MULTI-CORE MODEL PREDICTIVE CONTROL STRATEGY FOR A TILT-ROTOR UAV IN SYSTEM-IN-THE-LOOP SIMULATION

Gabriela Maria Teixeira Miranda
Belo Horizonte, Brazil
2018
MULTI-CORE MODEL PREDICTIVE CONTROL STRATEGY FOR A TILT-ROTOR UAV IN SYSTEM-IN-THE-LOOP SIMULATION

Thesis submitted to the Graduate Program in Electrical Engineering of Escola de Engenharia at the Universidade Federal de Minas Gerais, in partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

Advisor: Guilherme Vianna Raffo
Co-Advisor: Janier Arias García

Belo Horizonte, Brazil
2018
To my family and my love for all
the supporting and strength.
Acknowledgements

I wish to thank all the people who have been in my life in this past two years. You all contributed and helped me a lot to my personal and professional growth.

Thanks to my boyfriend for understanding me, for the sacrifices made along the way and for giving me strength. Thanks to my family for being my support, safety, and comfort.

Specially I would like to thank my advisor Guilherme and co-advisor Janier for all the guiding and partnership. You pushed me to become always better. Thanks to the valuable advice that you gave.

Thanks to my lab colleagues for the companionship. I couldn’t be successful without this academic family. Brenner and Daniel, my big brothers, Alex, Ana, Antônio, Arturo, Diana, Ernesto, Edson, Fred, Fredy, Juancho, Letícia, Marcelo, Mariana, Rafael, and Stella, thank you for sharing not just a lab, but also your dreams, successes, frustrations, fears, and bravery to overcome the struggles. This journey was a lot more fun because of you.

Thanks to the members of MACRO and ProVANT for enriching experience of being in a group with elevated exchange of knowledge.

I would also like to thank the PPGEE members and the UFMG staff that provides a high quality of education and infrastructure that make capable researchers.
Resumo

Essa dissertação aborda a implementação de um controlador preditivo (MPC) embarcado, projetado para resolver o problema de rastreamento de trajetória de um veículo aéreo não tripulado (VANT) tilt-rotor. A arquitetura de software do MPC é desenvolvida para múltiplos núcleos visando melhorar o tempo de execução do controlador. Com esse objetivo, a viabilidade da paralelização do MPC invariante no tempo é investigada, obtendo resultados numéricos em um ambiente de simulação, no qual o MPC é implementado para controlar o VANT tilt-rotor no simulador ProVANT desenvolvido usando o arcabouço Robot Operating System (ROS) e o simulador Gazebo. Devido a melhoras obtidas no tempo de execução do MPC invariante no tempo, uma arquitetura de software para múltiplos núcleos é desenvolvida para o MPC variante no tempo. Essa arquitetura proposta é implementada no sistema embarcado do VANT formando um Processor-in-the-Loop (PIL) com o simulador do tilt-rotor. Através dessa abordagem, uma plataforma eficiente para desenvolvimento e testes é criada, facilitando a otimização do código e detecção de defeitos de software específicos da plataforma. As simulações PIL também permitem a análise de possíveis problemas relacionados a execução de múltiplas threads e compartilhamento de recursos. Assim, melhorias também foram encontradas no tempo de execução do MPC variante no tempo.
Abstract

This Master Thesis addresses the implementation of an embedded model predictive controller (MPC) designed to solve the path tracking problem of a tilt-rotor unmanned aerial vehicle (UAV). The MPC software architecture is designed for multiple cores to improve controller execution time. For this purpose, the feasibility of time-invariant MPC parallelization is investigated, obtaining numerical results in a simulation environment, in which the MPC is implemented to control the tilt-rotor UAV in the ProVANT simulator developed using the Robot Operating System (ROS) framework and the Gazebo simulator. Due to improvements obtained in time-invariant MPC runtime, a multi-core software architecture is developed for time-varying MPC. This proposed architecture is implemented in the UAV embedded system, creating a Processor-in-the-Loop (PIL) with the tilt-rotor simulator. Through this approach, an efficient platform for development and testing is created, facilitating code optimization and platform-specific software defect detection. PIL simulations also allow the analysis of potential problems related to multi-threaded execution and resource sharing. Thus, improvements were also found in time-variant MPC runtime.
## Contents

List of Figures xiii  
List of Tables xv  
Acronyms xvii  
Notation xix  

1 Introduction 1  
1.1 Motivation ................................................. 1  
1.2 State of the Art ............................................. 5  
1.2.1 Model Predictive Control to Tilt-Rotor UAVs ............... 5  
1.2.2 Embedded MPC ............................................. 5  
1.3 Justification ................................................. 6  
1.4 Objectives .................................................. 7  
1.4.1 Structure of the Text ..................................... 7  

2 Modeling and Control 9  
2.1 Dynamic Modeling .......................................... 9  
2.1.1 Generalized Coordinates ................................ 11  
2.1.2 Euler-Lagrange Formulation .............................. 11  
2.1.3 Vector of Generalized Forces .......................... 12  
  Propellers Forces ............................................. 13  
  Servomotors Forces .......................................... 13  
  Aerodynamic Forces ......................................... 14  
2.1.4 State-Space Representation .............................. 15  
2.2 UAV Linear Model .......................................... 16  
2.2.1 UAV Linear Time-Invariant Model ....................... 16  
2.2.2 UAV Linear Time-Varying Model ......................... 17  
2.3 Time-Invariant Model Predictive Control .................. 17  
2.4 Time-Varying Model Predictive Control .................... 21
3 Multi-Core Software Architecture

3.1 ProVANT Simulation Environment ................................................. 29
3.2 Multi-Core Software Approach to Time-Invariant MPC .................... 32
3.3 Processor-In-The-Loop .............................................................. 36
3.4 Multi-Core Software Architecture to Time-Varying MPC .................... 38
3.5 Chapter Summary ................................................................. 39

4 Simulation Results

4.1 System Parameters ................................................................. 41
4.2 Results from ProVANT Simulator ............................................... 45
4.3 Results from PIL Simulation ..................................................... 50

5 Conclusion

Bibliography
List of Figures

1.1 XV-15 Tilt rotor Aircraft (Courtesy of Nasa) . . . . . . . . . . . . . . . . . . 2
1.2 Model 1-G Tilt rotor Aircraft (Maisel et al. (2000)) . . . . . . . . . . . . . 2
1.3 Bell Boeing V-22 Osprey (Courtesy of Bell Helicopter) . . . . . . . . . . . . 3
1.4 ProVANT Tilt-rotor UAV 1.0 (Donadel (2015)) . . . . . . . . . . . . . . . . . 4
1.5 ProVANT Tilt-rotor UAV 2.0 (Rego (2016)) . . . . . . . . . . . . . . . . . . . 4
1.6 Tilt rotor UAV 3.0 (Queiroz (2014)) . . . . . . . . . . . . . . . . . . . . . . . 4

2.1 Tilt rotor UAV 3.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.2 Control inputs of the tilt-rotor UAV (Cardoso (2016)) . . . . . . . . . . . . 11
2.3 Frame position to obtain forward kinematics (Cardoso (2016)) . . . . . . 12

3.1 Provant Simulator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
3.2 Data Flow of ProVANT simulator . . . . . . . . . . . . . . . . . . . . . . . . . 31
3.3 Flowchart of the communication thread . . . . . . . . . . . . . . . . . . . . . . 33
3.4 Flowchart of the control thread . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.5 Profile of the TIMPC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.6 Parallel Regions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
3.7 Deployment Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
3.8 Flowchart of the communication thread on SoC . . . . . . . . . . . . . . . . 37
3.9 Processor-In-The-Loop Architecture . . . . . . . . . . . . . . . . . . . . . . . 38
3.10 Profile of the TVMPC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

4.1 Trajectory performed by the tilt-rotor UAV using the TIMPC . . . . . . . . 46
4.2 Tracking error of $x,y,z$ and $\psi$, using the TIMPC . . . . . . . . . . . . 46
4.3 Time evolution of $\phi,\theta$ and $\psi$, using the TIMPC . . . . . . . . . . . . 47
4.4 Time evolution of $\alpha_R$ and $\alpha_L$, using the TIMPC . . . . . . . . . . . 47
4.5 Time evolution of $u,v$ and $w$, using the TIMPC . . . . . . . . . . . . . . . 48
4.6 Time evolution of $p,q$ and $r$, using the TIMPC . . . . . . . . . . . . . . . 48
4.7 Time evolution of $\dot{\alpha}_R$ and $\dot{\alpha}_L$ states, using the TIMPC . . . . 49
4.8 Control signals generated by the TIMPC . . . . . . . . . . . . . . . . . . . . . 49
4.9 Wind speed . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
4.10 Time-Invariant MPC Runtime - Single-core ........................................ 50
4.11 Time-Invariant MPC Runtime - Multi-core ........................................ 51
4.12 Trajectory performed by the tilt-rotor UAV using the TVMPC. .............. 52
4.13 Tracking error of \(x,y,z\) and \(\psi\), using the TVMPC. ......................... 52
4.14 Time evolution of the \(\phi, \theta\) and \(\psi\) states, using the TVMPC. ........ 53
4.15 Time evolution of \(\alpha_R\) and \(\alpha_L\) states, using the TVMPC. ........... 53
4.16 Time evolution of the \(u,v\) and \(w\) states, using the TVMPC. ............. 54
4.17 Time evolution of the \(p,q\) and \(r\) states, using the TVMPC. ................ 54
4.18 Time evolution of \(\alpha_R\) and \(\alpha_L\) states, using the TVMPC. ........... 55
4.19 Control signals generated by the TVMPC. ........................................ 55
4.20 Wind speed. ......................................................................................... 56
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>UAV physical parameters</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Equilibrium Point of the Tilt Rotor UAV for the TIMPC</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Reference Values for the Linearized Model of the TVMPC</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>State Constraints of the Tilt Rotor UAV</td>
<td>44</td>
</tr>
<tr>
<td>4.5</td>
<td>Input Constraints Of The Tilt Rotor UAV</td>
<td>44</td>
</tr>
<tr>
<td>4.6</td>
<td>Desired Trajectory</td>
<td>45</td>
</tr>
</tbody>
</table>
Acronyms

3D Three-Dimensional
BH Belo Horizonte
MACRO Mechatronics, Control, and Robotics
ROS Robot Operating System
UFMG Universidade Federal de Minas Gerais
UAV Unmanned aerial vehicle
VTOL Vertical take-off and landing
MPC Model predictive control
TIMPC Time-Invariant Model predictive control
TVMPC Time-Varying Model predictive control
LMPC Linear model predictive controller
LMI Linear matrix inequalities
CAD Computer-Aided Design
SOC System-on-a-chip
SIL Software-In-The-Loop
HIL Hardware-In-The-Loop
ProVANT Projeto de Veículo Aereo Nao Tripulado
Notation

$\mathbb{R}$ Set of real numbers

e Error between the reference and actual values in the control law

$I$ Identity matrix

$0_{n \times m}$ Matrix of zeros with $n$ lines and $m$ columns

$R^A_B$ Rotation matrix of $B$ with respect to $A$

$R_{n,\alpha}$ Rotation of angle $\alpha$ about axis $n$

$(.)'$ Denotes the transpose of $(.)$

$R$ Jacobian matrix

$\dot{A}$ Time derivative of $A$

$A^T$ Transpose of $A$

$A^{-1}$ Inverse of $A$

Boldface italic letters represent matrices:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}, \ A \in \mathbb{R}^{n \times m}$$

$n_y$ Prediction horizon

$n_u$ Control horizon
1

Introduction

1.1 Motivation

Unmanned aerial vehicles (UAVs) have attracted enormous interest from military to commercial applications. The recent progress in technology resulted in size reduction of their components, greater accuracy of sensors and actuators, as well as cost lowering, making these type of systems more accessible and allowing the commercial production. Likewise, in the last decades, researches advanced in several fields, such as robotics, control, embedded systems, and path planning, improving the UAV performance. Consequently, their use has being expanded in a broad and far-reaching way, exploring mechanical and electronic advances, from which many new applications arisen, such as inspection of high voltage transmission line (Araar & Aouf (2014)), mining-related operations (Ge et al. (2016)), forest fire detection and monitoring (Ghamry et al. (2016)), wildfire detection (Lee et al. (2017)), navigation with collision avoidance (Sigurd & How (2003)), aerial assistance in search-and-rescue operations (Dhaliwal & Ramirez-Serrano (2009)).

Most of the studied UAVs are helicopters and fixed-wing vehicles. The helicopter configuration allows Vertical Take-Off and Landing (VTOL), and is capable of hovering, moving sideways, backwards, and changing direction quickly. On the other hand, fixed-wing aircrafts can perform forward flights with improved forward speed. A hybrid configuration of UAV, known as tilt-rotor UAV, combines the benefits of helicopters and fixed-wing aircrafts. This type of UAV receives special attention due to its capabilities of performing vertical take-off and landing as helicopters, as well as cruiser flights with the range and
speed of airplanes. The tilt-rotor UAV flies autonomously or teleoperated, offering less risk to human life, and has increased maneuverability provided by its configuration.

The first tilt-rotor aircraft built in history, capable of proving its desired concept, was the Bell XV-15, presented in Figure 1.1. The development of this aircraft was initiated in 1973. It combined the vertical and short take-off and landing (V/STOL) in the helicopter mode with the standard cruise flight in the airplane mode. These abilities allowed the XV-15 to deliver payloads in missions, with reduced fuel consumption.

Figure 1.1: XV-15 Tilt rotor Aircraft (Courtesy of Nasa).

Many attempts were made until the first flying tilt-rotor was able to perform VTOL and cruise flights (Maisel et al. (2000)). The Model 1-G, in Figure 1.2, was the first aircraft to explore the conversion between flight modes. However, it never successfully attained flights in the airplane mode.

Figure 1.2: Model 1-G Tilt rotor Aircraft (Maisel et al. (2000)).

The success in the tilt-rotor XV-15 research, influenced the production of the Bell’s V-22, show in Figure 1.3, the first production tiltrotor aircraft used to military transport.

Nowadays, some of the main issues involving the development process of a tilt-rotor UAV arises in designing the control strategies, as well as mapping properly its hardware/software architecture in the embedded system. The complexity in control area is attributed to
1.1. MOTIVATION

Figure 1.3: Bell Boeing V-22 Osprey (Courtesy of Bell Helicopter).

the high nonlinear behavior of the UAV, besides of unmodeled dynamics, parametric uncertainties, and aerodynamic disturbances they are usually subjected (Raffo (2011)). In addition, these systems are underactuated, i.e., there are more degrees of freedom than actuators. Therefore, advanced control techniques are often required to reach good performance in autonomous flight.

A suitable control strategy is the Model Predictive Control (MPC) (Camacho, E. F.; Bordons (2007)), that produces optimal response attending constraints by design. The significant performance of MPC attracted intense research consolidating the MPC theory. However, one of its main drawbacks is the computational cost of the algorithm, as for systems with fast dynamics such as a tilt-rotor UAV. Therefore, to control the tilt-rotor UAV by using an MPC algorithm, a suitable embedded system must be designed to deal with different system requirements, such as performance, low-power consumption, area, precision, reliability, among others. Furthermore, communication among sensors and actuators must be executed in such way that the data path delay among the system levels is reduced. Thus, the challenge is to find a proper trade-off among the system requirements to improve parameters like performance, area, etc.

In that context, in 2012, the Federal University of Minas Gerais (UFMG) and the Federal University of Santa Catarina (UFSC) started a project named ProVANT, aiming to study the related aspects of the design and control of Unmanned Aerial Vehicles.

The first researches developed in the project involved the tilt-rotor UAV 1.0, shown in Figure 1.4, which resulted in the works of Gonçalves (2014), Bodanese (2014), Neto (2014), and Donadel (2015).

Design modifications led to the tilt-rotor UAV 2.0, presented in Figure 1.5, to deal with load transportation tasks. This version resulted on research such as the work of Rego (2016).

Currently, the tilt-rotor UAV 3.0 is being developed at the ProVANT project. Its
mechanical design, proposed by Queiroz (2014), is shown in Figure 1.6. This prototype has improved aerodynamic fuselage and tail surfaces, which is composed by the vertical stabilizer and the rudder, and the horizontal stabilizer and the elevator, providing two more control inputs. Studies about adaptive control strategies for improved forward flight, dealing with wind effects and aerodynamic forces, are shown in Cardoso (2016).
1.2 State of the Art

This section presents a literature review on model predictive control approaches and techniques employed to improve the implementation of embedded MPCs.

1.2.1 Model Predictive Control to Tilt-Rotor UAVs

The predictive control philosophy leads to the optimal predict output over some finite horizon. Its control law are based on the prediction, computed using the system model. Then, an optimization problem, that measures the predicted performance is solved obtaining the current control input (Rossiter (2005)).

In recent years, few works have addressed the design of MPCs for controlling tilt-rotor UAVs. In Papachristos et al. (2011) is proposed an MPC for attitude control of a tilt-rotor UAV. A simplified model is obtained by Newton-Euler approach, and the control signals of the MPC affects only the orientation. The elevation thrust is given by a feedforward controller. This strategy allowed trajectory tracking while rejecting external disturbances.

Alexis et al. (2014) addresses the problem of robust model predictive control of rotorcrafts with disturbance avoidance. A minimum deviation from the reference is possible with the proposed receding horizon control strategy, which uses minimum peak performance as metric of optimality. Furthermore, it is proposed an augmentation of the framework to achieve obstacle avoidance.

In Santos & Raffo (2016a), an MPC is proposed to solve the path tracking problem of a tilt-rotor UAV carrying a suspended load. A multi-body dynamic model is obtained by means of the Euler-Lagrange formulation. The controller is designed based on the linearized error dynamics of the tilt-rotor UAV with suspended load. The input and state constraints used to compute the optimal solution are given by the physical limitations of the system. Then, Andrade et al. (2016) proposes a similar strategy with a robust terminal cost, which is included to guarantee stability and reduces the receding horizon, resulting in a successful controller to path tracking and disturbance rejection.

1.2.2 Embedded MPC

The research activities advances in embedded predictive control aiming to obtain a dependable MPC (Johansen (2017)). However, the application of real time embedded model predictive control has been a challenge in several studies. In Bleris et al. (2006), aiming to improve the computationally expensive performance of the controller, it was proposed reducing the precision of the microprocessor, hence increasing the optimization speed, and providing reduced power consumption and computational cost. In addition, the lower precision also reduced the overall chip area. In Bleris & Kothare (2005), with the PIL simulation, the authors could customize the optimization algorithm of the MPC with
the proposed low precision, obtaining the optimal parameters adapted to the used board.

Code parallelization could be used to improve the performance of the MPC algorithm by converting sequential code into multithreaded code, taking advantage of multi-core system architectures. The use of this kind of technique is leading to different applications, such as graph algorithms Lu et al. (2015), earliest deadline first schedulability (Chwa et al. (2013)), and quadratic problem solver (Joachim et al. (2012)).

On the other hand, several authors dealt with this issue through hardware acceleration on the MPC implementation. The Field Programmable Gate Array (FPGA) is a reconfigurable hardware often employed to run embedded MPCs, due to its customizability. Khusainov et al. (2016) defended the co-design for MPC with the FPGA, considering trade-offs between software and hardware through a multi-objective architecture. However, the time spent to implement the MPC code in hardware hinders this task. Thus, works as the one proposed by Suardi et al. (2015), developed a methodology to automatically implement the MPC code on FPGAs. In Currie et al. (2012), automatic code generation was proposed for fast embedded MPCs, directly from Matlab. Processor-in-the-loop demonstrations are also presented in this work.

1.3 Justification

This work addresses the challenges of designing a predictive control strategy suitable for the embedded systems of the tilt-rotor UAV. Some techniques have been studied to reduce the execution time of the MPCs, such as parallelization techniques, that have been applied in the solution of many problems exploring the potential of multi-core hardware.

The controllers for electronic systems are not a trivial problem, so, software design methodologies and testing procedures were developed with the goal of boosting the reliability of designs of embedded control. When these algorithms are developed, testing the software code, in the embedded target insures the correctness of the program. A procedure to develop, test and validate the code is the Processor-in-the-Loop (PIL) implementation (Lentijo et al. (2003)). This process allows to run the control strategy in the embedded system, controlling a virtual plant on the computer. The communication between this type of simulation is via serial link and does not run in real time, since the system model runs in a host computer without a multitasking real-time operating system. With this process, its possible adjust specifics configurations of the target processor and to obtain real time execution information. Another procedures that are used in the model based development is the Model-in-the-Loop (MIL), Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL). Usually, simulations stages evolve from MIL, an initial stage used to test the behavior of the model and control system in a simulation platform, SIL, a loop without any hardware that runs the controller code compiled by a locally-hosted compiler, to PIL and then HIL, where the plant model is simulated presenting the same
outputs given by the real plant and runs in real time in loop with the target hardware. With these techniques, it is possible to correct undesired behaviors on early stages of the application development, realize conformance testing, study the behavior of the model, the performance of control system and the communication protocols.

1.4 Objectives

The main objective of this master thesis is to develop a multi-core software architecture for an embedded model predictive controller based on the tilt-rotor UAV 3.0 (Figure 1.6). To reach this goal, some specific objectives are defined:

- Study and implement the dynamic model of the tilt-rotor UAV using the Euler-Lagrange formulation.
- Design time-invariant and time-varying model-based predictive control strategies to address the path tracking problem.
- Develop a multi-core architecture for both controllers, seeking time improvements along the code implementation.
- Test the designed architecture for the time-invariant MPC in a simulation environment that emulates a reliable representation of the tilt-rotor UAV.
- Use the processor-in-the-loop approach to develop, test and validate the multi-core software architecture of the time-varying MPC on the final target embedded system.

1.4.1 Structure of the Text

The first chapter of this work presented an introduction about the tilt-rotor UAVs, and embedded predictive controllers, motivating the study of techniques that address improvements on time performance of embedded MPCs. The next chapters are structured as follow:

- **Chapter 2** presents the system model, and describes the linear time-invariant predictive control (LTIMPC) and the linear time-varying predictive control (LTVMPMPC) design for path tracking of the tilt-rotor UAV.

- **Chapter 3** describes the parallelization of both MPC strategies, the multi-core architecture implemented on the simulator environment, and then its implementation on the final system to integrate the processor-in-the-loop.

- **Chapter 4** presents and discusses the simulation results.
• Chapter 5 summarizes the contributions and results presented in this work, and suggests possible future works in this research line.
This chapter presents the development of predictive controllers based on both time-invariant and time-varying models of the ProVANT tilt-rotor UAV 3.0. It starts by presenting the tilt-rotor UAV nonlinear equations of motion. Thus, these equations are linearized around an equilibrium point and the reference trajectory to obtain a linearized time-invariant model and a linearized time-varying model, used to design the LTIMPC and the LTVMPG, respectively. Finally, these controllers are designed to perform path tracking using a full-state feedback approach.

This section is structured as: in Section 2.1 the equations of motion of the ProVANT tilt-rotor UAV are presented; in Section 2.2 the nonlinear model is linearized around an equilibrium point, and also, around the trajectory, which in Section 2.3, the time-invariant model predictive controller based on the linearized time-invariant model of the tilt-rotor UAV is designed; in Section 2.4 the time-varying MPC is designed; finally, in Section 2.5 the controllers implementation in C++ language are presented.

2.1 Dynamic Modeling

The tilt-rotor UAV 3.0 was modeled in Cardoso (2016) and its dynamic nonlinear model is briefly presented in this section.

The tilt-rotor UAV is illustrated in Figure 2.1, which is composed by three rigid bodies:

- The main body - composed by the fuselage, made of acrylic material and gather the group of electronic devices embedded in the UAV, the empennage, and the
servomotors that tilt the thrusters;

- Two thrusters’ groups - composed by brushless motors connected to the propellers, one at each side of the aircraft and attached to the main body by revolute joints.

![Figure 2.1: Tilt rotor UAV 3.0](image)

The tail assembly has the horizontal and vertical stabilizers in a T-tail configuration providing better aerodynamic stability and improving the pith and yaw motion when comparing with UAVs without tail. Thanks to the tail surfaces, the forward flight is improved allowing better maneuverability.

In order to generate the movements, the aircraft is guided using as control inputs: the forces applied by the right and left thrusts, \( f_R \) and \( f_L \), respectively; the torques applied by the servomotors, \( \tau_R \) and \( \tau_L \); and the deflections \( \delta_e \) of the elevator and \( \delta_r \) of the rudder. The effectiveness of the elevator and rudder deflections to generate forces depend on the magnitude of the relative wind acting on the aircraft (some more details about the aircraft are illustrated in Figure 2.2).

During the flight, roll motion is obtained by producing different values of thrusts \( f_R \) and \( f_L \). Pitch motion is performed by regulating the deflection \( \delta_e \) and equally tilting \( \alpha_R \) and \( \alpha_L \), that are the rotation angles of the thrusters produced by the torques \( \tau_R \) and \( \tau_L \). Lastly, yaw motion is produced varying the deflection \( \delta_r \) and the angles \( \alpha_R \) and \( \alpha_L \) in opposite directions.
2.1. DYNAMIC MODELING

2.1.1 Generalized Coordinates

To attain the forward kinematics seven frames are used: the inertial frame $I$; the body frame $B$; the main body center of mass frame $C_1$; the right and left thrusters’ groups centers of mass frames, $C_2$ and $C_3$, respectively; and two auxiliary frames $C_{aux}^2$ and $C_{aux}^3$ on the rotation axis of the servomotors. The frame positions are illustrated in Figure 2.3.

The tilt-rotor UAV has eight degrees of freedom, being its configuration in space described by the following generalized coordinates vector

$$q = [x \ y \ z \ \phi \ \theta \ \psi \ \alpha_R \ \alpha_L]^T,$$

where $x$, $y$, and $z$ are the position of the aircraft body frame $B$ with respect to the inertial reference frame $I$; $\phi$, $\theta$, and $\psi$ are Euler angles using the roll, pitch, and yaw convention; $\alpha_R$ is the right servomotor angle; and $\alpha_L$ the left servomotor angle.

2.1.2 Euler-Lagrange Formulation

The equations of motion are obtained by the Euler-Lagrange formulation (Spong et al. (2006)), which can be written in the canonical form as

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = F(q, \dot{q}, u, \zeta),$$  \hspace{1cm} (2.1)
where \( M(q) \in \mathbb{R}^{8 \times 8} \) is called the inertia matrix, \( C(q, \dot{q}) \in \mathbb{R}^{8 \times 8} \) is the Coriolis and centrifugal forces matrix, \( G(q) \in \mathbb{R}^{8 \times 1} \) is the gravitational force vector, and \( \mathcal{F}(q, \dot{q}, u, \zeta) \in \mathbb{R}^{8 \times 1} \) is the vector of generalized forces. The vector of generalized forces is function of the generalized coordinates vector \( q \), its velocities \( \dot{q} \), the control inputs
\[
u = \begin{bmatrix} f_R & f_L & \tau_R & \tau_L & \delta_e & \delta_r \end{bmatrix}',
\]
and the disturbance vector produced by the environment wind
\[
\zeta = \begin{bmatrix} u_a & v_a & w_a \end{bmatrix}',
\]
where \( u_a, v_a \) and \( w_a \) are wind components expressed in the body frame.

2.1.3 Vector of Generalized Forces

The vector of generalized forces \( \mathcal{F}(q, \dot{q}, u, \zeta) \) can be decomposed in a sum of seven components:
\[
\mathcal{F}(q, \dot{q}, u, \zeta) = \mathcal{F}_{pR} + \mathcal{F}_{pL} + \mathcal{F}_{sR} + \mathcal{F}_{sL} + \mathcal{F}_f + \mathcal{F}_h + \mathcal{F}_v.
\]
in which \( \mathcal{F}_{pR} \) and \( \mathcal{F}_{pL} \) are forces related to the right and left thrusters, respectively; \( \mathcal{F}_{sR} \) and \( \mathcal{F}_{sL} \) are the forces related to the right and left servomotors; and \( \mathcal{F}_f, \mathcal{F}_h, \mathcal{F}_v \) are the aerodynamic forces related to the fuselage, horizontal stabilizer and vertical stabilizer, respectively.
Propellers Forces

Propellers create thrust forces and torques due to blade’s drag. Assuming the rotors in steady state, the drag torque generated by the propellers is given by $k_t b f$ (Castillo et al. (2005)), where $k_t$ and $b$ are aerodynamic constants obtained experimentally, and $f$ is the thrust. Aiming to reduce the yaw torque generated by the propellers, their rotations have opposite directions, the left one rotates clockwise and the right one counter-clockwise. Therefore, the generalized forces related to the propellers are written as follow:

$$
\mathbf{F}_{pR} = (J_{w_2})^T R_{c_2}^z \begin{bmatrix} 0 \\ 0 \\ f_R \end{bmatrix} + (J_{w_2})^T R_{c_2}^z \begin{bmatrix} 0 \\ 0 \\ \frac{k_t b f_R}{} \end{bmatrix},
$$

where $J_{w_2}$ and $J_{w_3}$ are the linear and angular velocity Jacobian matrices associated with the right thrusters’ group. Similarly, on the left side, the propeller contribution to the generalized forces is:

$$
\mathbf{F}_{pL} = (J_{w_3})^T R_{c_3}^z \begin{bmatrix} 0 \\ 0 \\ f_L \end{bmatrix} + (J_{w_3})^T R_{c_3}^z \begin{bmatrix} 0 \\ 0 \\ -\frac{k_t b f_L}{} \end{bmatrix},
$$

where $J_{w_3}$ and $J_{w_3}$ are the linear and angular velocity Jacobians related to the left thrusters’ group center of mass.

Servomotors Forces

The generalized forces produced by the servomotors are computed by

$$
\mathbf{F}_{sR} = (J_{w_2})^T R_{c_2}^{\tau_{aux}} \begin{bmatrix} 0 \\ \tau_R \\ 0 \end{bmatrix} + (J_{w_B})^T R_{c_2}^{\tau_{aux}} \begin{bmatrix} 0 \\ -\tau_R \\ 0 \end{bmatrix},
$$

$$
\mathbf{F}_{sL} = (J_{w_3})^T R_{c_3}^{\tau_{aux}} \begin{bmatrix} 0 \\ \tau_L \\ 0 \end{bmatrix} + (J_{w_B})^T R_{c_3}^{\tau_{aux}} \begin{bmatrix} 0 \\ -\tau_L \\ 0 \end{bmatrix}.
$$

Since the torques are applied by servomotors rigidly attached to the UAV, torque reactions are generated on the main body, which are represented by the second part of the summation. Expanding equations (2.5) and (2.6), one obtain

$$
\mathbf{F}_{sR} = \begin{bmatrix} 0_{6 \times 1} \\ \tau_R \\ 0 \end{bmatrix},
$$

$$
\mathbf{F}_{sL} = \begin{bmatrix} 0_{6 \times 1} \\ \tau_L \\ 0 \end{bmatrix}.
$$
Aerodynamic Forces

The aerodynamic forces acting on the tilt-rotor UAV were computed in Queiroz (2014), in which wind component was split in two planes of study: acting on x-z and x-y planes. Thereby, the generalized force $F_f$, related to the fuselage, is obtained by the following equation:

$$F_f = (J_f)^T R_b^\alpha \left\{ R_{y,\alpha_f} \begin{bmatrix} -\frac{1}{2} \rho (v_{air}^{xiz})^2 s_f c^{d}_{fzz} (\alpha_f) \\ 0 \\ \frac{1}{2} \rho (v_{air}^{xiz})^2 s_f c^{l}_{fzz} (\alpha_f) \end{bmatrix} + R_{z,-\beta_f} \begin{bmatrix} -\frac{1}{2} \rho (v_{air}^{xy})^2 s_f c^{d}_{fxy} (\beta_f) \\ \frac{1}{2} \rho (v_{air}^{xy})^2 s_f c^{l}_{fxy} (\beta_f) \\ 0 \end{bmatrix} \right\} + (J_f)^T R_b^\gamma \begin{bmatrix} c^e (\delta_e) \end{bmatrix} \right\}, \quad (2.9)$$

where $c^{d}_{fzz}$, $c^{l}_{fzz}$, $c^{d}_{fxy}$, and $c^{l}_{fxy}$ are, respectively, the drag and lift aerodynamic coefficients of the fuselage in x-z and x-y plane, which aggregate its aerodynamic characteristics. Besides, $J_f = \frac{\partial p_I^f}{\partial \dot{q}}$ is the Jacobian matrix related to the fuselage aerodynamic centers, which $p_I^f = d_{IB} + R_{IB} d_{Bf}$ is the position of the fuselage aerodynamic center with respect to the inertial frame.

The angle of attack

$$\alpha_b = -\tan^{-1}\frac{w_h - w_a}{u_b - u_a}$$

defines the orientation of the wind acting in x-z plane. It is the angle between the body frame line and the air-speed vector. The side slip angle

$$\beta_b = -\tan^{-1}\frac{v_h - v_a}{u_b - u_a}$$

is the angle between the body frame and the air-speed vector represented in the x-y body plane, that is, the orientation of the wind acting in x-y plane.

To calculate the vector of generalized force $F_h$ generated by the horizontal stabilizer, it is considered only the influence of wind speed in x-z plane $v_{air}^{xiz}$. Thus, the force is written as

$$F_h = (J_h)^T R_b^{\gamma} \left\{ R_{y,\alpha_h} \begin{bmatrix} -\frac{1}{2} \rho (v_{air}^{xiz})^2 s_h c^{d}_{hzz} (\alpha_h) \\ 0 \\ \frac{1}{2} \rho (v_{air}^{xiz})^2 s_h c^{l}_{hzz} (\alpha_h) \end{bmatrix} \right\}, \quad (2.10)$$

where $c^e (\delta_e)$ is a function of the elevator deflection angle $\delta_e$, responsible to map the elevator deflection influence on the aircraft forces, and $J_h = \frac{\partial p_I^h}{\partial \dot{q}}$ is the linear velocity Jacobian, with $p_h = d_{IB}^h + R_{IB}^h d_{Bh}^h$ being the position of the horizontal stabilizer aerodynamic center.

$^1b$ represents the fuselage, the horizontal stabilizer or the vertical stabilizer.
2.1. Dynamic Modeling

To compute the generalized force \( F_v \) generated by the vertical stabilizer, it is considered only the influence of wind speed in x-y plane \( v_{\text{air}}^{\text{xy}} \). Therefore, the force is given by

\[
F_v = (J_v)^T R^T_{I_B} \begin{bmatrix}
-\frac{1}{2} \rho (v_{\text{air}}^{\text{xy}})^2 s_v c_{\text{d}_{\text{xy}}} \left( \beta_v \right) \\
\frac{1}{2} \rho (v_{\text{air}}^{\text{xy}})^2 s_v \left[ c_{\text{d}_{\text{xy}}} \left( \beta_v \right) + c' \left( \delta_v \right) \right] \\
0
\end{bmatrix},
\]  

(2.11)

where \( c' \left( \delta_v \right) \) is a function of the rudder deflection angle \( \delta_v \), responsible to map the rudder deflection influence on the aircraft forces, and \( J_v = \frac{\partial p^T_I}{\partial q} \) is the linear velocity Jacobian, with \( p^T_I = d^T_B + R^T_{I_B} d^T_B \) being the position of vertical stabilizer aerodynamic center.

2.1.4 State-Space Representation

The state-space model associated with the dynamic equation (2.1) is written as

\[
\dot{x} = \begin{bmatrix}
\dot{q} \\
\dot{\hat{q}}
\end{bmatrix} = M(q)^{-1} \left[ \mathcal{F}(q, \dot{q}, u, \zeta) - C(q, \dot{q}) \dot{q} - G(q) \right].
\]  

(2.12)

The generalized velocities are mapped from the inertial reference frame to the body frame as

\[
\hat{q} = \begin{bmatrix}
u \\
v \\
w \\
p \\
q \\
r \\
\dot{\alpha}_R \\
\dot{\alpha}_L
\end{bmatrix} = \begin{bmatrix}
(R^T_{I_B})' & 0 & 0 \\
0 & W_\eta & 0 \\
0 & 0 & I_{2 \times 2}
\end{bmatrix} \begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{z} \\
\dot{\phi} \\
\dot{\psi} \\
\dot{\alpha}_R \\
\dot{\alpha}_L
\end{bmatrix} = \Lambda^{-1} \dot{q},
\]  

(2.13)

with \( u, v \) and \( w \) the UAV linear velocities with respect to the inertial frame represented in body frame; \( p, q \) and \( r \) are the UAV angular velocities with respect to the inertial frame represented in the body frame; \( \dot{\alpha}_R \) and \( \dot{\alpha}_L \) are the angular velocities of the tiltable mechanisms; \( R^T_{I_B} \) gives the rotation of the body frame relative to the inertial frame; and \( W_\eta \) is defined by

\[
W_\eta = \begin{bmatrix}
1 & 0 & -\sin(\theta) \\
0 & \cos(\phi) & \cos(\theta) \sin(\phi) \\
0 & -\sin(\phi) & \cos(\phi) \cos(\theta)
\end{bmatrix}.
\]  

(2.14)

By performing the mapping (2.13), the new state-space representation of the tilt-rotor
UAV nonlinear model is
\[
\dot{x} = \begin{bmatrix} \dot{q} \\ \dot{\dot{q}} \end{bmatrix} = \begin{bmatrix} \Lambda \dot{q} \\ \Lambda^{-1}[-\Lambda \dot{q} + M(q)^{-1}(\mathcal{F}(\Lambda \dot{q}, q, u, \xi) - G(q) - C(q, \Lambda \dot{q})\Lambda \dot{q})] \end{bmatrix}.
\] (2.15)

### 2.2 UAV Linear Model

The linearized model of the tilt-rotor UAV is obtained for hovering flight, considering no disturbances, from the nonlinear model presented in (2.15). Since it is intended to address a linear time-invariant and a linear time-varying MPC, two linearized models are obtained.

#### 2.2.1 UAV Linear Time-Invariant Model

In order to obtain a linearized model, it is needed to find the equilibrium point of the system. Therefore, the equilibrium is computed by
\[
\dot{x} = f(\dot{x}_{eq}, u_{eq}) = 0,
\] (2.16)
yielding to
\[
\Lambda \dot{q}_{eq} = 0
\] (2.17)
and
\[
\mathcal{F}(\Lambda \dot{q}_{eq}, q_{eq}, u_{eq}, 0) - G(q_{eq}) = B_u(q_{eq})u_{eq} - G(q_{eq}) = 0
\] (2.18)
where \((.)_{eq}\) denotes equilibrium.

Since the tilt-rotor UAV is an underactuated mechanical system, there are more variables in (2.18) than equations to be solved, thus many solutions are possible. In this work we linearize the model around the state equilibrium point given by
\[
\dot{x}_{eq} = \begin{bmatrix} x_{ref} & y_{ref} & z_{ref} & \phi_{eq} & \theta_{eq} & 0 & \alpha_{Req} & \alpha_{Leq} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}',
\]
and the equilibrium control input
\[
u_{eq} = \begin{bmatrix} f_{Req} & f_{Leq} & \tau_{Req} & \tau_{Leq} & \delta_{eq} & \delta_{eq} \end{bmatrix}',
\]
where \((.)_{ref}\) denotes trajectory reference values, and the time-derivatives of \(x_{ref}, y_{ref}, z_{ref}\) are assumed null and, consequently, do not change the dynamic behavior of the linearized model.

Therefore, the linearized model around an equilibrium point is given by
\[
\Delta \dot{x}(t) = A \Delta x(t) + B \Delta u(t),
\] (2.19)
where

\[\Delta \dot{x}(t) = \dot{x}(t) - \dot{x}_{eq}; \quad \Delta u(t) = u(t) - u_{eq};\]

\[A = \left. \frac{\partial f(\hat{x}, u)}{\partial \hat{x}} \right|_{\hat{x} = \hat{x}_{eq}, u = u_{eq}}; \quad B = \left. \frac{\partial f(\hat{x}, u)}{\partial u} \right|_{\hat{x} = \hat{x}_{eq}, u = u_{eq}}.\]

### 2.2.2 UAV Linear Time-Varying Model

In order to obtain the linear time-varying model of the tilt-rotor UAV, the equations of motion are linearized around the nominal trajectory using the same approach of the previous section.

Hence, the reference state vector is

\[\hat{x}_{ref}(t) = [x_{ref}(t) \ y_{ref}(t) \ z_{ref}(t) \ \phi_{eq} \ \theta_{eq} \ \alpha_{eq} \ \alpha_{zeq} \ \alpha_{xeq} \ \alpha_{yeq} \ \alpha_{yref}(t) \ \alpha_{zref}(t) \ \alpha_{xref}(t) \ \alpha_{yref}(t) \ \alpha_{zref}(t) \ \alpha_{xref}(t) \ v_{ref}(t) \ w_{ref}(t) \ 0 \ 0 \ 0 \ 0 \ 0].\]

Then, the reference control signals are computed replacing the state reference point in the UAV equations of motion (2.18), resulting in

\[u_{ref}(t) = B_u^+(q_{ref})[M(q_{ref})\dot{\hat{q}}_{ref} + \Lambda \dot{\hat{q}}_{ref}] + C(q_{ref}, \Lambda \dot{\hat{q}}_{ref})\dot{\hat{q}}_{ref} + G(q_{ref})]. \quad (2.20)\]

where \(B_u^+\) is the left pseudo-inverse of \(B_u\), given a feasible reference trajectory.

Thereby, the linearized model around the nominal trajectory is given by

\[\Delta \dot{x}(t) = A_v(t)\Delta \dot{x}(t) + B_v(t)\Delta u(t), \quad (2.21)\]

where

\[A_v(t) = \left. \frac{\partial f(\hat{x}, u)}{\partial \hat{x}} \right|_{\hat{x} = \hat{x}_{ref}, u = u_{ref}}; \quad B_v(t) = \left. \frac{\partial f(\hat{x}, u)}{\partial u} \right|_{\hat{x} = \hat{x}_{ref}, u = u_{ref}}.\]

### 2.3 Time-Invariant Model Predictive Control

In this section, the time-invariant model predictive control based on the linearized model of the tilt-rotor UAV from (2.19) is presented. This strategy solves the path tracking problem using a full-state feedback approach with null steady state error in the closed loop, and assuming low velocities. Furthermore, integral actions are added to improve the closed loop system and provide disturbances rejection. First, the state vector is augmented with the integral actions of \(x, y, z, \psi\) errors (Raffo et al. (2011)), and then the MPC incremental form is used (Santos & Raffo (2016b); Alfaro (2016)).
Defining
\[ e(t) = \begin{bmatrix} x(t) - x_{ref}(t) \\ y(t) - y_{ref}(t) \\ z(t) - z_{ref}(t) \\ \psi(t) - \psi_{ref}(t) \end{bmatrix}, \tag{2.22} \]
the augmented state-space equation is given by
\[
\begin{bmatrix} \Delta \dot{x}(t) \\ e(t) \end{bmatrix} = \begin{bmatrix} A & 0_{16 \times 4} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x(t) \\ \int e(t) \end{bmatrix} + \begin{bmatrix} B \\ 0_{4 \times 14} \end{bmatrix} \Delta u(t), \tag{2.23} \]
\[
\Delta \dot{x}(t) = A \Delta x(t) + B \Delta u(t). \tag{2.24} \]

By discretizing (2.24) through the Euler Method results in
\[
\Delta \bar{x}(k + 1) = A_d \Delta \bar{x}(k) + B_d \Delta u(k), \tag{2.25} \]
where \( A_d = I + At_s \) and \( B_d = Bt_s \), with \( t_s \) being the sampling time of the system.

The incremental predictive control aims to use the incremental control signal \( \delta u = \Delta u(k) - \Delta u(k - 1) \) as the control input. Thus, the state vector is augmented with \( \Delta u(k - 1) \), leading to
\[
\bar{x}(k) = \begin{bmatrix} \Delta \bar{x}(k) \\ \Delta u(k - 1) \end{bmatrix}, \tag{2.26} \]
and the resulting state-space model
\[
\bar{x}(k + 1) = \bar{A}_d \bar{x}(k) + \bar{B}_d \delta u, \tag{2.27} \]
where
\[
\bar{A}_d = \begin{bmatrix} A_d & B_d \\ 0_{6 \times 20} & I_{6 \times 6} \end{bmatrix}, \tag{2.28} \]
\[
\bar{B}_d = \begin{bmatrix} B_d \\ I_{6 \times 6} \end{bmatrix}. \tag{2.29} \]

The state-space model (2.27) is used to recursively predict the behavior of the tilt-rotor UAV. It is assumed the control horizon \( n_u \) smaller or equal to the prediction horizon \( n_y \),
2.3. TIME-IN Variant MODEL PREDICTIONAL CONTROL

and the incremental control is null in instants of time greater than the control horizon, that is, \( \delta u(k + i) = 0 \) \( \forall i > n_u \). Thus, the prediction model is

\[
\Delta \bar{x} = P \Delta \dot{x}(k) + Q \delta \bar{u}, 
\]

in which \( P \) and \( Q \) matrices are given, respectively, by

\[
P = \begin{bmatrix}
\tilde{A}_d \\
\tilde{A}_d^2 \\
\vdots \\
\tilde{A}_d^{n_y-1}
\end{bmatrix},
\]

\[
Q = \begin{bmatrix}
\tilde{B}_d & 0 & \ldots & 0 \\
\tilde{A}_d \tilde{B}_d & \tilde{B}_d & \ldots & 0 \\
\tilde{A}_d^2 \tilde{B}_d & \tilde{A}_d \tilde{B}_d & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{A}_d^{n_y-1} \tilde{B}_d & \tilde{A}_d^{n_y-2} \tilde{B}_d & \ldots & \tilde{A}_d^{n_y-n_u} \tilde{B}_d
\end{bmatrix}.
\]

The predicted state vector results in

\[
\Delta \bar{x} = \begin{bmatrix}
\dot{x}(k + 1) \\
\dot{x}(k + 2) \\
\vdots \\
\dot{x}(k + n_y)
\end{bmatrix},
\]

and the predicted control vector is

\[
\delta \bar{u} = \begin{bmatrix}
\delta u(k) \\
\delta u(k + 1) \\
\vdots \\
\delta u(k + n_u - 1)
\end{bmatrix}.
\]

The cost function used to obtain the optimal control signal is given by

\[
J = \sum_{i=1}^{n_y} \| \Delta \dot{x}(k + i) - \delta \bar{x}_r(k + i) \|^2_{w_y} + \sum_{j=d}^{n_u-1} \| \delta u(k + j) \|^2_{w_u} + L(\Delta \ddot{x}(k + n_y) - \Delta \bar{x}_r(k + n_y)).
\]

where \( \delta \bar{x}_r(k + i) = \bar{x}_r(k + i) - \ddot{x}_r(k) \) is the future reference variation, \( w_y \) is the diagonal state weighting matrix, \( w_u \) the diagonal control input weighting matrix, and \( L \) is the terminal cost.
Manipulating (2.35), it can be represented in the matrix form leading to
\[ J = [\Delta \bar{x} - \Delta \bar{x}_{r}]^T W_y [\Delta \bar{x} - \Delta \bar{x}_{r}] + \delta \bar{u}^T W_y \bar{u}, \] (2.36)
with the terminal cost matrix \( L \) being the last term of the diagonal state weight matrix \( W_y \)
\[
W_y = \begin{bmatrix}
w_y & 0 & \ldots & 0 \\
0 & w_y & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & L
\end{bmatrix}.
\] (2.37)

Aiming to use the qpOASES library, an open-source c++ implementation of the online active set strategy (Ferreau et al. (2017)), the equation (2.35) is represented in a quadratic programming form as follows:
\[ J = \frac{1}{2} \delta \bar{u}^T H \delta \bar{u} + \delta \bar{u}^T g + f_0, \] (2.38)
where
\[
H = 2(Q W_y Q + W_u),
\]
\[
g = 2(P \Delta \bar{x} - \Delta \bar{x}_{r})^T W_y Q,
\]
\[
f_0 = (P \Delta \bar{x} - \Delta \bar{x}_{r})^T W_y (P \Delta \bar{x} - \Delta \bar{x}_{r}),
\]
subject to input constraints
\[ lb \bar{A} \leq \bar{A} \delta \bar{u} \leq ub \bar{A}. \]

The input constraints used in the MPC are the physical limits of the tilt-rotor UAV actuators, which are represented by
\[ u_{\min} \leq u(k) \leq u_{\max}. \] (2.39)

The control input can be written as
\[ u(k) = u_{eq}(k) + \Delta u(k-1) + \delta u. \] (2.40)
By extending (2.40) throughout the control horizon yields
\[
\begin{bmatrix}
u(k) \\
u(k+1) \\
\vdots \\
u(k+n_u - 1)
\end{bmatrix} = \begin{bmatrix}
I \\
I \\
\vdots \\
I
\end{bmatrix} \begin{bmatrix}
u_{eq} + \Delta u(k-1) + \delta u_k \end{bmatrix} = \begin{bmatrix}
I & 0 & \ldots & 0 \\
I & I & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
I & I & \ldots & I
\end{bmatrix} \delta \bar{u}_k,
\] (2.41)
which is written in a compact form

\[ u = u_{eq} + \Delta u(k - 1) + I_m \delta u. \]  \hfill (2.42)

Thus, the control input constraints can be rewritten as

\[ u_{\min} \leq u_{eq} + \Delta u(k - 1) + I_m \delta u \leq u_{\max}. \]  \hfill (2.43)

In order to improve the performance, it is also considered state constraints as

\[ x_{\min} \leq \dot{x}(k) \leq x_{\max}. \]  \hfill (2.44)

Expanding (2.44) along the prediction horizon and using (2.30), the state constraints can be expressed as

\[ \Delta x_{\min} \leq P \Delta \dot{x}(k) + Q \delta u \leq \Delta x_{\max}. \]  \hfill (2.45)

Since the MPC strategy is based on the incremental error model, from equations (2.42) and (2.45), the constraints are mapped to incremental ones considering also the prediction and control horizons as follows

\[ \begin{bmatrix} \Delta u_{\min} - \Delta u(k - 1) \\ \Delta x_{\min} - P \Delta \dot{x}(k) \end{bmatrix} \leq \begin{bmatrix} I_m \\ Q \end{bmatrix} \delta u \leq \begin{bmatrix} \Delta u_{\max} - \Delta u(k - 1) \\ \Delta x_{\max} - P \Delta \dot{x}(k) \end{bmatrix}. \]  \hfill (2.46)

The terminal cost in equation (2.35) is computed by a Lyapunov matrix, which is a solution of the algebraic Riccati equation. It ensures the closed loop stability of the system (Raffo (2011)).

### 2.4 Time-Varying Model Predictive Control

This section presents the time-varying model predictive control. This controller is capable of update the model along the prediction horizon, which gives the MPC a better knowledge of the system behavior in the future. The procedure to obtain the LTVMPC is similar with the one presented in Section 2.4, however this MPC is based on the linearized model of the tilt-rotor UAV (2.21).

Augmenting the state vector with the integral of the error to ensure that the system
will converge to reference trajectory, a new state-space model is obtained:

\[
\begin{bmatrix}
\Delta \dot{x}(t) \\
e(t)
\end{bmatrix} = 
\begin{bmatrix}
A_v(t) & 0_{16x4} \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\Delta x(t) \\
e(t)
\end{bmatrix} + 
\begin{bmatrix}
B_v(t) \\
0_{4x6}
\end{bmatrix} \Delta u(t),
\tag{2.47}
\]

\[
\Delta \dot{x}(t) = \bar{A}_v(t) \Delta \bar{x}(t) + \bar{B}_v(t) \Delta u(t).
\tag{2.48}
\]

Discretizing (2.48) through the Euler Method, it is obtained:

\[
\Delta x(k+1) = A_{vd}(k) \Delta x(k) + B_{vd}(k) \Delta u(k),
\tag{2.49}
\]

where \( A_{vd}(k) = I + \bar{A}_v(t) t_s \) and \( B_{vd}(k) = \bar{B}_v(t) t_s \).

Using the incremental control, the system is represented in the following form

\[
\dot{x}(k+1) = \bar{A}_{vd}(k) \dot{x}(k) + \bar{B}_{vd}(k) \delta u,
\tag{2.50}
\]

where

\[
\bar{A}_{vd}(k) = 
\begin{bmatrix}
A_{vd}(k) & B_{vd}(k) \\
0_{6x20} & I_{6x6}
\end{bmatrix}
\tag{2.51}
\]

\[
\bar{B}_{vd}(k) = 
\begin{bmatrix}
B_{vd}(k) \\
I_{6x6}
\end{bmatrix}
\tag{2.52}
\]

As in the previous section, the same considerations about the predictions are valid. From (2.50), it is possible to recursively predict the behavior of the UAV, resulting in the prediction model:

\[
\Delta \bar{x} = P \Delta \bar{x}(k) + Q \delta u, 
\tag{2.53}
\]

in which the \( P \) matrix is given by

\[
P = 
\begin{bmatrix}
\prod_{i=0}^{0} \bar{A}_{vd}(k+i) \\
\prod_{i=1}^{0} \bar{A}_{vd}(k+i) \\
\prod_{i=2}^{0} \bar{A}_{vd}(k+i) \\
\vdots \\
\prod_{i=(n_y-1)}^{0} \bar{A}_{vd}(k+i)
\end{bmatrix},
\tag{2.54}
\]
and the $Q$ matrix by

$$Q = \begin{bmatrix}
\hat{B}_{vd}(k) & 0 & \ldots & 0 \\
\prod_{i=1}^{1} \hat{A}_{vd}(k+i)\hat{B}_{vd}(k) & \hat{B}_{vd}(k+1) & \ldots & 0 \\
\prod_{i=2}^{2} \hat{A}_{vd}(k+i)\hat{B}_{vd}(k) & \hat{B}_{vd}(k+1) & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\prod_{i=(n_y-1)}^{n_y} \hat{A}_{vd}(k+i)\hat{B}_{vd}(k) & \prod_{i=(n_y-1)}^{2} \hat{A}_{vd}(k+i)\hat{B}_{vd}(k+1) & \ldots & \prod_{i=(n_y-1)}^{n_u} \hat{A}_{vd}(k+i)\hat{B}_{vd}(k+n_u-1)
\end{bmatrix},$$

(2.55)

The cost function used to obtain the optimal control signal is

$$J = \sum_{i=1}^{n_y} \|\Delta \dot{x}(k+i) - \delta \dot{x}_r(k+i)\|_{w_y}^2 + \sum_{j=0}^{n_u-1} \|\delta u(k+j)\|_{w_u}^2 + L_v(\Delta \dot{x}(k+n_y) - \Delta \dot{x}_r(k+n_y)).$$

(2.56)

where $L_v$ is the terminal cost computed by the method presented in Alfaro (2016).

The state and input constraints are obtained in a similar way to the presented in Section 2.3.

The terminal cost value aims to ensure stability, from $n_y$ step, to the control design and reduce the prediction and control horizons.

According to Alfaro (2016), the terminal value $L_v$ is obtained by solving the following Linear Matrix Inequalities (LMIs) by a semidefinite programing, for the discrete system (2.50):

Minimize $\kappa$

subjected to

$$\begin{bmatrix}
\kappa & 0 \\
0 & M
\end{bmatrix} > 0,$$

(2.57)

$$\begin{bmatrix}
-M & MC_z^T+Y^TD_z^T & MA_z^T+Y^TB_z^T \\
C_zM+D_zY & -I & 0 \\
A_zM+BY & 0 & -M
\end{bmatrix} < 0,$$

(2.58)

$$M > 0$$

wherein $M = L^{-1}$, $M = M^T$ and $K = YM^{-1}$.

The $C_z$ and $D_z$ weight matrices are arbitrarily chosen satisfying

$$C_z^TD_z = 0$$

$$D_z^TD_z = W_u$$

$$C_z^TC_z = W_y.$$
2.5 MPCs Algorithm

The MPC strategies were implemented using C++ programming language allowing the study and tests of these controllers. In Section 2.5.1, the used code to implement the TIMPC designed in Section 2.3 is presented. Then, in Section 2.5.2 the algorithm that generates the TVMPC strategy, described in Section 2.4, is shown. Based on these implementations was possible to propose parallelized algorithms to improve the MPC performances, such as presented in the next chapter.

2.5.1 Time-Invariant MPC Algorithm

An object-oriented implementation of the time-invariant MPC was made by designing a class named \textit{MpcController}. Its constructor is presented in Algorithm 1, computing the equations showed in this Algorithm, on initialization.

The method \textit{MpcController::Controller()} presented in the Algorithm 2, described the computations that happen every time that a new control input is required. The control input is obtained solving the quadratic optimization problem with qpOASES library, which is an open-source C++ implementation of the online active set strategy.

2.5.2 Time-Varying MPC Algorithm

The object-oriented implementation of time-varying MPC was made by designing a class named \textit{MpcTimeVariant}. Its constructor is presented in Algorithm 3, where it was possible to observe that offline computations were reduced comparing with the time-invariant algorithm.

In the TVMPC algorithm, the method responsible to compute the control input in each sampling step is \textit{MpcTimeVariant::Controller()}, presented in the Algorithm 4.

2.6 Chapter Summary

This chapter presented the dynamic modeling of the ProVANT’s tilt-rotor UAV 3.0 (Figure 2.1), and also the development of the controllers used in this work. Section 2.1 showed the system dynamics equations of motion, obtained through the Euler-Lagrange approach. Then, in Section 2.2, the nonlinear model of the tilt-rotor UAV was linearized around an equilibrium point and the reference trajectory. Based on the linearized model, predictive controllers were designed to solve the path tracking problem using a full-state feedback approach. Thus, in Section 2.3, the obtention of the time-invariant model predictive control was illustrated, and in Section 2.4, the obtention of the time-varying MPC. Finally, in Section 2.5 the algorithms of the predictive control strategies were presented to give the reader understanding of their implementation.
Algorithm 1 MpcController::MpcController()

1: \( n_s = \text{size}(\hat{x}) \)
2: \( n_{in} = \text{size}(u) \)
3: \( Q.setZero() \)
4: for \((j = 0; j < n_u \times n_{in}; j + = n_{in})\) do
5: \( \text{auxA} = \text{Identity}(n_s, n_s) \)
6: for \((i = 0; i < n_y \times n_s; i + = n_s)\) do
7: \( \text{if } i \geq j \text{ then} \)
8: \( Q(i, j) = \text{auxA} \times B_d \)
9: \( \text{auxA} = A_d \times \text{auxA} \)
10: \( \text{if } j == 0 \text{ then} \)
11: \( P(i, 0) = \text{auxA} \)
12: \( \text{end if} \)
13: \( \text{end if} \)
14: \( \text{end for} \)
15: \( \text{end for} \)
16: for \((i = 0; i < n_u \times n_{in}; i + = n_{in})\) do
17: \( u_{eq}(i, 0) = u_{eq} \)
18: \( \text{end for} \)
19: for \((i = 0; i < n_y \times n_s; i + = n_s)\) do
20: \( \Delta x_{\text{min}}(i, 0) = \dot{x}_{\text{min}} - \dot{x}_{eq} \)
21: \( \Delta x_{\text{max}}(i, 0) = \dot{x}_{\text{max}} - \dot{x}_{eq} \)
22: \( \text{end for} \)
23: for \((i = 0; i < n_y \times n_s; i + = n_s)\) do
24: \( \text{if } i < n_s \times (n_s - 1) \text{ then} \)
25: \( W_y(i, i) = w_y \)
26: \( \text{else} \)
27: \( W_y(i, i) = L \)
28: \( \text{end if} \)
29: \( \text{end if} \)
30: \( W_u(i, i) = w_u \)
31: \( \text{end for} \)
32: for \((j = 1; j \geq n_u \times n_{in}; j + +)\) do
33: for \((i = 1; i \geq n_u \times n_s; i + +)\) do
34: \( \text{if } (i - j) \geq 0 \text{ then} \)
35: \( \text{aux}(n_{in}(i - 1), 0) = \text{Identity}(n_{in}, n_{in}) \)
36: \( \text{else} \)
37: \( \text{aux}(n_{in}(i - 1), 0) = \text{Zero}(n_{in}, n_{in}) \)
38: \( \text{end if} \)
39: \( \text{if } i == n_{in} \text{ then} \)
40: \( I_m(0, n_{in}(j - 1)) = \text{aux} \)
41: \( I_{aux}(n_{in}(j - 1), 0) = \text{Identity}(n_{in}, n_{in}) \)
42: \( \text{end if} \)
43: \( \text{end if} \)
44: \( \text{end for} \)
45: \( \text{end for} \)
46: \( A_r << I_m, Q \)
47: \( H = 2(Qw_yQ + W_u) \)
Algorithm 2 MpcController::Controller($\Delta \bar{x}$)

1: for ($i = 1; i \leq n_y \times n_x; i + +$) do
2:  \[ \Delta x = \text{trajectory} \to \text{TrajectoryReference}(k+i) - \text{trajectory} \to \text{TrajectoryReference}(k) \]
3: end for
4: \[ g = 2(P \Delta x - \Delta \bar{x})'W_y Q \]
5: \[ \text{ubAr}(0, 0) = I_{aux} (u_{max} - \Delta u(k-1)) - u_{eq} \]
6: \[ \text{ubAr}(n_u, 0) = \Delta \bar{x}_{max} - P \Delta \bar{x} \]
7: \[ \text{lbAr}(0, 0) = I_{aux} (u_{min} - \Delta u(k-1)) - u_{eq} \]
8: \[ \text{lbAr}(n_u, 0) = \Delta \bar{x}_{min} - P \Delta \bar{x} \]
9: //Coping variables in row-major order
10: for ($i = 0; i < (n_y n_x + n_x n_u); i + +$) do
11:  \[ \text{ubArq}[i] = \text{ubAr}(i) \]
12:  \[ \text{lbArq}[i] = \text{lbAr}(i) \]
13:  for ($j = 0; j < n_u; i + +$) do
14:   if $i < (n_y n_u)$\&\&$j < (n_x n_u)$ then
15:    \[ H[i(n_u n_x) + j] = H[i + (n_y n_u)] \]
16:  end if
17:  if $i == 0$ then
18:    \[ gq[j] = g(j) \]
19:  end if
20:  \[ Arq[i(n_y n_x) + j] = Ar(i + (n_x n_y + n_y n_u)) \]
21: end for
22: end for
23: Create an QProblem object qp (from qpOASES)
24: \[ qp.init(Hq, gq, Arq, 0, 0, lbArq, ubArq, 200) \]
25: \[ qp.getPrimalSolution(\delta u) \] //Obtain the optimal solution
Algorithm 3 MpcTimeVariant::MpcTimeVariant()

1: \( n_s = \text{size}(\dot{x}) \)
2: \( n_{in} = \text{size}(u) \)
3: \( \text{for} \ (i = 0; i < n_y \times n_z; i+ = n_s) \ \text{do} \)
4: \( \Delta \dot{x}_{\text{min}}(i, 0) = \dot{x}_{\text{min}} - \dot{x}_{eq} \)
5: \( \Delta \dot{x}_{\text{max}}(i, 0) = \dot{x}_{\text{max}} - \dot{x}_{eq} \)
6: \( \text{end for} \)
7: \( \text{for} \ (i = 0; i < n_y \times n_z; i+ = n_s) \ \text{do} \)
8: \( \text{if} \ i < n_x \times (n_s - 1) \ \text{then} \)
9: \( W_y(i, i) = w_y \)
10: \( \text{else} \)
11: \( W_y(i, i) = L \)
12: \( \text{end if} \)
13: \( \text{end for} \)
14: \( \text{for} \ (i = 0; i < n_u \times n_{in}; i+ = n_{in}) \ \text{do} \)
15: \( W_u(i, i) = w_u \)
16: \( \text{end for} \)
17: \( \text{for} \ (j = 1; j \geq n_u \times n_{in}; j++) \ \text{do} \)
18: \( \text{for} \ (i = 1; i \geq n_u \times n_{in}; i++) \ \text{do} \)
19: \( \text{if} \ (i - j) \geq 0 \ \text{then} \)
20: \( \text{aux}(n_{in}(i - 1), 0) = \text{Identity}(n_{in}, n_{in}) \)
21: \( \text{else} \)
22: \( \text{aux}(n_{in}(i - 1), 0) = \text{Zero}(n_{in}, n_{in}) \)
23: \( \text{end if} \)
24: \( \text{if} \ i == n_{in} \ \text{then} \)
25: \( I_{in}(0, n_{in}(j - 1)) = \text{aux} \)
26: \( I_{aux}(n_{in}(j - 1), 0) = \text{Identity}(n_{in}, n_{in}) \)
27: \( \text{end if} \)
28: \( \text{end for} \)
29: \( \text{end for} \)
Algorithm 4 MpcTimeVariant::Controller($\Delta \bar{x}$)

1: for ($i = 1; i \leq n_y \times n_u; i + +$) do
2: $\Delta x_i =$ trajectory$\rightarrow$TrajectoryReference($k$) $-$ trajectory$\rightarrow$TrajectoryReference($k$)
3: end for
4: for ($i = 1; i \leq n_u \times n_{in}; i + +$) do
5: $\hat{x}_{ref} =$ trajectory$\rightarrow$Acceleration($k$)
6: $u_{ref}(i, 0) = u_{ref}(\hat{x}_{ref})$
7: end for
8: $Q.setZero()$
9: for ($j = 0; j < n_u \times n_{in}; j + +$) do
10: $auxA = Identity(n_u, n_u)$
11: for ($i = 0; i < n_y \times n_{in}; i + +$) do
12: if $i \geq j$ then
13: $\hat{x}_{ref} =$ trajectory$\rightarrow$Acceleration($k$)
14: $A_{vd} =$ Model$\rightarrow$LinearModelA($\hat{x}_{ref}$)
15: $Q = auxA \times B_{vd}$
16: $auxA = A_{vd} \times auxA$
17: if $j == 0$ then
18: $P(i, 0) = auxA$
19: end if
20: end if
21: end for
22: end for
23: $Ar << I_{n_{in}} Q$
24: $H = 2(Q^T W_y Q + W_u)$
25: $g = 2(P \Delta x_{ref} - \Delta x_{ref})^T W_y Q$
26: $ubAr(0, 0) = I_{aux}(u_{max} - \Delta u(k - 1)) - u_{ref}$
27: $ubAr(n_{in} n_u, 0) = \Delta x_{max} - P \Delta \bar{x}$
28: $lbAr(0, 0) = I_{aux}(u_{min} - \Delta u(k - 1)) - u_{ref}$
29: $lbAr(n_{in} n_u, 0) = \Delta x_{min} - P \Delta \bar{x}$
30: //Coping variables in row-major order
31: for ($i = 0; i < (n_{in} n_y + n_{in} n_u); i + +$) do
32: $ubArq[i] = ubAr(i)$
33: $lbArq[i] = lbAr(i)$
34: for ($j = 0; j < n_{in} n_u; i + +$) do
35: if $i < (n_{in} n_u) \&\& j < (n_{in} n_u)$ then
36: $H[i(n_{in} n_u) + j] = H[i + (n_{in} n_u) j]$
37: end if
38: if $j == 0$ then
39: $gq[j] = g(j)$
40: end if
41: $Arq[i(n_{in} n_u) + j] = Ar(i + (n_{in} n_y + n_{in} n_u) j)$
42: end for
43: end for
44: Create an QProblem object qp
45: qp.init($H_q, g_q, Arq, 0, 0, lbArq, ubArq, 200$)
46: qp.getPrimalSolution($\delta u$)
This chapter aims to present the multi-core software approach developed to improve the time performance of the predictive control strategies presented in the previous chapter. Initially, the simulation environment created to represent and control the tilt-rotor UAV 3.0 is presented in Section 3.1. Then, in Section 3.2, the multi-core software approach for the time-invariant MPC is developed. In Section 3.3, the architecture of the procedure processor-in-the-loop (PIL), built to develop and test the tilt-rotor UAV controller in the final target processor, is described. Finally, the designed multi-core software architecture for the time-varying MPC used in the PIL application is presented in Section 3.4.

3.1 ProVANT Simulation Environment

The ProVANT simulation environment is a simulator for testing control strategies of tilt-rotor UAVs developed in ProVANT project that was initially presented in Lara (2016); Lara et al. (2017). This simulator uses the Robot Operating System (ROS) framework (Quigley et al. (2009)) and the Gazebo simulator (Koenig & Howard (2004)), which is a platform for development of 3D simulations, designed to represent the physical world. It is composed by the distribution ROS Kinetic and Gazebo 7, emulating the dynamic behavior of articulated rigid bodies providing graphic output and collision detection.

Initially, the mechanical design of tilt-rotor UAV 2.0 was incorporated in the simulator. And later, in Machado (2017); Miranda et al. (2017) was added the tilt-rotor UAV 3.0. The Computer Aided Design (CAD) model was designed in Solidworks and through the
tool SolidWorks to URDF Exporter, the meshes and Unified Robot Description Format (URDF) file were generated. Then, it is transformed into a Simulation Description Format (SDF) extension for a more complete description. In Figure 3.1, the tilt-rotor UAV 3.0 is illustrated on the simulation environment.

![Figure 3.1: Provant Simulator.](image)

Through the programming language C++, the implementation of dynamic libraries, plugins, in Gazebo, that give control and access to its several aspects and functionalities, sensors and actuators were incorporated in the aircraft enabling the motion. There are some types of plugins in Gazebo, the model plugins are the ones attached to the tilt-rotor UAV model, controlling its joints and states. Four model plugins were created:

- Alldata plugin: it has the purpose of retrieving information about the UAV’s pose, and setup a message with the states of the UAV.

- Servomotor plugin: this plugin is an interface in charge of applying a given torque on the servomotors and a given deflection on the elevator and rudder.

- Brushless plugin: it is an actuator, that produces a given thrust force on the thrusters’ group.

- Aerodynamic plugin: it is in charge of computing and applying aerodynamic forces generated by the relative wind on the UAV surfaces.

Moreover, the simulator possesses sensor plugins capable of settling sensor properties and acquiring information from a specific sensor attached to a link. Further, a world
3.1. PROVANT SIMULATION ENVIRONMENT

The plugin is in charge of synchronizing the ROS node with the Gazebo simulation, controlling the world properties. This ROS node is an executable file responsible for sending control signals to their specific topics (that are communication mechanisms of ROS), and receiving data from the topic States with values of the currently states. All the presented plugins exchange information by specific topics defined in the SDF file model.

The simulation environment in Gazebo updates at each simulation step of 1ms. Then, the sensor information is updated into the State topic, the control inputs are computed by the controller, and the control signals are applied on the UAV actuators.

The MPC was implemented as a class in C++ language and included in the ROS node, which is responsible to execute the controller method of the MPC class. The controller method receives the UAV states and returns the control signal: forces applied by the thrusters, torques applied by the servomotors, and deflections of elevator and rudder. The ROS node obtains the state information, subscribing it to the States topic, then, publishes the control inputs received from the controller on the topics: Right Thrust, Left Thrust, Right Torque, Left Torque, Elevator Deflection and Rudder Deflection. Figure 3.2, illustrates this information.

![Diagram of data flow in ProVANT simulator](image-url)

Figure 3.2: Data Flow of ProVANT simulator.
Although the simulation step of the simulator is 1ms, the simulation step of the controller is 12ms. Thus, the published control signals are held until new values are available.

### 3.2 Multi-Core Software Approach to Time-Invariant MPC

The execution time of an application may be improved by using a multithreaded code instead of using a sequential code since computational expenses can be shared between various processors. With this goal, a multi-core software architecture of the MPC for the tilt-rotor UAV application is proposed.

The implemented class MPC has a complex algorithm with a large execution time. Thus, the multi-core software approach pursues the parallelization technique to speed up the controller. With this end, some modifications are made on the simulation environment to use the four cores of the computer. The host computer running this program is an Intel Core i5 with Ubuntu 16.04 operating system.

The ROS node illustrated in Section 3.1, is divided in a communication and control thread, to facilitate the understanding and implementation. The communication thread is in charge of publishing the control inputs on its respective topics and receiving the states by subscribing to the States topic, as shown in Figure 3.3.

This thread starts with initializations, subscribes to the topics of interest, and then waits the data be published on the State topic by the simulator. The received data are the current states of the UAV, which are updated on the shared memory, that has the access controlled by condition variable. After that, the control signals are read, and published on the topics.

The second thread is responsible for computing the control signals. Its behavior is presented in the Flowchart of Figure 3.4. It starts by initializing the control variables and then taking the states values from the shared memory. With these new values, control signals are calculated, checked and, if OK, the shared control vector is updated, if not, the previous control inputs are considered to the update, then the thread waits until new states are received.

Each of these threads run simultaneously in a different core, then a parallelization technique is proposed on the control thread. Thus, the communication thread runs in the core number one, and the control thread in core number two. The latter one is called master thread, it creates a parallel region, where the master thread is split into four threads. These threads dispute the available time of the CPU to run, and in the end of the parallel region, these four threads became the master thread again. The created threads, in parallel region, operate on the four cores available, obtaining a parallelism relation...
3.2. MULTI-CORE SOFTWARE APPROACH TO TIME-ININVARIANT MPC

Figure 3.3: Flowchart of the communication thread.

between them, and a concurrency relation on the core onste, where the communication thread sleeps. This process is implemented using Open Multi-Processing (OpenMP - Barbara Chapman, Gabriele Jost (2008)), an application programming interface that supports multi-platform shared memory multiprocessing programming. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.

As show in Chapter 2, the MPCTI has the method Controller(), that computes the variables that change at each iteration step and solves the optimization problem, and the PredictionModel() that computes the time invariant structures. As the method Controller() runs every step, it has more affects on the performance of the controller, its profile was generated by the programming tool Valgrind, as shown in Figure 3.10, leading the choices made to parallelize the MPC code. Observing the column incl., that means, inclusive costs, it was noted that Eigen methods to work in matrices operations had higher computational costs, so, the parallelization technique was applied in the matrices operations of the controller.

When we create a parallel region using OpenMP, the master thread divides itself in other threads, as illustrated in Figure 3.6. At the end of the parallel region, at the barrier,
the threads wait and terminate at the same time. Aiming to facilitate this procedure, directives were created to inform the compiler how the following code will be interpreted. This directives are the pragmas. Thus, the code is analyzed and rewritten in an independent form taking advantage of the instructions available in OpenMP.

We start the controller formulation by adding the openMP pragma to compute $P$ and $Q$ matrices (equations (2.31) and (2.32)) like show the Algorithm 5.

The directive `#pragma omp parallel` creates the parallel region. By default, this pragma assumes that variables created outside the parallel region are shared variables, i.e., the variables are visible to all threads, and changes made in this data are seen by all of them.
3.2. MULTI-CORE SOFTWARE APPROACH TO TIME-INVARIANT MPC

![Parallel Regions Diagram](image)

Figure 3.6: Parallel Regions.

**Algorithm 5** Calculation of $P$ and $Q$ matrices

1: `#pragma omp parallel`
2: `id = get the thread number`
3: if `id = 0` then
4: Compute $Q$
5: else
6: Compute $P$
7: end if

Subsequently, we compute the vectors of constraints, the weight matrices and some auxiliaries matrices using the pragmas as shown in Algorithm 6.

**Algorithm 6** Calculation of matrices to prediction

1: `#pragma omp for firstprivate`
2: Compute states weight matrix
3: `#pragma omp for firstprivate`
4: Compute control weight matrix
5: `#pragma omp for collapse(2)`
6: auxiliaries matrices
7: Finish the parallel region

The directive `#pragma omp for firstprivate` allows the variables to be initialized with the value of a given variable at the parallelization moment. Then, the variable is private in the parallel region, that is, the data can be accessed only by the thread that owns it. And, the directive `#pragma omp for collapse(2)` indicates that multiple (2) loops can be parallelized in the nest, forming a single loop and then parallelizing it.

To compute the $H$ matrix, we use the OpenMP to activate the internal parallelization of Eigen library.

Then, we compute the future reference vector with the pragma show in the Algorithm 7, and finish the calculus of the control signal using the internal Eigen parallelization.
**Algorithm 7** Calculation of future reference vector

1: #pragma omp parallel for private
2: Compute the reference vector

### 3.3 Processor-In-The-Loop

This section presents the Processor-In-The-Loop technique used in this work to develop and test in an embedded system the software architecture of the MPC strategy that will be presented in Section 3.4. The incorporation of PIL aims to fulfill test simulations with the tilt-rotor UAV model, verifying the controllers effectiveness.

The PIL simulation was assembled with a DE0-Nano-SoC Kit/Atlas-SoC Kit, an ARM-based hard processor system (HPS) consisting of a dual-core Cortex-A9 processor, peripherals and memory interfaces. The operating system installed on the SoC was the Debian Jessie 8.5 Console, and the codes are compiled with NEON technology and floating point architecture. The SoC was connected to a computer Intel Core i5 with the Ubuntu 16.04 operating system. It is represented by the deployment diagram in Figure 3.7.

![Deployment Diagram](image)

**Figure 3.7:** Deployment Diagram.

The communication between the devices was established by an Ethernet crossover cable, allowing the remote access on the board through the host PC console, and by a serial connection for data transmission between the Communication thread running on the embedded processor and the ROS control system on the PC. Into the PC block is shown a third component, the Provant simulator presented in section 3.1. In the present work some modifications were made in the ProVANT simulator to enable serial communication, and hence allow the PIL implementation.

The De0-nano-SoC block of the embedded system illustrates two components that represents the Communication and Controller threads, which are similar to the ones
presented in Section 3.2, but modified to run in the SoC, instead of publishing and subscribing on topics, the information of sensors and actuators is read and write on the serial port by the Communication thread.

The Communication thread is responsible for sending the control inputs to the control system on the PC and receiving the states from it, as shown the flowchart in Figure 3.8. This thread starts initializing variables and then reads the states from the serial port, the exactly number of bytes corresponding to the states. The current states of the UAV are updated so, the received data is shared with the control thread. The thread then waits for the control signals. If the control signals are not calculated in a interval of 12 ms, the previous values are sent through the port. After that, the thread waits for new states to arrive on the serial port.

![Flowchart of the communication thread on SoC.](image)

The second thread is specifically responsible to compute the control signal. Its behavior
is same presented in the flowchart of Figure 3.4.

On the PC, the Control System is composed by one thread created to handle the serial data and the communication with the Provant simulator. This thread reads the state information from a ROS topic and sends it to the controller through the serial connection using the Boost.Asio cross-platform C++ library. Afterward, it reads the control input and publishes it on the six topics intended to the control inputs, like exposed in Section 3.1. The Figure 3.9 shows the final architecture of PIL simulation.

![Figure 3.9: Processor-In-The-Loop Architecture.](image)

While the computer runs the Provant simulator, that contains the tilt-rotor model description developed in Gazebo, the De0-nano-SoC runs the MPC.

## 3.4 Multi-Core Software Architecture to Time-Varying MPC

This section proposes a multi-core software architecture for the time-varying MPC, embedded on the dual-core device previously presented, to built the PIL simulation. This architecture is a refined version from the one developed for the simulation on ProVANT, in section 3.2, modified to LTVMPC.

The profile for the time-variant MPC was obtained, also showing that matrices operations had higher computational costs, so, the parallelization technique was applied in the matrices operations of the controller.

The controller formulation starts by adding the openMP pragma to offline computation of the constraints vectors, the weight matrices and auxiliaries matrices in (2.42). The pragmas are used as follow:

Subsequently, starting the calculation of the online structures, we compute the future reference vector with the pragma show in the Algorithm 9.

Then, we compute $P$ (2.54) and $Q$ (2.55) matrices as in Algorithm 10.

To compute the $H$ matrix in (2.38), we use the OpenMP to activate the internal parallelization of Eigen library. The matrix parallelization by Eigen remains activated until the optimization problem ends.
3.5 Chapter Summary

This chapter presented a multi-core software architecture for both MPC developed in this work. In Section 3.1, the Provant Simulator was described, illustrating the mechanisms that allow the tilt-rotor UAV 3.0 to be controlled by the parallel TIMPC on the simulation. In Section 3.2, the time-invariant predictive controller implementation, applying a parallelization technique, was presented. In Section 3.3, was developed a PIL application that allows the controller to be embedded on the target processor. Its features were presented and then, in Section 3.4, the multi-core software architecture of the TVMPC, developed
to the target device, is described.
In this chapter, some simulation results for path tracking of the tilt-rotor UAV 3.0 obtained with the control strategies developed in Chapter 2 are presented. The time-invariant MPC designed for multi-core processors is simulated in the ProVANT simulator, as shown in Section 3.2. And, the time-varying MPC designed for multi-core processors, presented in Section 3.4, is evaluated in a processor-in-the-loop simulation. Then, the data from both simulations are collected and plot using MATLAB.

Initially, in Section 4.1, the physical parameters that characterize the model of the Tilt-rotor UAV 3.0, taken from its CAD design, are presented, as well as the tuning parameters of the controller. In Section 4.2, the results from the simulation in the ProVANT simulator are presented, evaluating the performance of the parallelized time-invariant MPC. Also, the execution time of this MPC is measured and exposed. In Section 4.3, the results from the procedure PIL is presented, illustrating the effectiveness of the parallelized time-varying MPC and evaluating its execution time performance.

## 4.1 System Parameters

The UAV physical parameters used to obtain the dynamic model and configured the simulator are:
Table 4.1: UAV physical parameters

<table>
<thead>
<tr>
<th>Mass of the i-th body</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>1.5538</td>
<td>Kg</td>
</tr>
<tr>
<td>$m_2$</td>
<td>0.0844</td>
<td>Kg</td>
</tr>
<tr>
<td>$m_3$</td>
<td>0.0844</td>
<td>Kg</td>
</tr>
</tbody>
</table>

Distances between the i-th frames

<table>
<thead>
<tr>
<th>Distance</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_B^C_{1}$</td>
<td>$\begin{bmatrix} 0 &amp; 0 &amp; 0 \end{bmatrix} \times 10^{-3}$</td>
<td>m</td>
</tr>
<tr>
<td>$d_{C_2}^{B\text{aux}}$</td>
<td>$\begin{bmatrix} 0.000678 &amp; -0.270482 &amp; 0.065905 \end{bmatrix}'$</td>
<td>m</td>
</tr>
<tr>
<td>$d_{C_2}^{B\text{aux}}$</td>
<td>$\begin{bmatrix} 0.00595 &amp; 0.270515 &amp; 0.065905 \end{bmatrix}'$</td>
<td>m</td>
</tr>
<tr>
<td>$d_{C_3}^{B\text{aux}}$</td>
<td>$\begin{bmatrix} 0 &amp; 0 &amp; 0.054 \end{bmatrix}'$</td>
<td>m</td>
</tr>
<tr>
<td>$d_{C_3}^{B\text{aux}}$</td>
<td>$\begin{bmatrix} 0 &amp; 0 &amp; 0.054 \end{bmatrix}'$</td>
<td>m</td>
</tr>
</tbody>
</table>

Propellers tilt angle

<table>
<thead>
<tr>
<th>Tilt angle</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$</td>
<td>4</td>
<td>degree</td>
</tr>
</tbody>
</table>

Thrusters coefficient

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_r$</td>
<td>$1.7 \times 10^{-7}$</td>
</tr>
<tr>
<td>$b$</td>
<td>$9.5 \times 10^{-6}$</td>
</tr>
</tbody>
</table>

BH density of the air

<table>
<thead>
<tr>
<th>Density</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>1.21</td>
<td>Kg/m$^3$</td>
</tr>
</tbody>
</table>

i-th aerodynamic surface area

<table>
<thead>
<tr>
<th>Surface</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_f$</td>
<td>0.20723585</td>
<td>m$^2$</td>
</tr>
<tr>
<td>$s_h$</td>
<td>0.0528758</td>
<td>m$^2$</td>
</tr>
<tr>
<td>$s_v$</td>
<td>0.026482</td>
<td>m$^2$</td>
</tr>
</tbody>
</table>

Gravitational acceleration

<table>
<thead>
<tr>
<th>Acceleration</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g$</td>
<td>$\begin{bmatrix} 0 &amp; 0 &amp; -9.80665 \end{bmatrix}$</td>
<td>m/s$^2$</td>
</tr>
</tbody>
</table>

Distance between the aerodynamic center $d_i$ and the body frame $B$

<table>
<thead>
<tr>
<th>Distance</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_f^B$</td>
<td>$\begin{bmatrix} -0.0050 &amp; 0 &amp; 0.0326 \end{bmatrix}'$</td>
<td>m</td>
</tr>
<tr>
<td>$d_h^B$</td>
<td>$\begin{bmatrix} -0.3858 &amp; 0 &amp; 0.1194 \end{bmatrix}'$</td>
<td>m</td>
</tr>
<tr>
<td>$d_v^B$</td>
<td>$\begin{bmatrix} -0.3277 &amp; 0 &amp; 0.0288 \end{bmatrix}'$</td>
<td>m</td>
</tr>
</tbody>
</table>

Inertia moment of the i-th body

<table>
<thead>
<tr>
<th>Moment</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{C_1}$</td>
<td>$\begin{bmatrix} 4788.34 &amp; -1.54722 &amp; -363.249 \ 2275.21 &amp; 0.822828 &amp; \times 10^{-5} \ 6062.31 \end{bmatrix}$</td>
<td>Kg.m$^2$</td>
</tr>
</tbody>
</table>
4.1. SYSTEM PARAMETERS

\[
I_{C_2} = \begin{bmatrix}
4.04442 & -0.00767 & -0.0101488 \\
* & 3.83961 & 0.119379 \\
* & * & 1.58774
\end{bmatrix} \times 10^{-5} \text{ Kg.m}^2
\]

\[
I_{C_3} = \begin{bmatrix}
4.04347 & -0.008146 & -0.0101554 \\
* & 3.83961 & -0.139683 \\
* & * & 1.58679
\end{bmatrix} \times 10^{-5} \text{ Kg.m}^2
\]

Aerodynamic coefficients

\[c_{fxx}^d(\alpha_f) = 0.4566\alpha_f^2 - 0.0403\alpha_f + 0.0601\]

\[c_{fxx}^l(\alpha_f) = 0.5405\alpha_f - 0.0353\]

\[c_{fxy}^d(\beta_f) = 0.3513\beta_f^2 - 10^{-15}\beta_f + 0.0604\]

\[c_{fxy}^l(\beta_f) = 0.3821\beta_f - 0.0003\]

\[c_{axy}^d(\beta_v) = 2.2019\beta_v^2 - 2 \times 10^{-7}\beta_v + 0.0149\]

\[c_{axy}^l(\beta_v) = -45.392\beta_v^3 + 0.0011\beta_v^2 + 6.1126\beta_v\]

\[c^e(\delta_e) = 2.1873375\delta_e\]

\[c_{axx}^d(\alpha_h) = 1.9382\alpha_h^2 + 0.0088\]

\[c_{axx}^l(\alpha_f) = -35.216\alpha_h^3 + 9 \times 10^{-15}\alpha_h^2 + 6.5306\alpha_h - 6 \times 10^{-14}\]

\[c^e(\delta_e) = 2.1873375\delta_e\]

For more details about the aerodynamic of the tilt-rotor UAV see Queiroz (2014).

The equilibrium point of the tilt-rotor UAV used to generate the linearized model in (2.19) is presented in Table 4.2.

Table 4.2: Equilibrium Point of the Tilt Rotor UAV for the TIMPC

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\phi_{eq})</td>
<td>3.6864 \times 10^{-6}</td>
<td>rad</td>
</tr>
<tr>
<td>(\theta_{eq})</td>
<td>0.0113</td>
<td>rad</td>
</tr>
<tr>
<td>(\alpha_{R_{eq}})</td>
<td>-0.0113</td>
<td>rad</td>
</tr>
<tr>
<td>(\alpha_{L_{eq}})</td>
<td>-0.0113</td>
<td>rad</td>
</tr>
<tr>
<td>(F_{R_{eq}})</td>
<td>8.4674</td>
<td>N</td>
</tr>
<tr>
<td>(F_{L_{eq}})</td>
<td>8.4665</td>
<td>N</td>
</tr>
<tr>
<td>(\tau_{R_{eq}})</td>
<td>-1.8433 \times 10^{-6}</td>
<td>N.m</td>
</tr>
<tr>
<td>(\tau_{L_{eq}})</td>
<td>-3.1111 \times 10^{-6}</td>
<td>N.m</td>
</tr>
<tr>
<td>(\delta_{eq})</td>
<td>0</td>
<td>rad</td>
</tr>
<tr>
<td>(\delta_{req})</td>
<td>0</td>
<td>rad</td>
</tr>
</tbody>
</table>

The reference values used to generate the linearized model (2.21) is described in Table 4.3.

Constraints of the dynamic model are defined by the real limits of the tilt-rotor UAV. State constraints are presented in Table 4.4 (the remaining states are not limited):
Table 4.3: Reference Values for the Linearized Model of the TVMPC

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_{ref}$</td>
<td>$3.6864 \times 10^{-6}$</td>
<td>rad</td>
</tr>
<tr>
<td>$\theta_{ref}$</td>
<td>0.0113</td>
<td>rad</td>
</tr>
<tr>
<td>$\alpha_{R_{ref}}$</td>
<td>-0.0113</td>
<td>rad</td>
</tr>
<tr>
<td>$\alpha_{L_{ref}}$</td>
<td>-0.0113</td>
<td>rad</td>
</tr>
<tr>
<td>$F_{R_{ref}}$</td>
<td>$0.7797 v_d - 0.0099 u_d + 0.8633 w_d + 8.4674$</td>
<td>N</td>
</tr>
<tr>
<td>$F_{L_{ref}}$</td>
<td>$0.8633 w_d - 0.7797 v_d - 0.0096 u_d + 8.4665$</td>
<td>N</td>
</tr>
<tr>
<td>$\tau_{R_{ref}}$</td>
<td>$4.5573 \times 10^{-3} u_d + 3.617 \times 10^{-6} v_d + 5.1728 \times 10^{-5} w_d - 1.8433 \times 10^{-6}$</td>
<td>N.m</td>
</tr>
<tr>
<td>$\tau_{L_{ref}}$</td>
<td>$4.5573 \times 10^{-3} u_d - 3.6081 \times 10^{-6} v_d + 5.1599 \times 10^{-5} w_d - 3.1111 \times 10^{-6}$</td>
<td>N.m</td>
</tr>
<tr>
<td>$\delta_{e_{ref}}$</td>
<td>0</td>
<td>rad</td>
</tr>
<tr>
<td>$\delta_{r_{ref}}$</td>
<td>0</td>
<td>rad</td>
</tr>
</tbody>
</table>

Table 4.4: State Constraints of the Tilt Rotor UAV

<table>
<thead>
<tr>
<th>state</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>$y$</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>$z$</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$\phi$</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$\theta$</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$\psi$</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha_R$</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha_L$</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

Control input constraints are described in Table 4.5.

Table 4.5: Input Constraints Of The Tilt Rotor UAV

<table>
<thead>
<tr>
<th>control input</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_R$</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>$f_L$</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>$\tau_R$</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_L$</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>$\delta_e$</td>
<td>-0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>$\delta_r$</td>
<td>-0.7</td>
<td>0.7</td>
</tr>
</tbody>
</table>

To obtain the discrete linearized prediction model for the TIMPC, it was used a sampling time of 12ms, a prediction horizon of 40 and a control horizon of 5ms, and for the TVMPC a sampling time of 12ms, a prediction horizon of 5 and a control horizon of 3.

The weighing matrices are computed by the Bryson’s methods \cite{Johnson1987}, resulting for both MPCs in

$$ w_y = diag(9, 9, 9, 16, 16, 9, 16, 16, 0.25, 0.25, 0.25, 0.25, 0.11, 0.11, 0.25, 0.25, 0.06, 0.06, 25, 25, 25, 25, 0.01, 0.01, 0.25, 0.25, 1, 1) $$
$w_u = \text{diag}(2.3434 \times 10^{-5}, 2.3427 \times 10^{-5}, 2.5 \times 10^{-5}, 1.6211 \times 10^{-5})$}

### 4.2 Results from ProVANT Simulator

The desired trajectory for the tilt-rotor UAV that evaluates the TIMPC is presented in Table 4.6

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_{ref}$</th>
<th>$y_{ref}$</th>
<th>$z_{ref}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq t &lt; 2$</td>
<td>0 m</td>
<td>0 m</td>
<td>$1+0.5t$ m</td>
</tr>
<tr>
<td>$2 \leq t &lt; 32$</td>
<td>$4 - 4\cos\left(\frac{\pi}{15}(t - 2)\right)$ m</td>
<td>$4\sin\left(\frac{\pi}{15}(t - 2)\right)$ m</td>
<td>$6 - 4\cos\left(\frac{\pi}{15}(t - 2)\right)$ m</td>
</tr>
<tr>
<td>$32 \leq t &lt; \infty$</td>
<td>0 m</td>
<td>0 m</td>
<td>2 m</td>
</tr>
</tbody>
</table>

Initially, the trajectory starts simulating a vertical take-off, then a circular path is performed returning to its start point. The performed path by the tilt-rotor UAV on the scenario shown in Section 3.2 is presented in Figure 4.1. It is noted that the system follows the reference, stabilizing on the final position in two meters in height.

In Figure 4.2 to 4.7, the state variables are shown. Observe that, due to the integral action, the tracking of $x$, $y$, $z$, and $\psi$, in Figure 4.2, achieve null steady-state error at the end of the trajectory, when the reference stop varying. In Figure 4.3, only the pitch angle, given by state $\theta$, stabilized in a different value from the reference. This fact occurred due the presence of wind disturbances (presented in Figure 4.9) affecting the aircraft. The same happens in Figure 4.4, with angles $\alpha_R$ and $\alpha_L$, since the wind leads the states to achieve a new equilibrium point. In Figure 4.8 are presented the control signals applied on the tilt-rotor UAV.
The simulations with the time-invariant MPC were made in an Intel Core i5 computer with the Ubuntu 16.04 operating system, using double-precision floating-point format. The simulation time was measured using the original TIMPC architecture and the proposed one consisting of multiple processors. The obtained execution time of the controller are shown in Figures 4.10 and 4.11. By comparing the average of the results, it is noted an improvement from 33.94ms on the serial MPC to 26.55ms spent by the parallel MPC.
4.2. RESULTS FROM PROVANT SIMULATOR

Figure 4.3: Time evolution of $\phi$, $\theta$ and $\psi$, using the TIMPC.

Figure 4.4: Time evolution of $\alpha_R$ and $\alpha_L$, using the TIMPC.

The variation between execution times is due to a set of variables, such for instance the processing power of the computer at that instant, and the trajectory to be followed, that generates a different optimization problem at each point of the path. It was also noted that, the whole simulation was performed 35% faster using the multi-core architecture.
The temporal analysis performed over specific structures of the TIMPC strategy that are time-invariant, in the prediction interval, in which the parallelization was carried out, showed an improvement of approximately 10% of the execution time when compared with the sequential code. Furthermore, analyzing the structures time-varying over the receding
4.2. RESULTS FROM PROVANT SIMULATOR

Figure 4.7: Time evolution of $\dot{\alpha}_R$ and $\dot{\alpha}_L$ states, using the TIMPC.

Figure 4.8: Control signals generated by the TIMPC.

horizon of the MPC, by using the proposed approach, the execution time was reduced approximately by 35% using multi-core processing.
Figure 4.9: Wind speed.

Figure 4.10: Time-Invariant MPC Runtime - Single-core

4.3 Results from PIL Simulation

To evaluates the TVMPC running in PIL, the same desired trajectory presented in the previous section is chosen. The performed path by the UAV on the scenario shown in
4.3. RESULTS FROM PIL SIMULATION

Figure 4.11: Time-Invariant MPC Runtime - Multi-core

Section 3.4 is presented in Figure 4.12, and the same statements made in the previous section about the controller are valid here. The TVMPC designed for multi-core processors were capable of solving the path tracking problem as show Figure 4.12. Small errors are presented in Figure 4.13, and null steady state error is obtained for the states $x$, $y$, $z$, and $\psi$. Furthermore, null steady state error is presented for the states $u$, $v$, $w$ in Figure 4.16, $p$, $q$, $r$ in Figure 4.17, $\dot{\alpha}_R$, and $\dot{\alpha}_L$ in Figure 4.18, meaning that this states stabilized in the equilibrium point. The angles $\theta$ in Figure 4.14, $\alpha_n$ and $\alpha_e$ in Figure 4.15, as exposed before, stabilized in a different point of the reference trajectory due the disturbances affecting the tilt-rotor UAV.

The multi-core architecture of the MPC was motivated by the improvements shown in the previous study. This architecture was implemented into the DE0-Nano-SoC Development Kit with dual-core Cortex-A9 embedded cores, using single-precision floating-point format, and simulated in process-in-the-loop with the same computer used in the previous experiment, as presented in Section 3.4.

The execution time of the TVMPC was measured while it was running in a sequential mode, and the mean time obtained was 12.53ms. Then, the execution time of running the parallel MPC strategy is measured, obtaining the mean time of 11.35ms. The 1.18ms is reduced in each calculation of the control signal.
Figure 4.12: Trajectory performed by the tilt-rotor UAV using the TVMPC.

Figure 4.13: Tracking error of $x, y, z$ and $\psi$, using the TVMPC.
4.3. RESULTS FROM PIL SIMULATION

Figure 4.14: Time evolution of the $\phi$, $\theta$ and $\psi$ states, using the TVMPC.

Figure 4.15: Time evolution of $\alpha_R$ and $\alpha_L$ states, using the TVMPC.
CHAPTER 4. SIMULATION RESULTS

Figure 4.16: Time evolution of the $u,v$ and $w$ states, using the TVMPC.

Figure 4.17: Time evolution of the $p,q$ and $r$ states, using the TVMPC.
4.3. RESULTS FROM PIL SIMULATION

Figure 4.18: Time evolution of $\alpha_R$ and $\alpha_L$ states, using the TVMPC.

Figure 4.19: Control signals generated by the TVMPC.
Figure 4.20: Wind speed.
5

Conclusion

This master thesis addressed the implementation of a parallel predictive controller designed to solve the trajectory tracking problem of an unmanned tilt-rotor aerial vehicle. The controller was optimized to be used in an embedded system and validated using Process-In-the-Loop (PIL) simulation.

Initially, the software architecture was developed for two cores in a computer, proposing the TIMPC parallelization that was studied in a simulator environment composed by Robot Operating System and Gazebo simulator.

From the obtained results of the time-invariant MPC, the study of time-varying MPC strategies was encouraged.

After, a multi-core software architecture of the TVMPC strategy for controlling a tilt-rotor UAV was developed. With the goal of improves the MPC execution time, a parallelization approach was applied creating a multi-core architecture capable of achieve the required performance, which is the path tracking and disturbance rejection, with a reduce execution time to compute the control signal.

To improve the time-varying control strategy, the terminal cost used in the MPC was made more robust to reduce the prediction horizon making a faster TVMPC, as showed in Alfaro (2016).

As the goal of this work is to obtain better results on MPC time execution to incorporate the controller in embedded systems of the real UAV, a PIL simulation platform to the tilt-rotor UAV 3.0 was created helping the verification of this work.

The obtained results showed an improvement of 1.18 ms on execution time of TVMPC.
It was not a big reduction on the MPC execution time, but in this case enough to drive the controller to spend less than 12 ms, the maximum execution time allowed by the UAV dynamic.

In future works, it would be interesting the study of a parallel implementation of the active set strategy to solve the quadratic problem solver, using the OpenMP.

About the PIL, it leads to a development of a real time system, a Hardware-In-The-Loop (HIL). Making possible the error/event treatments and hence verifying it with more accuracy the reliability of this system.


