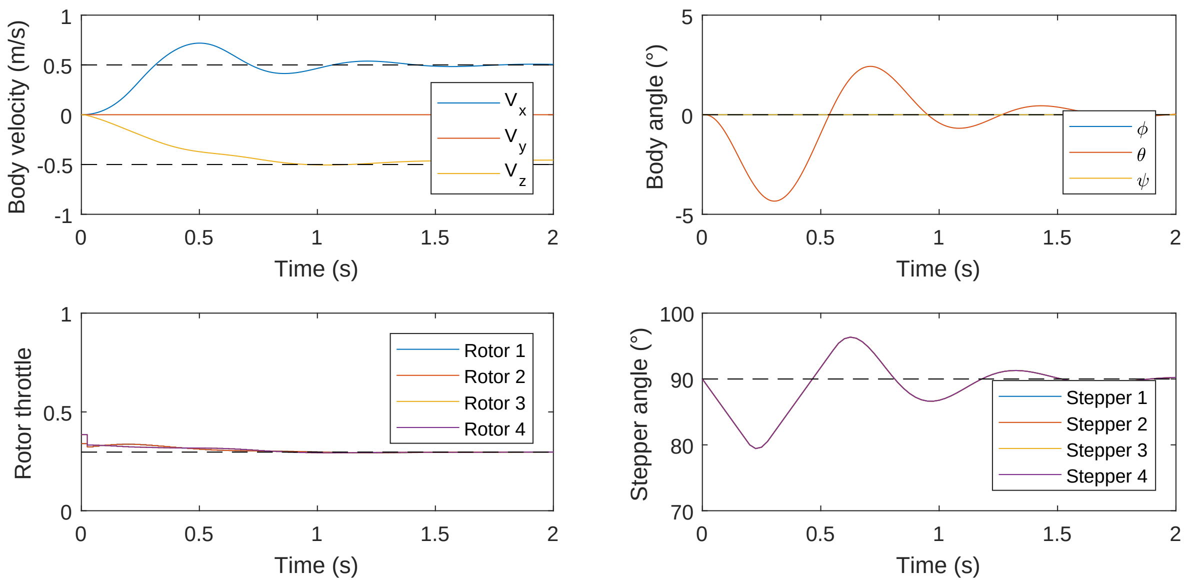


Figure 5-4: Body velocities, angles and control inputs for fixed-model MPC in a lift-off scenario

The fixed-model controller achieves quick and satisfactory tracking within 2 seconds. Both body pitching and stepper motor pitching are utilized to accelerate the vehicle forward. Vertical velocity tracking is achieved faster than forward velocity tracking, which is expected as the thrust forces are not yet pointing in the forward direction at the start of the simulation.

The adaptive controller overshoots the target reference more than the fixed-model controller and has significant trouble dampening the pitch oscillations and associated variations in forward velocity. The controller favors stepper motor action heavily over rotor action, but in doing so, fails to converge.

It appears that using an adaptive but constant prediction model along the prediction horizon underestimates the pitch and velocity response when the pitch is non-zero and the steppers are tilting.

The time-varying controller achieves an intermediate tracking performance compared to the fixed-model implementation and adaptive controller. The observed pitch response is slightly more aggressive than the fixed-model baseline, and shows convergence within 2 seconds. The favoring of stepper action versus rotor action is visible in the bottom two figures. It should be noted that the pitch angle does not return to a zero steady state. This is likely an effect of mismatched models along the prediction horizon.

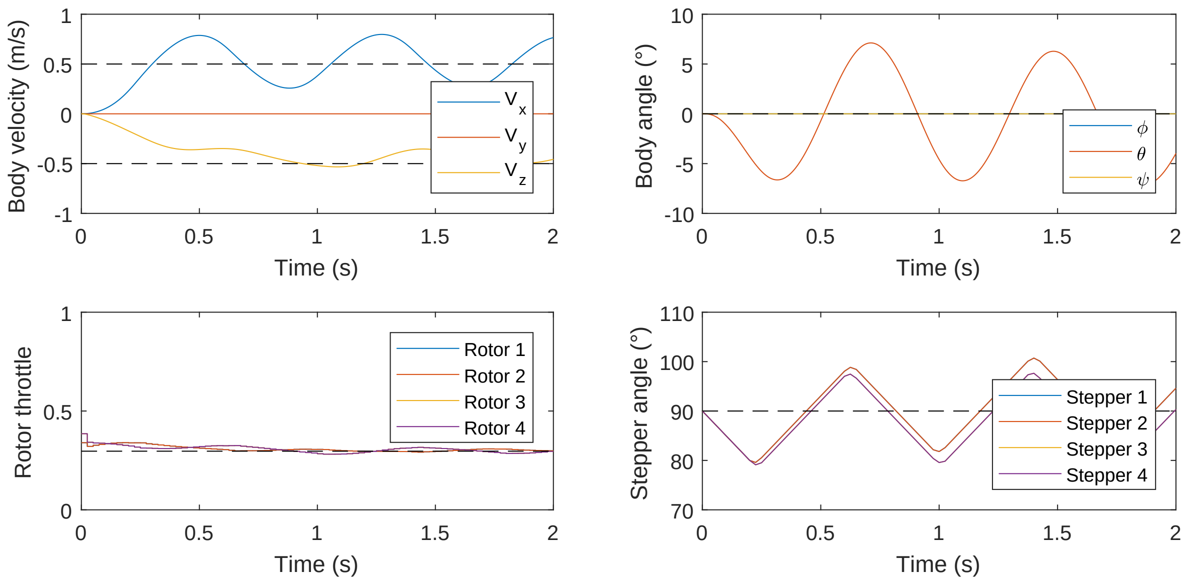


Figure 5-5: Body velocities, angles and control inputs for adaptive MPC in a lift-off scenario

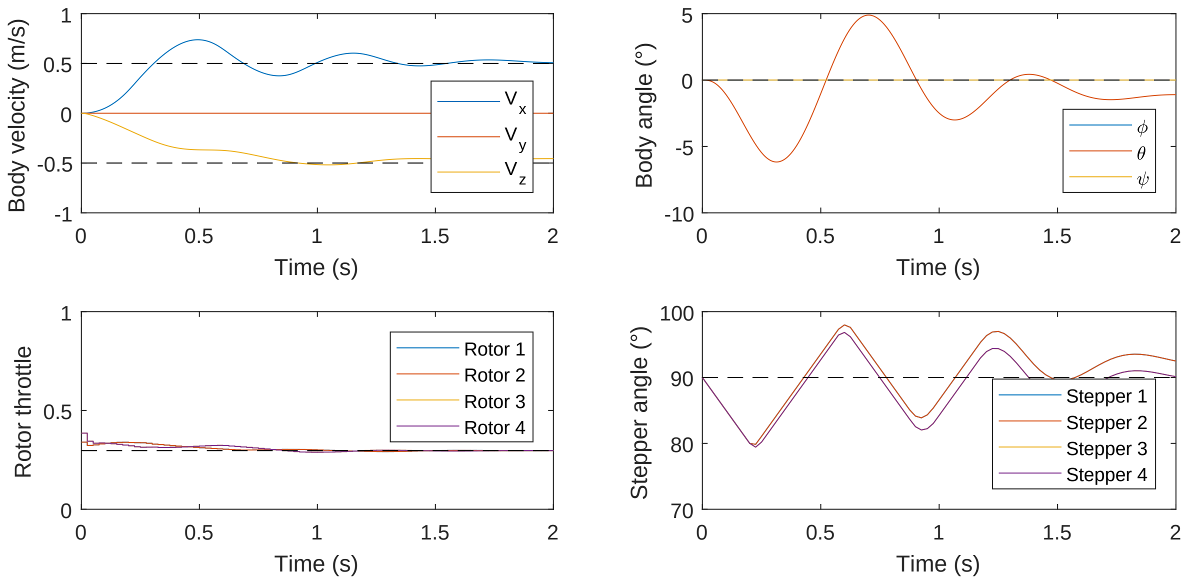


Figure 5-6: Body velocities, angles and control inputs for TVMPC in a lift-off scenario

5-1-3 Forward and upward flight 2

Arguably, stepper action is most suited for yaw control such as the simulation of Section 5-1-1, or pitch tracking such as Sections 5-1-4 and 5-1-5. Therefore, the forward and upward flight scenario is repeated with a reduction in penalties for the rotor states and input (see Tables 5-5 and 5-6).

Q	v_x	v_y	v_z	p	q	r	ϕ	θ	ψ	ω_{Ti}	ω_{Qi}	δ_i
Weight	10	10	10	0.1	0.1	0.1	5	5	5	0.02	0.02	0

Table 5-5: MPC state penalty weights

R	v_{2i}	u_{Mi}
Weight	0.01	0.01

Table 5-6: MPC control penalty weights

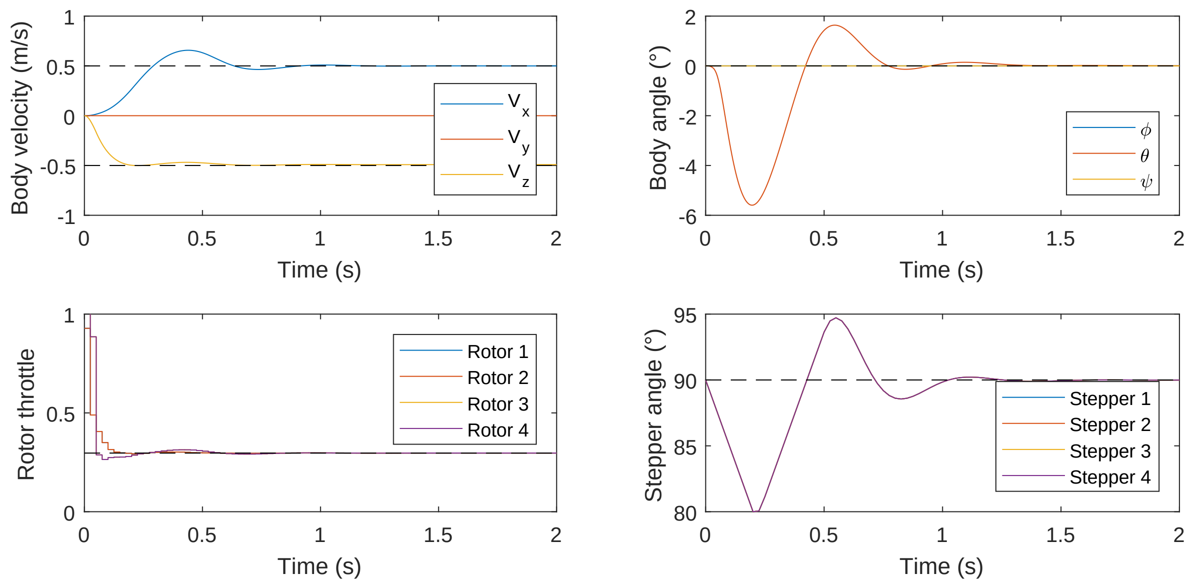


Figure 5-7: Body velocities, angles and control inputs for TVMPC in a lift-off scenario

This time, the response is much more favorable. The system converges within 1 second and shows no steady state error. The rotor action peak at the start of the simulation, in order to generate the forward-upward thrust is also directly apparent.

The choice to alleviate rotor action also has a positive influence on the adaptive controller response. Although the oscillations are non-negligible in comparison with the fixed-model implementation, they show better convergence than in Figure 5-5.

In this scenario, the time-varying controller arguably shows the most preferred response of the three implementations. Although the overshoot ripples in the body speeds are minimally larger than those of the fixed-model response, convergence is achieved within 0.7 seconds, whereas the other two controllers have not converged at that time yet.

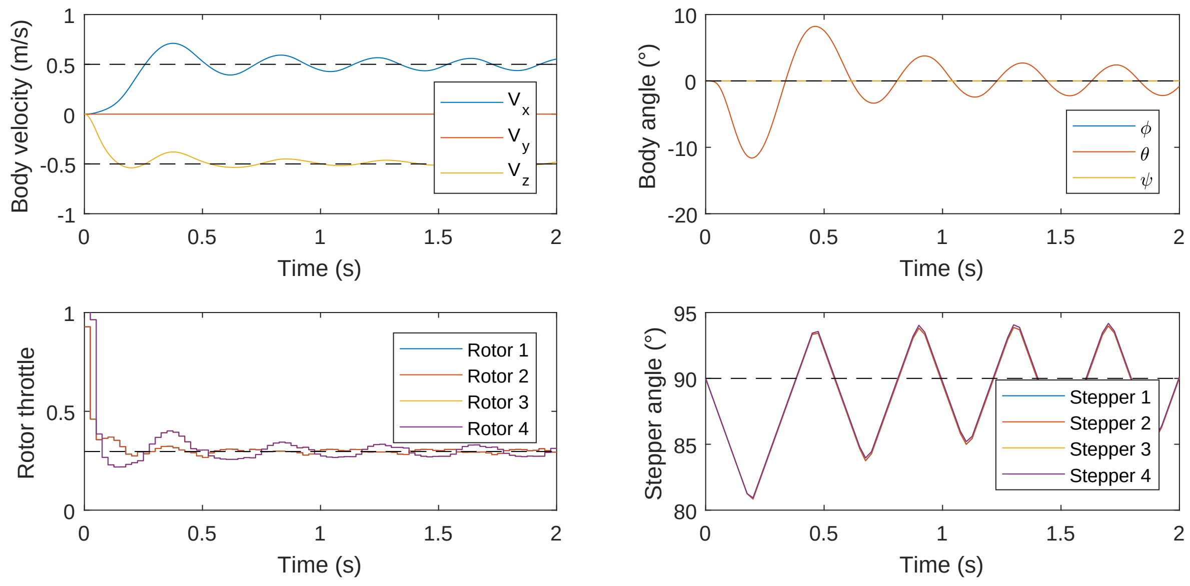


Figure 5-8: Body velocities, angles and control inputs for TVMPC in a lift-off scenario

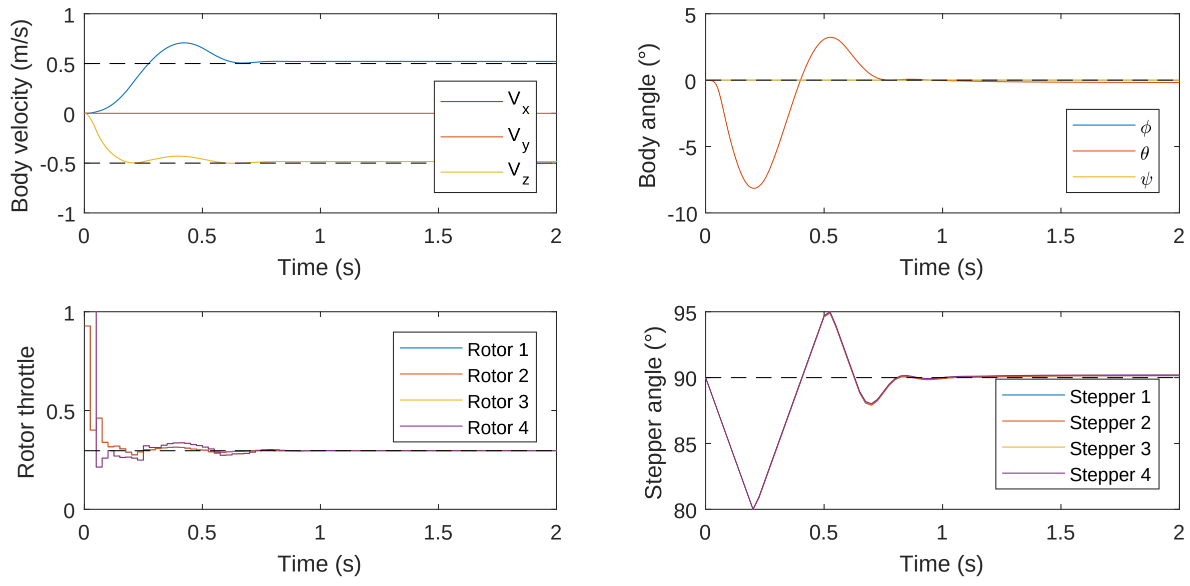


Figure 5-9: Body velocities, angles and control inputs for TVMPC in a lift-off scenario

The differences of the individual controllers in velocity and pitch tracking between each other and with the rotor-penalized scenario of Section 5-1-2 are summarized in Figure 5-10.

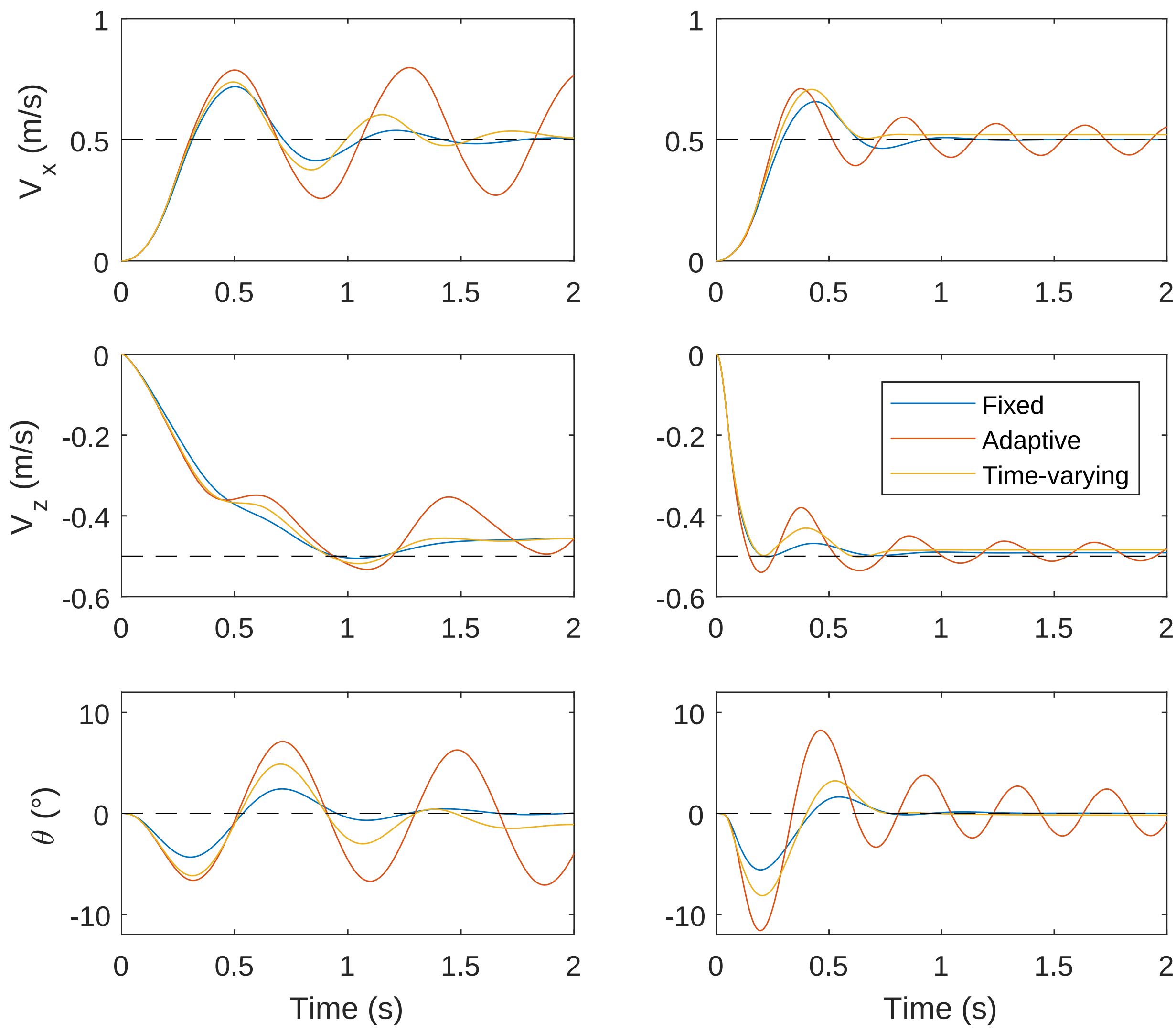


Figure 5-10: Left: V_x, V_z and θ for a heavily rotor penalized liftoff scenario. Right: V_x, V_z and θ for a lightly rotor penalized liftoff scenario.

5-1-4 Pitch-up

The controller is given the step reference signal of Table 5-7, to indicate a 20 degree pitch up command. The state penalties are again as in Table 5-1.

	v_x	v_y	v_z	p	q	r	ϕ	θ	ψ	ω_{Ti}	ω_{Qi}	δ_i
x_{ref}	0	0	0	0	0	0	0	0.3491	0	2.6670	2.6670	1.5708

Table 5-7: Step reference signal for the pitch-up scenario

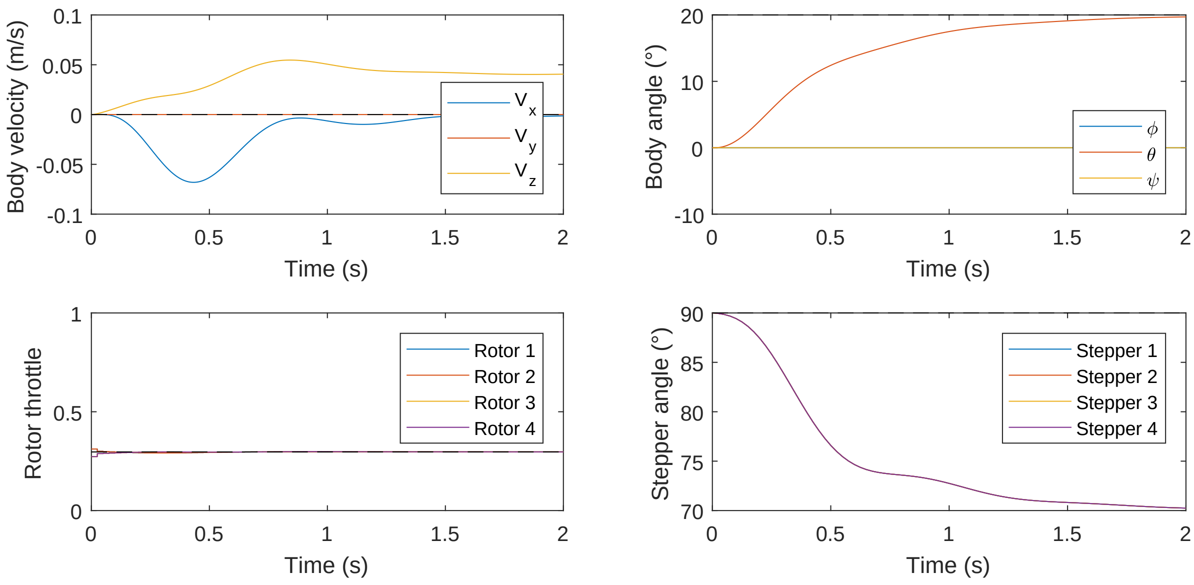


Figure 5-11: Body velocities, angles and control inputs for fixed-model MPC in a pitch-up scenario

Similar to the yaw scenario, the fixed-model controller achieves fast and stable tracking performance within seconds. The stepper motors tilt forward to an angle of 70 degrees, so that the thrust is perfectly countering the gravitational force. During its maneuver, the drone generates a small backwards motion, which is later canceled out. It ends the simulation with a small steady-state vertical velocity error of 4 cm/s downward.

The adaptive controller shows the recognizable oscillatory behavior, but achieves a stable position within 2 seconds as well. The time-varying controller shows a reduction of the oscillation, but similarly converges to a stead-state error. The only improvement over fixed-model MPC is a slight reduction in settling time.

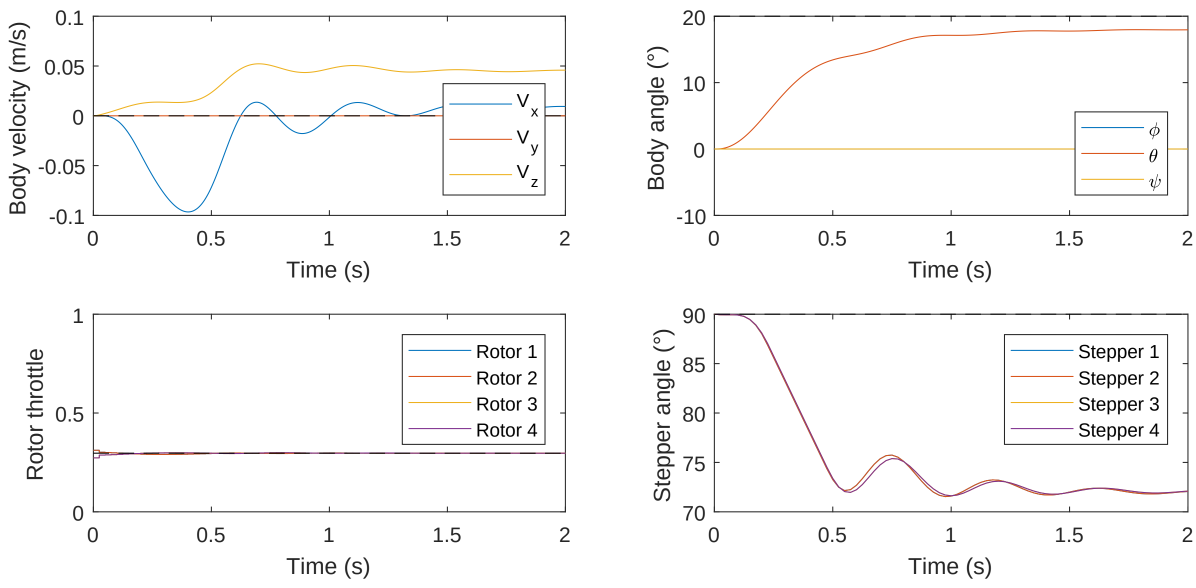


Figure 5-12: Body velocities, angles and control inputs for adaptive MPC in a pitch-up scenario

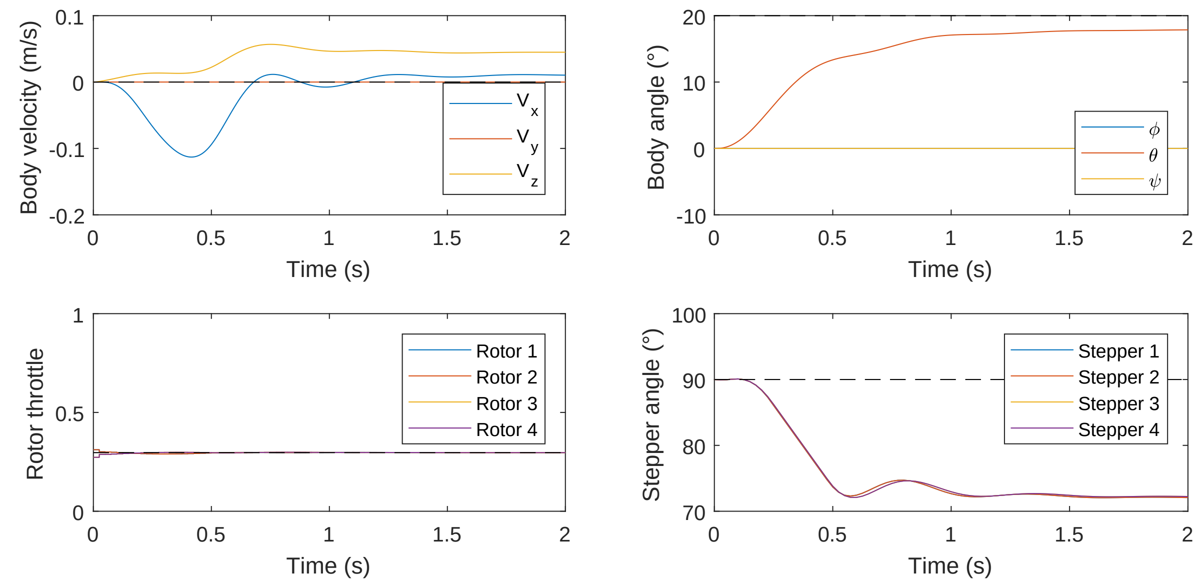


Figure 5-13: Body velocities, angles and control inputs for TVMPC in a pitch-up scenario

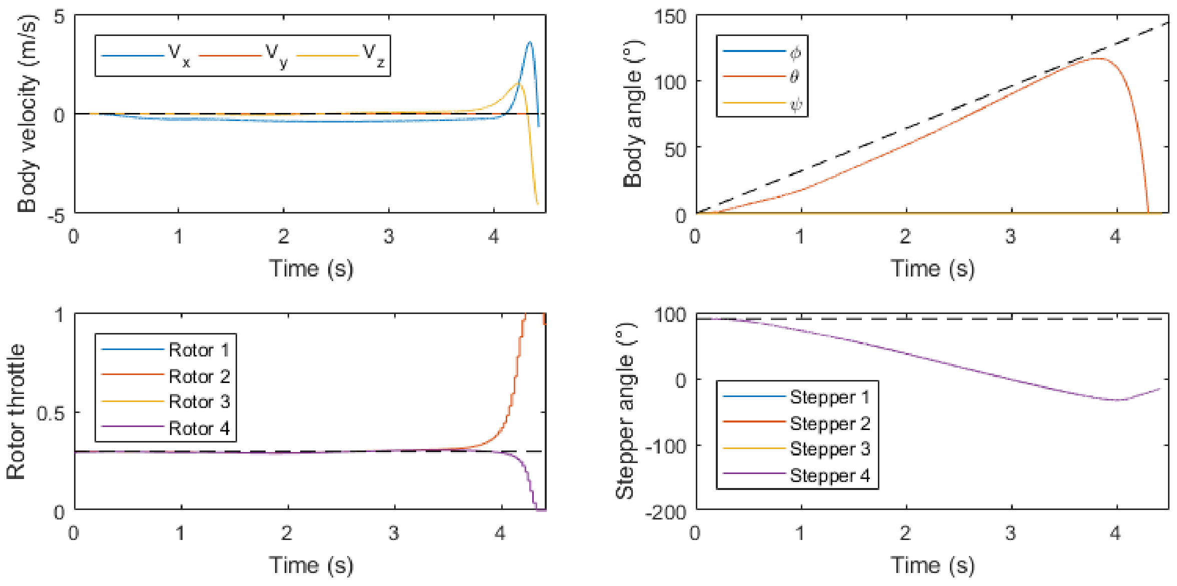


Figure 5-14: Body velocities, angles and control inputs for fixed-model MPC in a partial backflip scenario

5-1-5 Partial backflip

The vehicle is commanded to keep pitching up at a rate of 32°/s until instability occurs or flying completely upside down. For the last scenario, the control weights are re-tuned to the weights of Table 5-8, to slow the stepper motor tilting.

R	v_{2i}	u_{Mi}
Weight	0.01	0.10

Table 5-8: MPC control penalty weights

The fixed-model controller shows good tracking performance up until the upright position of 90° (Figure 5-14). At that point, the rotor throttles can be seen to abruptly increase, destabilizing the system. The fixed model assumes a thrust increase on the forward rotors to generate a pitch increase, and a thrust increase on the rear rotors to generate a pitch decrease. But when the vehicle has tilted past 90°, all rotors are essentially pointing downward with respect to the body frame, resulting in exactly the opposite behavior, resulting in a wrong optimization result and system destabilization.

The adaptive controller has little trouble approaching the upright position (Figure 5-15). With some small initial oscillations, the pitch angle is finally tracked and well and the aircraft hovers upside down without major body velocity errors.

The time-varying controller also makes it past the upright position towards full upside-down flying (Figure 5-16). The additional prediction models compared to the adaptive controller do not prove significantly beneficial compared to the time-varying controller, as the trajectory graphs are visually very similar. The oscillations of the pitch angle are lightly dampened,

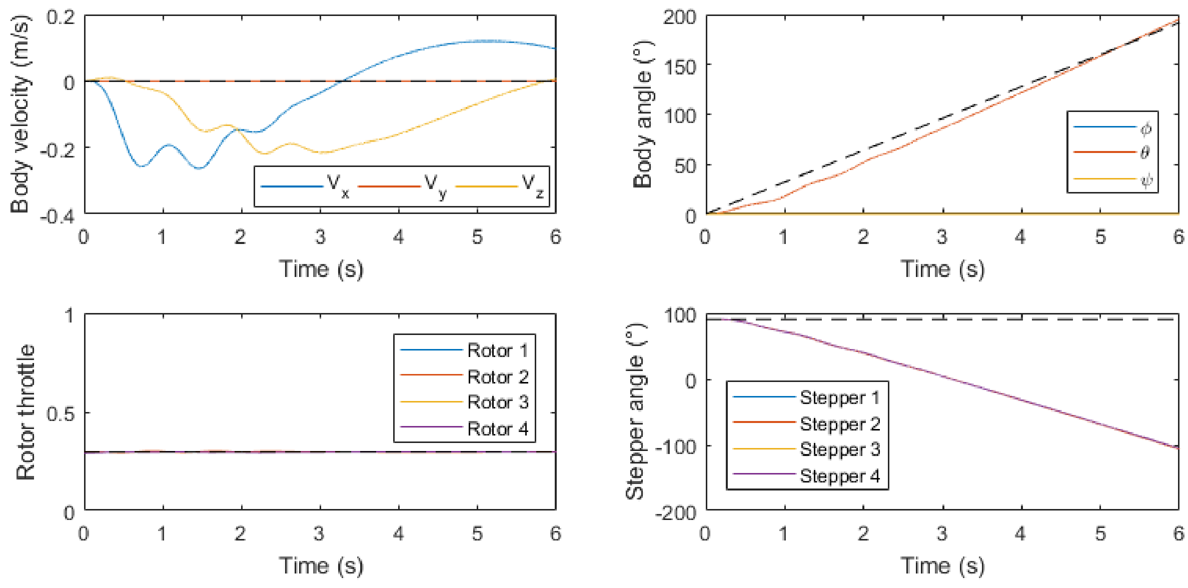


Figure 5-15: Body velocities, angles and control inputs for adaptive MPC in a partial backflip scenario

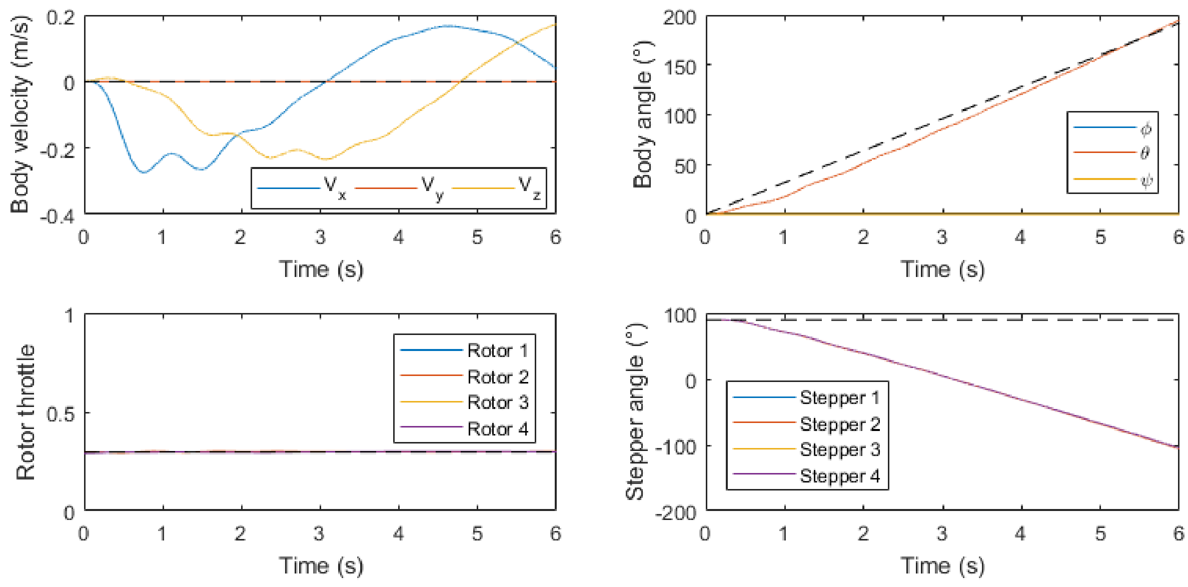


Figure 5-16: Body velocities, angles and control inputs for TVMPC in a partial backflip scenario

Coefficient	Value
τ	0.025
c_T	1.33
c_Q	0.133
m	4
I	I_3

Table 5-9: Co-simulation parameters

when compared with the trajectory of the adaptive controller - its variance is 1% less (1.0635 vs 1.0748). On the other hand, vertical velocity tracking has suffered.

5-2 Hardware-in-the-loop

The hardware-in-the-loop system was simulated using the parameters of Table 5-9 and controlled by Proportional Integral Derivative (PID) controllers operating at a rate of 40 Hz. Three flight scenarios are presented - altitude hold (Section 5-2-1), forward velocity control (Section 5-2-2) and yaw rate control (Section 5-2-3).

5-2-1 Altitude control

The system is restricted in its roll and pitch during simulation, as if moving along a vertical axis in the restricted flight experiment of Section 7-1. The system is initialized at its reference height $z_{ref} = 1.7$m and equilibrium throttle with zero vertical speed, and the vertical position error is passed to a PID controller copying the input to all four rotors. The PID controller is tuned at $K_P = 0.03$, $K_I = K_D = 0.05$.

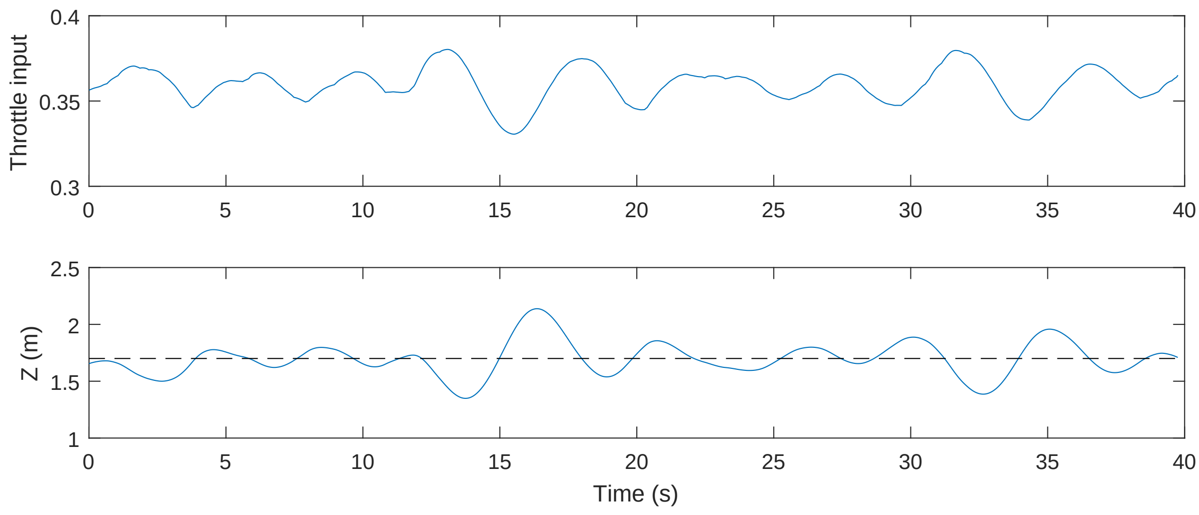


Figure 5-17: Real-time HIL altitude control on a restricted system

Altitude tracking is observed, but with significant disturbances (Figure 5-17). At times, the Rounds Per Minute (RPM) sensor signal, after filtering, still suffers from interference, showing large downward peaks in the measured rotational speed (Figure 5-18). This leads to a perceived thrust decrease in the model, prompting downward acceleration and an eventual increase in the controller output. After returning to undisturbed sensor values, the controller dampens the oscillation and returns to the tracking position.

The equilibrium rotational speed of $\omega = \frac{9.81}{1.333}$ is indicated in Figure 5-18 for reference, but is not a tracked value.

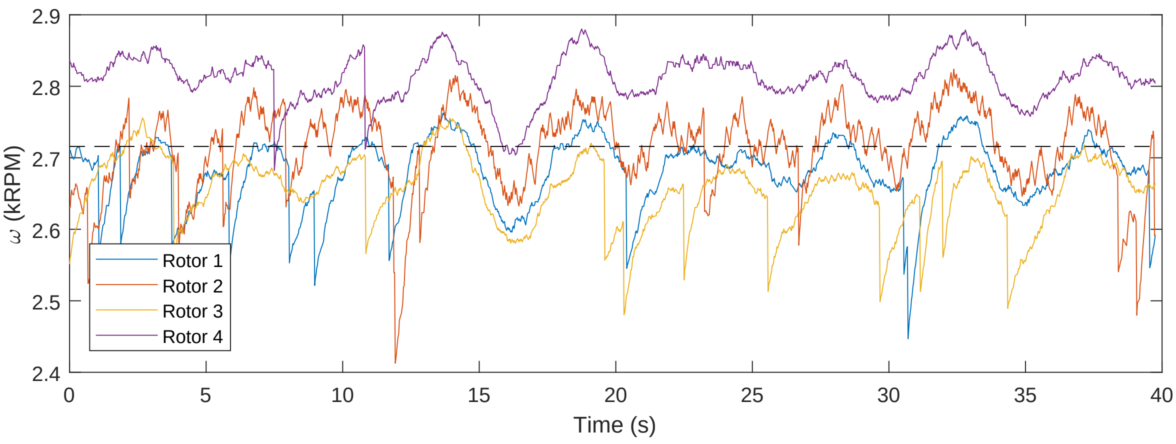


Figure 5-18: Filtered RPM sensor values during the hover test

5-2-2 Forward velocity control

The control is extended with a Proportional Integral (PI) controller providing equal input to all four stepper motors for forward body velocity tracking ($K_P = 50$, $K_I = 20$). The integrator is added as a means to compensate if the reference target is overshot, but is limited to avoid wind-up behavior. The pitch and roll angles are still fixed.

The tilt-rotor succeeds in tracking the forward velocity reference, but shows a persistent overshoot value (Figure 5-19). This is likely an effect of the integrator winding up quickly after the reference change has occurred, but not returning to compensate in the opposing direction fast enough. Removing the I control (making the controller proportional only) might reduce this behavior.

5-2-3 Yaw rate control

The control is extended with a PI controller providing differential signal input to the left and right stepper motors for yaw rate tracking ($K_P = 20$, $K_I = 40$). The pitch and roll angles are still fixed.

A similar behavior to the scenario of Section 5-2-2 occurs. The yaw rate is tracked, leading to gradual yaw changes. However, the overshoot effect is still present, leading to an orientation change of approximately a half-turn over the course of a minute.

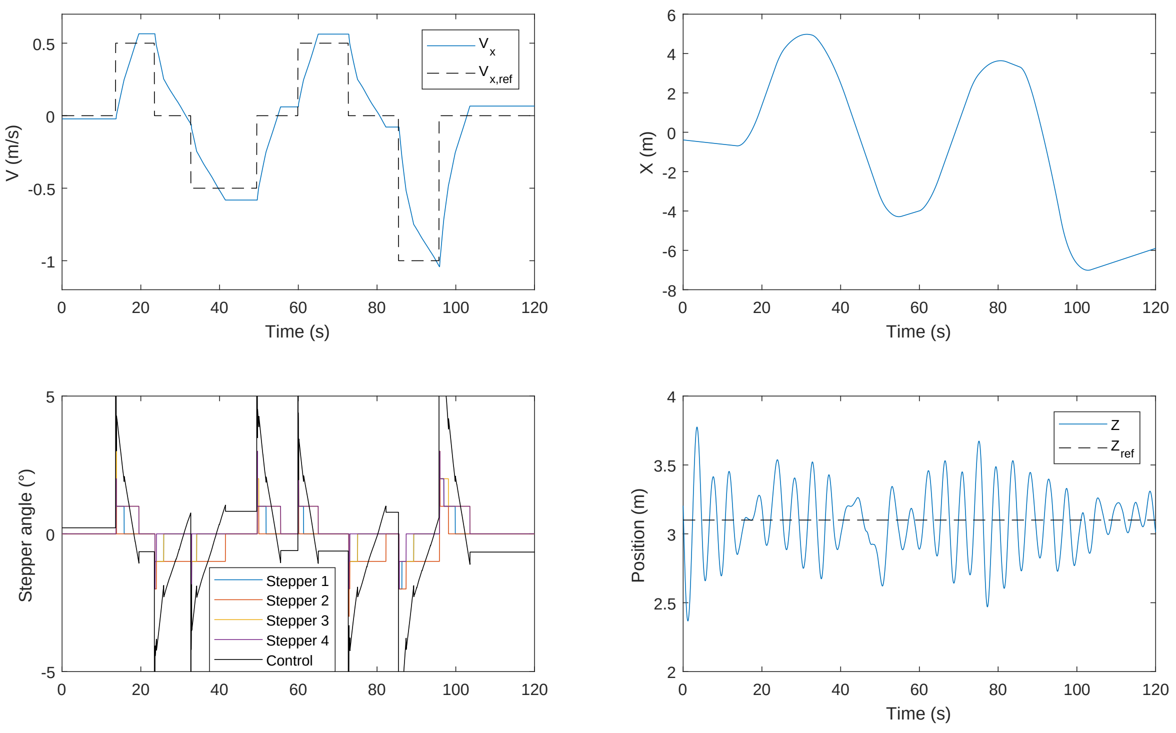


Figure 5-19: Real-time HIL forward velocity control on a restricted system

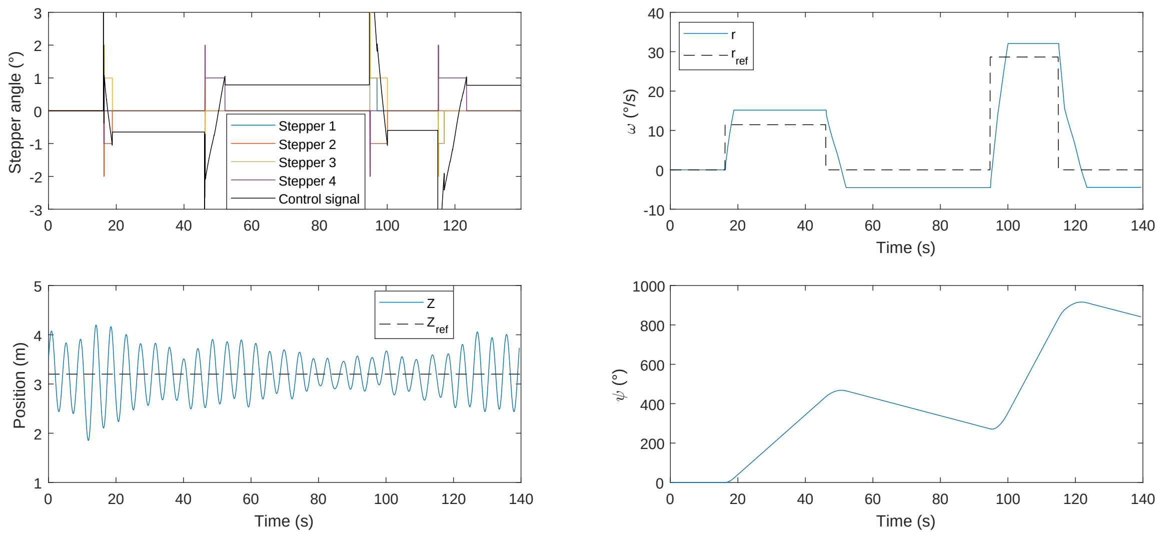


Figure 5-20: Real-time HIL yaw rate control on a restricted system

Chapter 6

Conclusions

6-1 TVMPC as a linear extension of adaptive MPC

The simulation of Time-Varying Model Predictive Control (TVMPC) on a small but highly non-linear system shows promising results, improving on fixed-model Model Predictive Control (MPC) and adaptive MPC for a sine-wave tracking scenario. For a prediction horizon of $N_P = 30$, a variation of the number of intermediate linearization models has been investigated, effectively ranging between adaptive MPC and the full TVMPC as proposed by this thesis. It shows that using a reduced set of models at the start of the prediction significantly improves on the tracking behavior of adaptive MPC, and performs equally well as having intermediate models over the full horizon.

The efficiency of quadratic optimization (as the time-varying models are still linear), together with the minimal increase in computational effort for linearizations, makes TVMPC a straightforward extension of adaptive control on non-linear systems. However, the optimal placement of intermediate models over the horizon, as well as optimal horizon length and time step remain topics for further research.

6-2 TVMPC on a tilt-rotor drone

Through the use of a rotor test bench, stepper motor modeling and estimating the remaining drone parameters, a full 21-state system with 8 inputs and 8 box constraints is developed. The non-linearities of the system are captured by augmenting the state space inputs and outputs with parametrized curve fits such as throttle mappings and quadratic thrust coefficients. The box constraints reflect physical limits on the rotor thrust and stepper rates.

In simulation, TVMPC does not significantly outperform fixed-model MPC in scenarios where the rotor thrust directions remain generally vertical. This is explained by the fact that the cosine of the tilt angles varies little with respect to a linear approximation for a large range of rotor tilt angles around $\pi/2$, hence the model variation and its potential benefits remain small. Weighting the stepper motor control also proves a difficult task, as the inputs are efficient at controlling yaw rate and pitch angle holding, but inefficient at pitch rate damping.

In highly non-linear situations such as hovering in a past-vertical orientation, TVMPC proves itself as a stable solution, where fixed-model control wrongfully estimates the control effectiveness. Although oscillations are slightly reduced in this scenario compared to adaptive MPC, the improvement is only marginal.

Caution should be exercised in selecting the prediction horizon and control weights, as adaptive and time-varying MPC are prone to oscillatory behavior and steady-state errors when the prediction models diverge heavily from the current state. A short prediction horizon, higher control penalty and small discretization time step succeed in dampening these oscillations, but also reduce the difference between the two approaches and ability to capture predicted non-linear effects.

6-3 Hardware-in-the-loop

A Hardware-in-the-Loop (HIL) system succeeds in co-simulating physical rotor speeds and tilt angles with a drone flying in a virtual environment which is rendered in real time. Using simple Proportional Integral Derivative (PID) control, the system is validated and stabilized for limited degrees of freedom. Even under heavy Rounds Per Minute (RPM) sensor disturbances, the physical system is controlled to stable hovering, forward velocity tracking and yaw rate tracking. Implementing TVMPC on the co-simulation setup remains a topic for further research. This topic, as well as a number of others are presented in Section 6-4 and Chapter 7.

6-4 Topics for further research

Several aspects of this thesis research project can be considered for further research.

- **Improve the rotor model**

  The thrust and torque of the rotor vary with respect to airspeed and altitude (ground effect). The existing test bench is usable to measure these effects by placement in front of a simulated ground surface or within a wind tunnel.

- **Simulated failure scenarios**

  As box constraints can be updated in real time, loss of rotor thrust can be simulated and investigated by reducing the maximum output constraint. Additionally, stepper motors can be simulated to be "stuck" in a specific position.

- **Attitude in co-simulation**

  The simulated attitude can be physically reproduced by rotating the HIL system in a gimbal construction. This way, attitude feedback using an Inertial Measurement Unit (IMU) and compass can be added to the flight control system.

- **Quaternion attitude controller**

  Uniquely controlling roll, pitch and yaw at a singularity point (upright hover) is mathematically impossible and problematic for individually tuned PID controllers. Developing an integrated attitude controller using the quaternion orientation system may overcome this issue.

- **Optimal model placement**

  The number of additional linearization models and their placement along the horizon is a broad topic of research. Additionally, model selection may depend on predicted state evolution only (and not on number of time step), so that fewer models are used if the state is expected to remain within specific bounds.

- **Dynamical flight experiment**

  An intermediate step between HIL experiments and free flight is constrained flight. In a constrained flight experiment, controllers can be tuned without the risk of crashing or the need to replace batteries. Software and hardware work for this experiment was initiated but not concluded over the course of this thesis. More details are included in Chapter 7.

Chapter 7

Future work

The drone modeling and subsequent hardware tests were designed as part of a longer research plan that could not be fully carried out in this thesis. For the remaining work, a large part of the hardware design is finished, but only briefly mentioned here.

The controllers, once verified in simulation and in real-time on the test setup, are to be implemented on an on-board system and tested using a suspended drone in a motion-restricted flight arena (Section 7-1). The motion of the drone in the arena can be limited to two translational degrees of freedom and one rotational degree of freedom, or less, to facilitate tuning for specific maneuvers. The motion can be tracked by self-developed low-cost optical sensors, wirelessly broadcasting the translational movement to the control system (Section 7-2). Additionally, a wireless control box broadcasting the state of 8 toggle switches can be added to simulate failure scenarios, implement model changes or change reference values in real-time (Section 7-3). Finally, the flight arena can be placed in a wind tunnel, to test aerodynamic effects and simulate forward flight (Section 7-4).

7-1 Flight arena

A flight arena was designed and built to carry out dynamical flight tests with a reduced crash risk. With a cube form and outer dimension of 2.5x2.5x2.5m, the resulting box provides ample space for a flying system to lift off, translate forward and back, and perform yawing maneuvers (Figure 7-1). The design includes only 1 kg of moving parts in the forward direction, as the bearing connections are milled and lathed from aluminium, the vertical axis is from carbon fiber and the sensor housings are 3D-printed plastic. In the vertical direction, the total mass of the altitude sensor including linear bearings is also kept under 1 kg.

The flying system to be suspended in the arena only needs to facilitate a 70x70x70mm sensor box in its center, and comprise attachments for linear bearings at the body axes.

This setup is sufficient to show the effects of rotor tilting in combination with differential thrust for a yaw maneuver, while having to overcome gravity. Additionally, the experiment is easily taken down and re-positioned on another face. In this fashion, other configurations can be flown, for example to test pitching in combination with forward motion, or perform a suspended barrel roll while having to compensate for forward and sideways motion.

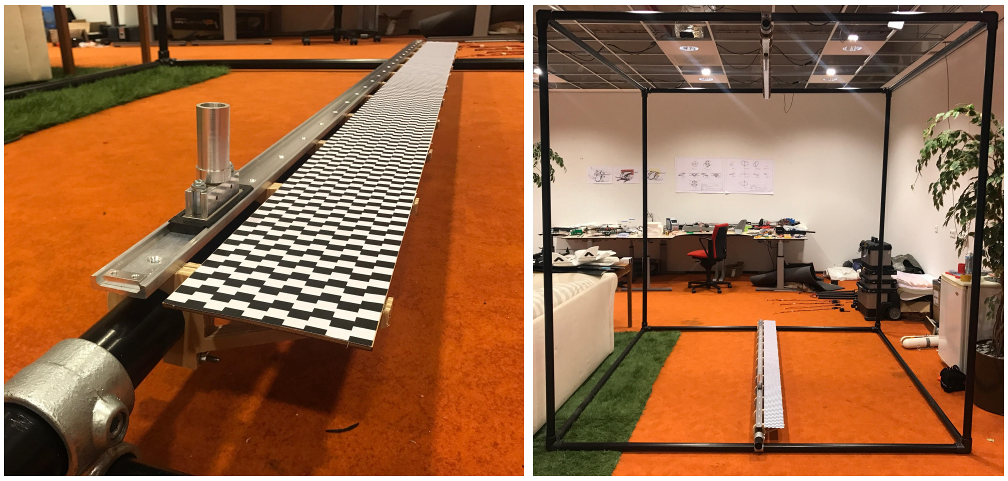


Figure 7-1: Left: Sensor track on the X axis of the flight experiment. Right: Partially constructed flight arena

Bottom plane	Drone orientation	Translation 1	Translation 2	Rotation	Self-lifting?
XY	X	X	Z	Yaw	Y
XY	Y	Y	Z	Yaw	Y
XZ	X	X	Y	Pitch	N
XZ	Z	X	Y	Roll	N
YZ	Y	Y	Z	Pitch	Y
YZ	Z	X	Z	Roll	Y

Table 7-1: Bottom plane configurations for the flight arena, drone orientations (world frame) and corresponding degrees of freedom (in body frame) for the suspended system

7-2 Optical sensors

Two types of optical sensors (COPS, Cheap Optical Position Sensors) are developed as a cheaper alternative for current tracking systems. Precision indoor positioning systems such as OptiTrack [26] easily cost thousands of euros, which can be prohibitive for small-scale research institutions or start-ups. The proposed alternative is based around multiple KY-033 sensors, commonly used in line follower robots to discriminate between a black line and white background, and is a low-friction, low-cost solution, both for flat surfaces and freely rotating axes.

To determine absolute position using N KY-033 sensors, a track of certain length needs to be divided in to 2^N segments which are coded into a binary number using a black and white pattern. For example, for a track of 2 meters, using 10 sensors, a resolution of 2 mm can theoretically be obtained.

The low-cost KY-033 is however only suited to determine differences between larger patches of light and dark, thus limiting the resolution in absolute position.

Using larger light and dark patches, for example of 10mm in length, and the same number of sensors, it is however possible to create a relative position sensor with a resolution of 1 mm. A staggered pattern design aims to reduce cross-lane interference between adjacent sensors.

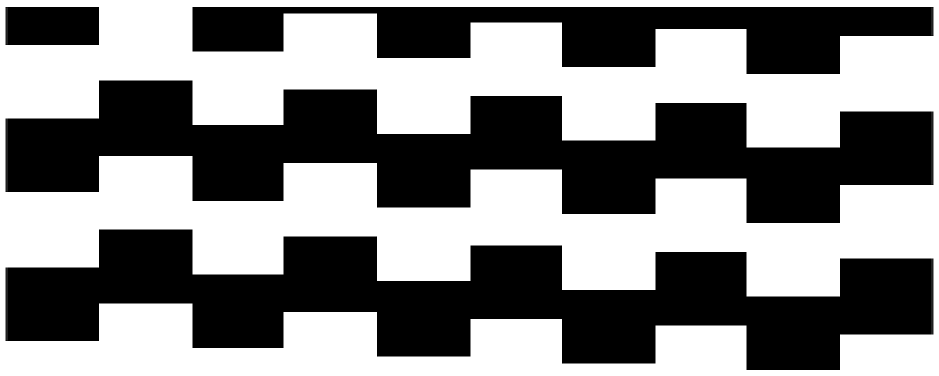


Figure 7-2: Relative linear position pattern for a 10-sensor tracking device - flat surface solution

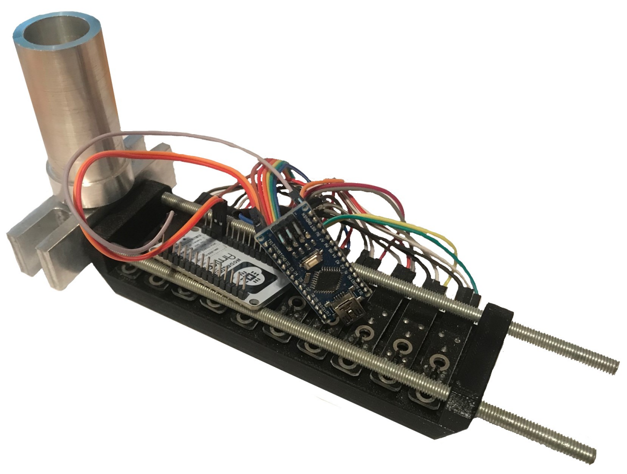


Figure 7-3: COPS (Cheap Optical Position Sensor) - linear version

This linear pattern is easily printed and glued to a board running along the length of a track. The position sensor is an Arduino system counting up or down according to the detecting pattern shift using low-level bit operations (Figure 7-3.

For position detection along a rotating axis (or a drone rotating around a fixed axis), an axis-symmetric pattern is used (Figure 7-4), so that the orientation changes do not influence the position detection along the length of the axis. Here, the sensors themselves are mounted in a staggered fashion with steps of 1 mm (Figure 7-5.

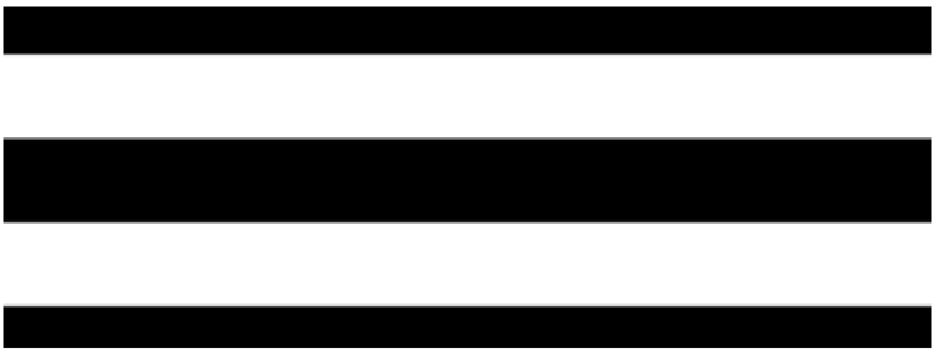


Figure 7-4: Relative linear position pattern for a 10-sensor tracking device - freely moving axis solution

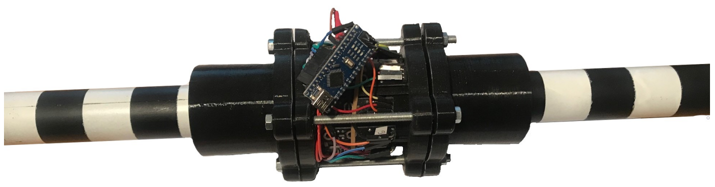


Figure 7-5: COPS (Cheap Optical Position Sensor) - rotating version

7-3 Wireless control box

A control box for use with the flying arena was developed and built to generate simulated actuator failures. The control box contains 8 toggle switches which can be mapped to constraint changes for throttle inputs or stepper angle limits, or any other mapping of 256 states to discrete system changes, such as model changes, input failures or reference positions.

The box is powered by a small 5V power bank and contains a NodeMCU microcomputer, which is programmed using the regular Arduino Integrated Development Environment (IDE) and contains a WiFi chip to broadcast the toggle switch states via User Datagram Protocol (UDP) packages to a dedicated topic on the Robot Operating System (ROS) master.

7-4 Wind tunnel testing

In a completed flight arena with tracking sensors, aerodynamic effects on the drone can be modeled during repeatable experiments. For this end, a wind tunnel with variable speed control was newly built. The wind tunnel is fitted with an array of parallel horizontal tubes to increase flow laminarity, and a funnel to further increase flow speed.

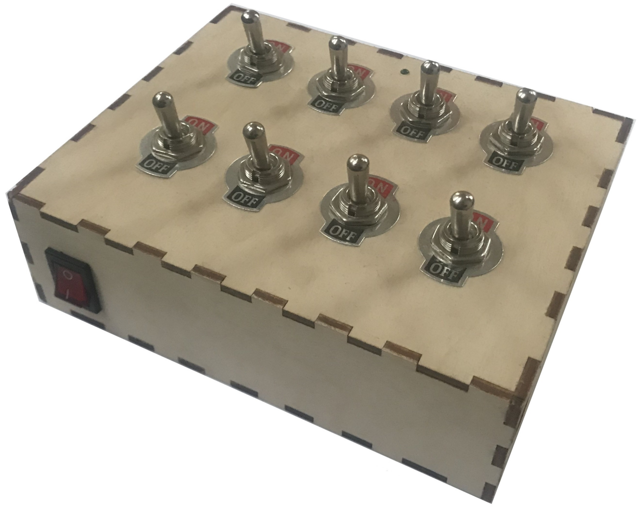


Figure 7-6: Wireless control box comprising 8 toggle switches

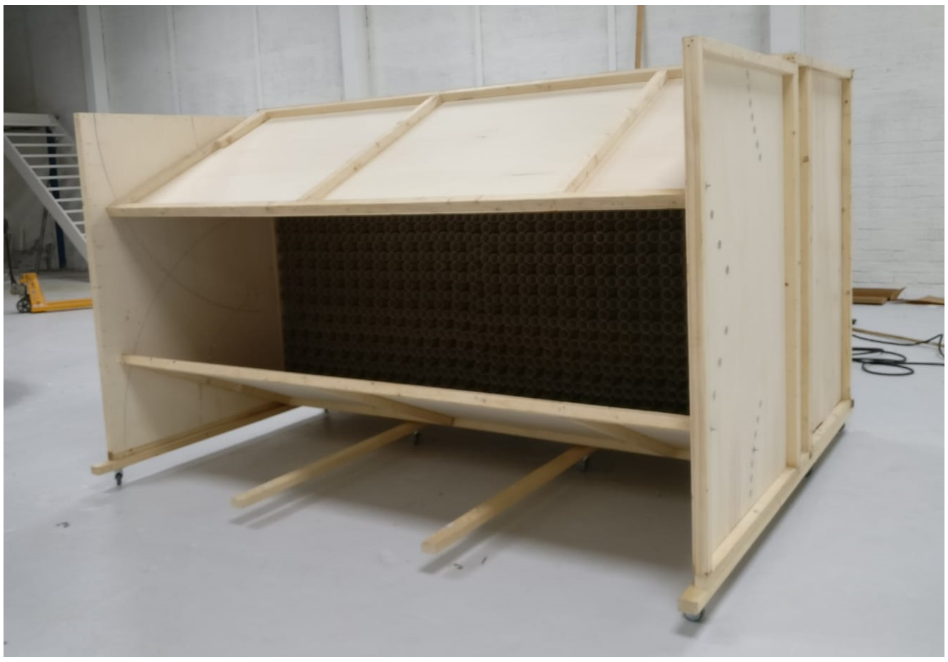


Figure 7-7: Experimental wind tunnel in the machining hall of Avy

Appendix A

Mathematical derivations and definitions

A-1 Time-varying MPC

A-1-1 State space discretization

Discrete-time state space models can be derived from continuous-time state space models by assuming the system input is constant during a given sample period. The continuous-time solution for

$$\dot{x}(t) = A_C x(t) + B_C u(t) + C_C, \qquad x(0) = x_0$$

is

$$x(t) = e^{A_C t} x_0 + \int_0^t e^{A_C (t-s)} (B_C u(s) + C_C) ds \tag{A-1}$$

with x_0 the initial state and s an integration variable. Replacing the system input u by a zero-order hold which is constant for a sample period τ, we can obtain the following solution:

$$\begin{aligned} x(\tau) &= e^{A_C \tau} x_0 + \int_0^\tau e^{A_C(\tau - s)} (B_C u(0) + C_C) ds \\ x(\tau) &= e^{A_C \tau} x_0 + e^{A_C \tau} \int_0^\tau e^{-A_C s} ds (B_C u(0) + C_C) \\ x(\tau) &= e^{A_C \tau} x_0 + A_C^{-1} (e^{A_C \tau} - I)(B_C u(0) + C_C) \end{aligned} \tag{A-2}$$

under the condition that A_C is invertible. From here it is clear that we can write for every sampling step

$$x_{k+1} = A_D x_k + B_D u_k + C_D \tag{A-3}$$

using

$$\begin{aligned} A_D &= e^{A_C \tau}, \\ B_D &= A_C^{-1}(A_D - I)B_C, \\ C_D &= A_C^{-1}(A_D - I)C_C. \end{aligned} \tag{A-4}$$

If A_C is non-invertible, the discrete-time matrices can still be computed using the matrix exponential (see MATLAB's c2d function).

A-1-2 Prediction matrix entries

The prediction matrix $\tilde{A}$ of size $N_P \times N_P$ is a lower triangular matrix with its diagonal equal to blocks of $N_S \times N_S$ identity matrices, where N_S is the state dimension. The remaining blocks are multiplications of state matrices, depending uniquely on the number of different prediction models used, and at what horizon a model switch is implemented.

For a single (non-time-varying) prediction model, the prediction matrix $\tilde{A}$ only contains N_P-1 unique submatrices that are not zero or an identity matrix. The full strictly lower triangle is composed of entries already listed on the leftmost column. For a 6-step ahead prediction, this looks like

$$\tilde{A} = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{A} & I & 0 & 0 & 0 & 0 \\ \boldsymbol{A^2} & A & I & 0 & 0 & 0 \\ \boldsymbol{A^3} & A^2 & A & I & 0 & 0 \\ \boldsymbol{A^4} & A^3 & A^2 & A & I & 0 \\ \boldsymbol{A^5} & A^4 & A^3 & A^2 & A & I \end{bmatrix},$$

where the unique entries are bold faced. Mathematically, the amount of reduction was

$$N_R = \underbrace{\frac{5 \cdot 6}{2}}_{\text{full triangle size}} - \overbrace{5}^{\text{left column size}} = 10. \tag{A-5}$$

If a model switch is added halfway along the prediction horizon, the amount of reduction is relatively smaller. It can best be visualized by a larger matrix of size $N_P = 10$. In the matrix

$$\tilde{A} = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{A_1} & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{A_1^2} & A_1 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{A_1^3} & A_1^2 & A_1 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{A_1^4} & A_1^3 & A_1^2 & A_1 & I & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{A_2A_1^4} & \boldsymbol{A_2A_1^3} & \boldsymbol{A_2A_1^2} & \boldsymbol{A_2A_1} & \boldsymbol{A_2} & I & 0 & 0 & 0 & 0 \\ \boldsymbol{A_2^2A_1^4} & \boldsymbol{A_2^2A_1^3} & \boldsymbol{A_2^2A_1^2} & \boldsymbol{A_2^2A_1} & \boldsymbol{A_2^2} & A_2 & I & 0 & 0 & 0 \\ \boldsymbol{A_2^3A_1^4} & \boldsymbol{A_2^3A_1^3} & \boldsymbol{A_2^3A_1^2} & \boldsymbol{A_2^3A_1} & \boldsymbol{A_2^3} & A_2^2 & A_2 & I & 0 & 0 \\ \boldsymbol{A_2^4A_1^4} & \boldsymbol{A_2^4A_1^3} & \boldsymbol{A_2^4A_1^2} & \boldsymbol{A_2^4A_1} & \boldsymbol{A_2^4} & A_2^3 & A_2^2 & A_2 & I & 0 \\ \boldsymbol{A_2^5A_1^4} & \boldsymbol{A_2^5A_1^3} & \boldsymbol{A_2^5A_1^2} & \boldsymbol{A_2^5A_1} & \boldsymbol{A_2^5} & A_2^4 & A_2^3 & A_2^2 & A_2 & I \end{bmatrix}$$

the unique entries are bold faced again. The appearance of two triangles with non-unique entries has is caused by the addition of a linearization point, and there is a large block in the lower left corner comprising only unique entries. The reduction here is

$$N_R = \underbrace{\frac{4 \cdot 5}{2}}_{\text{upper triangle size}} - \overbrace{4}^{\text{upper triangle height}} + \underbrace{\frac{5 \cdot 6}{2}}_{\text{lower triangle size}} - \overbrace{5}^{\text{lower triangle height}} = 6 + 10 = 16. \tag{A-6}$$

To introduce a formula for this reduction, we consider the prediction horizon per model. Let there be $N_L \geq 2$ models with respective prediction horizons $N_{P_L} \geq 1$ such that the sum of individual prediction horizons equals the total horizon $\sum_{i=1}^{N_L} N_{P_i} = N_P$. The number of unique entries N_U equals

$$N_U = N_{P_1} - 1 + \sum_{i=1}^{N_L - 1} (N_{P_{i+1}} \cdot \sum_{j=1}^{i} N_{P_j}) \tag{A-7}$$

To return to the remark made in Section 2-3-3, take $N_L = 2$, N_P even and $N_{P_1} = N_{P_2} = N_P/2$. The ratio of unique entries over total entries then equals

$$\begin{aligned} \eta &= 2\frac{N_U}{N_P \cdot (N_P - 1)} \\ &= 2\frac{N_{P_1} - 1 + N_{P_1}^2}{N_P^2 - N_P} \\ &= \frac{N_P - 2 + \frac{N_P^2}{2}}{N_P^2 - N_P} \\ &= \frac{2}{N_P} + \frac{1}{2 - 2N_P} + \frac{1}{2} \end{aligned} \tag{A-8}$$

and so

$$\lim_{N_P \to \infty} \eta = \lim_{N_P \to \infty} \frac{2}{N_P} + \lim_{N_P \to \infty} \frac{1}{2 - 2N_P} + \frac{1}{2} = \frac{1}{2}. \tag{A-9}$$

Appendix B

Coordinate framework

B-1 Reference frames

A minimum of two reference frames is used to describe the motion of the aircraft. The base frame is the world inertial frame W, fixedly located on the earth surface with the x_W-axis pointed northwards, the y_W-axis pointed eastwards and the z_W-axis pointed downwards, hence constituting a right-handed coordinate system. The body frame B is fixed to the aircraft body, with the x_B-axis pointed forward, the y_B-axis pointed to the right (starboard) and the z_B-axis again pointed downward. These reference frames are depicted in Fig. B-1.

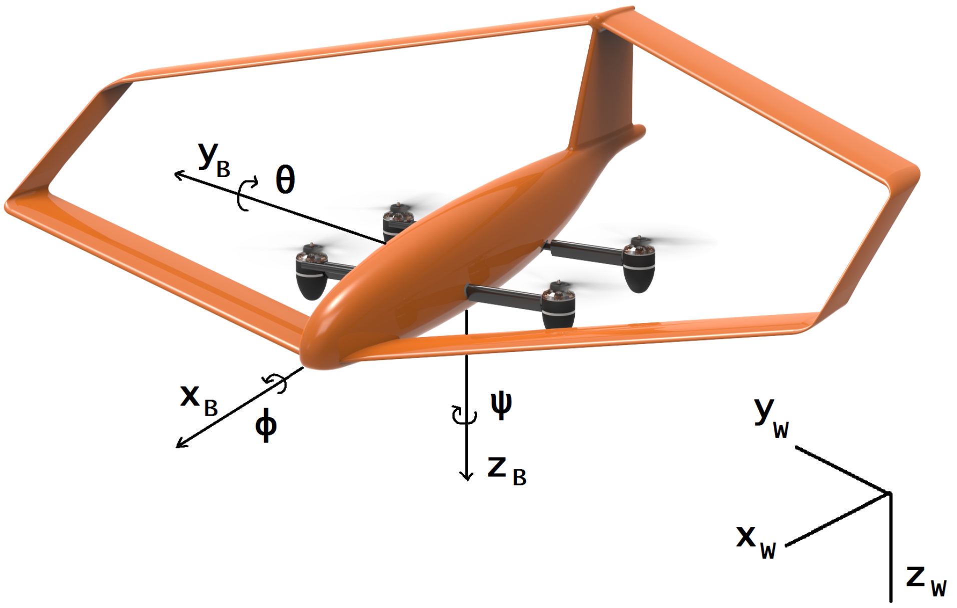


Figure B-1: Body and world reference frames

The world position is summarized in position vector $X_W = \begin{bmatrix} x_W & y_W & z_W \end{bmatrix}^T$, such that we can derive

$$V_W = \dot{X}_W = \begin{bmatrix} \dot{x}_W \\ \dot{y}_W \\ \dot{z}_W \end{bmatrix}, \tag{B-1}$$

where V_W is the velocity vector in the world frame. The speed in the body frame is denoted V_B, and can be calculated from the world frame velocity vector by means of

$$V_B = R_{BW} V_W, \tag{B-2}$$

where R_{BW} is a rotation matrix from the world frame (W) to the body frame (B). This rotation matrix is a member of $SO(3)$, the special orthogonal group, which uniquely describes all possible rotations in three-dimensional space. A special property of this group is that the inverse operation, i.e. a rotation in the opposite direction, corresponds to a transpose of the matrix. Mathematically,

$$R_{BW}^{-1} = R_{BW}^T = R_{WB}. \tag{B-3}$$

The rotation matrix can be specified using three or more parameters. The most common method of representation, Euler angles, is highlighted. An extensive work on representations and transformations between them can be consulted at [27].

B-1-1 Euler angle representation

Although a Euler angles representation may technically refer to a three-axis rotation in any order, in aerospace engineering the denomination refers to consecutive body-fixed rotations about a downward pointing z-axis, a starboard pointing y-axis and a forward-pointing x-axis. The angles corresponding to these respective axes are referred to as yaw (ψ), pitch (θ) and roll (ϕ), or collectively the attitude angles. The rotation order is also referred to as 3-2-1, in contrast to the common mathematical 3-1-3 rotation.

The rotation matrix corresponding to this representation is the combined matrix product of the three individual rotations, hence

$$\begin{aligned} R_{BW}(\phi,\theta,\psi) &= R_1(\phi)R_2(\theta)R_3(\psi) \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & -c_\theta s_\phi \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\theta c_\phi \end{bmatrix}, \end{aligned} \tag{B-4}$$

where BW denotes a rotation from world frame (W) to body frame (B) as before. Here the abbreviated notation c_α and s_α for $\cos(\alpha)$ and $\sin(\alpha)$ respectively, is adopted.

The body angular rate vector Ω_B is commonly specified as a set of derivatives from the Euler angles, so that

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \Omega_B. \tag{B-5}$$

Bibliography

[1] B. Lindenbaum, “V/STOL concepts and developed aircraft. Volume 1. A historical report (1940-1986),” tech. rep., Universal Energy Systems Inc., Dayton OH, 1986.

[2] A. S. Saeed, A. B. Younes, S. Islam, J. Dias, L. Seneviratne, and G. Cai, “A review on the platform design, dynamic modeling and control of hybrid UAVs,” in *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pp. 806–815, IEEE, 2015.

[3] J. de Jong, “Model-based Flight Control for a VTOL Aircraft with Independently Tilting Rotors,” Literature Survey, Delft University of Technology, Delft, The Netherlands, 2018.

[4] J. Wolkovitch, “The joined wing - An overview,” *Journal of Aircraft*, vol. 23, no. 3, pp. 161–178, 1986.

[5] K. J. Astrom and L. Rundqwist, “Integrator windup and how to avoid it,” in *American Control Conference, 1989*, pp. 1693–1698, IEEE, 1989.

[6] L. Lee and M. Spillman, “Control of slowly varying LPV systems-An application to flight control,” in *Guidance, Navigation, and Control Conference*, p. 3805, 1996.

[7] Z. Liu, Y. He, L. Yang, and J. Han, “Control techniques of tilt rotor unmanned aerial vehicle systems: A review,” *Chinese Journal of Aeronautics*, vol. 30, no. 1, pp. 135–148, 2017.

[8] C. Papachristos, K. Alexis, and A. Tzes, “Linear quadratic optimal trajectory-tracking control of a longitudinal thrust vectoring-enabled unmanned Tri-TiltRotor,” in *Proceedings, IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pp. 4174–4179, 12 2013.

[9] C. Papachristos, K. Alexis, and A. Tzes, “Dual–authority thrust–vectoring of a tri–tiltrotor employing model predictive control,” *Journal of intelligent & robotic systems*, vol. 81, no. 3-4, pp. 471–504, 2016.

[10] B. Kim, B. Kim, and N. Kim, “Trajectory tracking controller design using neural networks for a tiltrotor unmanned aerial vehicle,” *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 224, no. 8, pp. 881–896, 2010.

[11] D. Joosten, T. J. van den Boom, and T. Lombaerts, “Fault-tolerant control using dynamic inversion and model-predictive control applied to an aerospace benchmark,” *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 12030–12035, 2008.

[12] Z. Liu, Y. He, L. Yang, and J. Han, “Control techniques of tilt rotor unmanned aerial vehicle systems: A review,” *Chinese Journal of Aeronautics*, vol. 30, no. 1, pp. 135 – 148, 2017.

[13] H. Lee, S. Jung, and D. H. Shim, “Vision-based UAV landing on the moving vehicle,” in *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pp. 1–7, IEEE, 2016.

[14] N. Tijtgat, W. Van Ranst, B. Volckaert, T. Goedemé, and F. De Turck, “Embedded real-time object detection for a UAV warning system,” in *ICCV2017, the International Conference on Computer Vision*, pp. 2110–2118, 2017.

[15] A. Bemporad, “Linear Time Varying Nonlinear MPC.” `http://cse.lab.imtlucca.it/~bemporad/teaching/mpc/imt/2-ltv_nl_mpc.pdf`, 2018. Lecture Notes, IMT Lucca. Accessed 09-01-2019.

[16] C. Satzger, R. de Castro, and A. Knoblach, “Robust linear parameter varying model predictive control and its application to wheel slip control,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 1514 – 1520, 2017. 20th IFAC World Congress.

[17] T. Broomhead, C. Manzie, R. Shekhar, M. Brear, and P. Hield, “Robust stable economic mpc with applications in engine control,” in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pp. 2511–2516, IEEE, 2014.

[18] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, “qpoases: A parametric active-set algorithm for quadratic programming,” *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[19] G. Guennebaud, B. Jacob, *et al.*, “Eigen v3.” http://eigen.tuxfamily.org, 2010.

[20] L. Zaccarian, “DC motors: dynamic model and control techniques,” *Lecture Notes, Roma, Italy*, 2012.

[21] R. E. Kalman *et al.*, “Contributions to the theory of optimal control,” *Bol. Soc. Mat. Mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[22] I. C. Cheeseman, P. D, W. E. Bennett, P. D, and W. E. Bennett, “The effect of ground on a helicopter rotor in forward flight,” *ARC R&M 3021*, 1955.

[23] J. Leishman, *Principles of Helicopter Aerodynamics*. Cambridge Aerospace Series, Cambridge University Press, 2002.

[24] R. Mahony, V. Kumar, and P. Corke, “Multirotor Aerial Vehicles,” *IEEE Robotics and Automation Society*, vol. 19, no. 3, pp. 20–32, 2012.

[25] R. Vepa, *Nonlinear Control of Robots and Unmanned Aerial Vehicles: An Integrated Approach.* CRC Press, 2016.

[26] OptiTrack, “Build your own motion capture system.” https://optitrack.com/systems/#robotics/, 2018. Accessed: 2018-10-18.

[27] J. Diebel, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” *Matrix*, vol. 58(15-16), pp. 1–35, 2006.

Glossary

List of Acronyms

CNC	Computer Numerical Control
DC	Direct Current
EMF	Electromagnetic Force
ESC	Electronic Speed Controller
FCS	Flight Control System
GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
LiPo	Lithium Polymer
LPV	Linear Parameter Varying
LQR	Linear Quadratic Regulator
LTI	Linear Time Invariant
MPC	Model Predictive Control
NMPC	Non-linear Model Predictive Control
PI	Proportional Integral
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation

QP	Quadratic Programming
RMS	Root-Mean-Square
ROS	Robot Operating System
RPM	Rounds Per Minute
T3	Thrust and Torque Test
TVMPC	Time-Varying Model Predictive Control
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
USB	Universal Serial Bus
VTOL	Vertical Takeoff and Landing

List of Symbols

δ_i	Angle of motor i
Δ_M	Angular micro-stepping size
$\delta_{\text{ref},i}$	Reference angle of motor i
Ω_B	Body angular rates vector
ω_i	Rotor speed of DC motor i
ω_Q	Rotational speed with overshooting effect (four-parameter model)
ω_T	Rotational speed (four-parameter model)
ϕ	Roll attitude
ψ	Pitch attitude
ρ	Air density
$\sigma(.)$	Sign function
τ	Controller sampling time
τ_M	Motor controller loop time
θ	Pitch attitude
θ_i	Motor angle of DC motor i
$\tilde{A}$	Stacked state prediction matrix
$\tilde{B}$	Stacked control prediction matrix
$\tilde{C}$	Stacked vector of offset vectors
$\tilde{H}_u$	Stacked control constraint matrix
$\tilde{H}_x$	Stacked state constraint matrix
$\tilde{h}_{ul}$	Stacked lower matrix control constraint
$\tilde{h}_{uu}$	Stacked upper matrix control constraint

$\tilde{h}_{xl}$	Stacked lower matrix state constraint
$\tilde{h}_{xu}$	Stacked upper matrix state constraint
$\tilde{Q}$	Block diagonalization of Q_k's over the prediction horizon
$\tilde{R}$	Block diagonalization of R_k's over the prediction horizon
$\tilde{s}_{ul}$	Stacked lower simple control constraint
$\tilde{s}_{uu}$	Stacked upper simple control constraint
$\tilde{s}_{xl}$	Stacked lower simple state constraint
$\tilde{s}_{xu}$	Stacked upper simple state constraint
$\tilde{U}$	Stacked vector of future control inputs
$\tilde{X}$	Stacked vector of predicted states
$\tilde{X}_{\text{ref}}$	Stacked vector of state reference values
ξ_i	Spinning direction of rotor i
a	Model parameter (four-parameter model)
A_C	State matrix (continuous time)
A_D	State matrix (discrete time)
A_r	Rotor disk area
A_{L_i}	State matrix corresponding to linearization point i
b	Model parameter (four-parameter model)
B_C	Control matrix (continuous time)
B_D	Control matrix (discrete time)
b_f	DC motor friction coefficient
B_{L_i}	Control matrix corresponding to linearization point i
C_D	Offset vector (discrete time)
c_Q	Combined rotor torque coefficient
C_T	Rotor thrust coefficient
c_T	Combined rotor thrust coefficient
c_{AL}	Lower bound for the matrix constraint in optimization
c_{AU}	Upper bound for the matrix constraint in optimization
C_{L_i}	Offset vector corresponding to linearization point i
c_L	Lower bound for the vector constraint in optimization
c_U	Upper bound for the vector constraint in optimization
e_i	Back EMF of DC motor i
e_{PID}	Current Proportional Integral Derivative (PID) error value
e_{prev}	Previous PID error value
$f(.,.)$	State evolution function
$F_{g,B}$	Gravity force in the body frame
$F_{R,B}$	Total directed thrust
$F_{t,B}$	Total body force
g	Bias vector
H	Hessian matrix

I	Inertia matrix
i_i	DC motor current of motor i
i_{PID}	Current PID integral state
i_{prev}	Previous PID integral state
I_{xx}	Roll moment of inertia
I_{yy}	Pitch moment of inertia
I_{zz}	Yaw moment of inertia
J_R	Rotor inertia
K	DC motor constant (equal to K_t and K_e)
K_D	PID derivative gain
K_e	EMF constant
K_I	PID integral gain
K_P	PID proportional gain
K_t	DC motor torque constant
L	DC motor inductance
m	Vehicle mass
M_t	Total body torque
N_P	Prediction horizon
N_C	Control input dimension
N_S	State dimension
p	Roll rate
q	Pitch rate
Q_k	State weighting matrix at time k
Q_{Fi}	Torque induced moment of rotor i
$Q_{R,i}$	Rotor torque of rotor i
Q_{Ri}	Torque induced moment of rotor i
r	Rotor radius
r	Yaw rate
R_i	DC motor resistance of motor i
R_k	Control weighting matrix at time k
R_{BW}	Rotation matrix from world frame to body frame
$T_{M,i}$	DC motor torque of motor i
$T_{R,i}$	Rotor thrust of rotor i
u	Control input vector
u_L	Control vector linearization point
u_{PID}	PID control action
u_{k-1}	Previous control vector (discrete time)
v_1	Virtual throttle input (re-scaled around zero)
v_2	Virtual throttle input (reference kRPM input)
V_B	Velocity vector (body reference frame)

V_i	DC motor voltage of motor i
V_W	Velocity vector (world reference frame)
x	System state
x_0	Initial state
x_B	X coordinate (body reference frame)
x_k	Current state vector (discrete time)
x_L	State vector linearization point
X_W	Real-world position vector
x_W	X coordinate (world reference frame)
x_{ref}	State reference
x_{k-1}	Previous state vector (discrete time)
y	State space output
y_B	Y coordinate (body reference frame)
y_W	Y coordinate (world reference frame)
z	Optimization vector
z_B	Z coordinate (body reference frame)
z_W	Z coordinate (world reference frame)

Index

A

Adaptive MPC 6
altitude control 60
Avy 2

B

backflip experiment 58

C

cascaded control 3
command board 39
controllability 18

D

DC motor 18
discretization 8

E

Eigen 12
encoder 29
equations of motion 34
ESC 20
ESC calibration 23

F

filtering 40
forward speed control 61
four-parameter model 20
future research topics 64

G

gain scheduling 3
GUI 43

H

HIL 37

I

I2C 39
inertia matrix 33
integrator windup 3

L

linearization 7
LPV 6

M

micro-stepping 29
model reduction 10
MPC 6

N

NMPC 7
node 40
non-blocking control 31

O

observability 18
observer 42
outline 4

P

PID 42
pitch up experiment 56

Q

QP formulation 11
QP optimization 6
qpOases 11

R

rise time 27

ROS 39
rotor model 19
RPM sensor 22

S

sensor board 39
serial interface 38
simulator 42
state update equation 9

T

takeoff experiment 51
throttle curve 24
thrust 19
torque 19
trajectory prediction 7
TVMPC 7

V

virtual input 33
visualization 43

W

wind tunnel 70

Y

yaw right experiment 48