Master of Science Thesis

Tilt-rotor Tailsitter Global Acceleration Control: Behavioural Cloning of a Nonlinear Model Predictive Controller

Alexis van Wissen

Delft University of Technology

TU Delft

Master of Science Thesis

Tilt-rotor Tailsitter Global Acceleration Control: Behavioural Cloning of a Nonlinear Model Predictive Controller

by

Alexis van Wissen

to obtain the degree of Master of Science at the Delft University of Technology, to be defended publicly on Wednesday December 6, 2023.

Student number:	4676793	
Thesis committee:	Prof. Dr. G.C.H.E. de Croon	TU Delft, Chair
	Dr. Ir. E.J.J. Smeur	TU Delft, Supervisor
	Dr. N. Eleftheroglou	TU Delft, External
	Z. Ma MSc.	TU Delft, PhD Candidate
Project Duration:	May, 2022 - December, 2023	
Faculty:	Faculty of Aerospace Engineering, Delft	

Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Contents

Acronyms

BRF	Body Reference Frame. 46
INDI	Incremental Nonlinear Dynamic Inversion. 23, 24, 31
IRF	Inertial Reference Frame. 46
LQR	Linear Quadratic Regulator. 22, 23
MAV	Micro Air Vehicle. i, 1, 2, 21, 22, 35
MPC	Model Predictive Control. 31
NDI	Nonlinear Dynamic Inversion. 23
NMPC	Nonlinear Model Predictive Control. 40
PID	Proportional Integrative Differential. 22, 29
RMSE	Root Mean Square Error. 31
SLMPC	Sequentially Linearised Model Predictive Control. 32
UAV	Unmanned Air Vehicle. 1, 26, 28, 30

Nomenclature

α	Angle of attack
α_0	Zero-lift angle-of-attack
α_l	Left propeller tilt angle
α_r	Right propeller tilt angle
α_{prop}	Propeller angle-of-attack
$\bar{C}_l$	Averaged lift coefficient
$\bar{q}$	Dynamic pressure
$\dot{\boldsymbol{x}}$	Derivative of the state $\boldsymbol{x}$
$\boldsymbol{v}$	Virtual control input
$\boldsymbol{u_0}$	Initial input vector
$\boldsymbol{u}$	Input vector
$\boldsymbol{x_0}$	Initial state vector
$\boldsymbol{x}$	State vector
η	Angle between L_1 vector and the velocity vector
γ	Waypoint bearing gain
$\langle p, q, r \rangle$	Angular velocity components
$\langle q_0, q_x, q_y, q_z \rangle$	Attitude quaternion components
$\langle V_x, V_y, V_z \rangle$	Inertial velocity components
$\langle x, y, z \rangle$	Position coordinates
ω	Angular velocity
ω_l	Left propeller angular velocity
ω_r	Right propeller angular velocity
ϕ	Roll angle
$\Phi^{(fv)}$	Aerodynamics coefficients matrix
ϕ_{in}	Inflow angle
ψ	Yaw angle
ρ	Air density
σ	Propeller solidity
θ	Pitch angle
φ_d	Desired bearing

φ_R	Bearing from the previous waypoint to the destination waypoint
φ_T	Bearing from the UAV to the destination waypoint
$\vec{a}$	Acceleration in the inertial reference frame
$\vec{D}_A$	Vector from CG to point A
$\vec{F}_g$	Gravitational force vector
$\vec{F}_I$	Inertial force vector
$\vec{F}_{aero}$	Aerodynamic force vector
$\vec{F}_{B_{aero}}$	Aerodynamic force in the body reference frame
$\vec{g}$	Gravitational acceleration in the inertial reference frame
$\vec{T}_B$	Thrust force vector in the body reference frame
$\vec{T}_I$	Thrust force vector in the inertial reference frame
$\vec{U}$	Input vector
$\vec{v}_B$	Free-stream velocity vector in the body reference frame
$\vec{X}$	State vector
A	State transition matrix
a	Airfoil lift-curve slope
a_{cmd}	3D Commanded acceleration
A_{disc}	Propeller disc surface area
$a_{s_{cmd}}$	Commanded lateral acceleration
B	Control matrix
c	Chord length
C_d	Drag coefficient
c_T	Thrust coefficient
C_{d_0}	Zero-lift drag coefficient
C_{L_α}	Lift slope coefficient
C_{y_0}	Lateral force coefficient
D	Propeller diameter
D_T	Distance from target
D_{CT}	Cross track error
$e(t)$	State error as a function of time
$F(\boldsymbol{x_0}, \boldsymbol{u_0})$	State effectiveness matrix
f_a	Average propeller force
F_B	Force in the BRF
F_{drag}	Drag force function

F_{lift}	Lift force function
g	Gravity acceleration
$G(\boldsymbol{x_0}, \boldsymbol{u_0})$	Control effectiveness matrix
I	Inertia matrix
J	Advance ratio
J_{cost}	Cost function
K	Optimal control feedback matrix
K_C	Cross track error gain
K_D	Path reaching gain
K_d	Derivative gain
K_i	Integral gain
K_p	Proportional gain
K_{hvr}	Reduced flow turning efficiency correction coefficient
L	Lift
L_1	Vector between vehicle and reference point
m	Mass
M_B^I	Transformation matrix from the body reference frame to inertial reference frame
N	Number of blades
P	Solution to Riccati equation
P_P	Propeller pitch
Q	State penalising coefficients matrix
R	Input penalising coefficients matrix
r	Radius
$r(t)$	Reference value as a function of time
S	Wing surface area
S	Wing surface area
T	Thrust
T^*	Look-ahead time
$u(t)$	Input value as a function of time
V_g	Ground speed
V_N	Normal velocity component
V_∞	Free-stream velocity
w_0	Induced velocity
$y(t)$	State value as a function of time
Z_{fwd}	Corrective aerodynamic force in forward flight
Z_{hvr}	Corrective aerodynamic force in hovering flight

1 Introduction

Quad-rotor drones are the first drone configuration that comes to mind when thinking of small UAVs. Their ability to perform agile manoeuvres and perform stationary flight has seen their popularity explode. However, their wingless configuration severely limits their energy efficiency and thus their range. This is why work has been performed on tail-sitter aircraft that have the benefit of being able to perform more efficient horizontal flight as well as stationary flight. The typical tail-sitter has two fixed rotors and controls itself through differential thrust for yaw control, and uses its flaps for pitch and roll. The problem with this configuration is that the maximum pitch and roll moments that can be achieved are limited by the airspeed over the aerodynamic control surfaces. Additionally, a complete loss of control authority for pitch and roll can be caused by descending fast whilst hovering as the airflow over the wing would be reversed.

The proposed solution to this problem is removing the aerodynamic control surfaces and purely controlling the tail-sitter through two tilt-rotors. This removes the dependence of pitch and roll control on airspeed and greatly increases the moments that can be generated which facilitates general control of the drone but also the transition from hover to horizontal flight. For a better idea of the configuration, a picture of the drone can be seen in Figure 1.1.

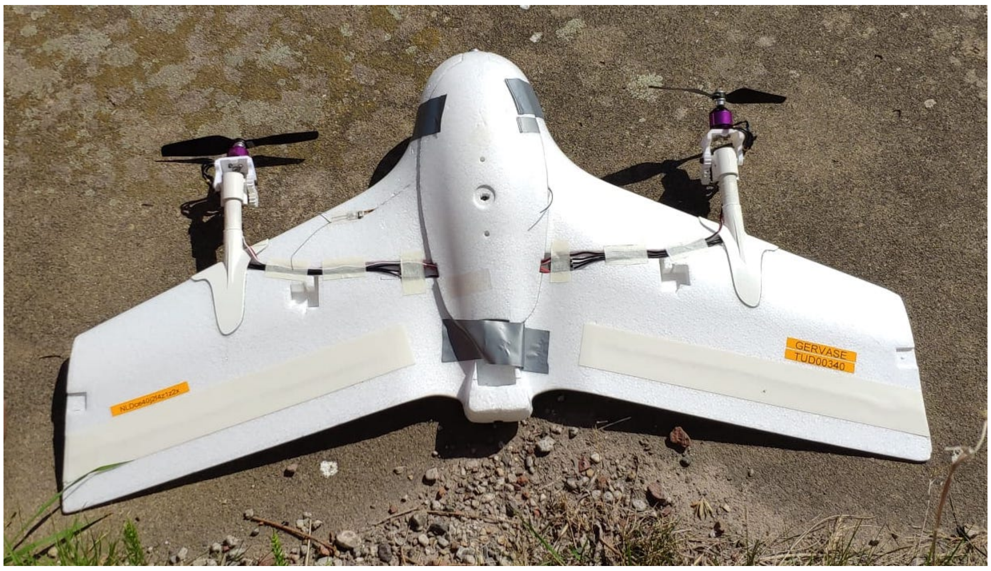


Figure 1.1: Tilt-rotor Tailsitter MAV

Some issues do arise from such a configuration. Mainly, the largely nonlinear behaviour of the system and the difficulty in modelling aerodynamic forces. To mitigate these issues and start developing a controller for this configuration, Incremental Nonlinear Dynamic Inversion was used to control the attitude in [1]. This acceleration-based controller allows for the system to be controlled by only needing a simplified model of the system and a control effectiveness matrix. The incremental nature of the control in turn deals with modelling discrepancies and external disturbances such as wind [2].

The novelty and new work that will be performed in this thesis will be related to figuring out the best way to implement acceleration control for the drone.

Research Objective:
To design and implement a global acceleration controller for a novel tilt-rotor tailsitter MAV.

Research Questions:

1. How can the acceleration of a tilt-rotor tailsitter MAV be controlled?
2. To what extent does the proposed acceleration controller for the tilt-rotor tailsitter MAV exhibit global effectiveness?
3. How can the global acceleration controller be practically implemented on the tilt-rotor tailsitter MAV?

Structure

In chapter 2, the academic paper summarising the main results from this thesis can be found. The paper can be read as a standalone document. Following the paper in chapter 3, the initial literature study is presented. Next in chapter 4, the results from a verification of the dynamics model used in the academic paper are shown. To finish, the main findings and answers to the research questions are summarised in chapter 5.

2

Academic Paper

Tilt-rotor Tailsitter Global Acceleration Control: Behavioural Cloning of a Nonlinear Model Predictive Controller

A.J.R. van Wissen, Z. Ma, and E.J.J. Smeur,
Delft University of Technology, 2628HS Delft, The Netherlands

***Abstract*—Capable of both vertical take-off and landing and forward flight, tail-sitters are a versatile class of UAVs with a large range of potential applications. A variant of tailsitters using tilt-rotors instead of ailerons for pitch and roll control has been proposed to mitigate the reduced control authority at low to zero velocities. The control of the translational dynamics for this platform is uniquely challenging. The extended flight envelope requires the controller to be able to perform hover and forward flight which are two flight phases with very different dynamics. Additionally, the tilt-rotor mechanism used to control the system is highly nonlinear which adds to the challenge. This paper presents a novel acceleration controller using Nonlinear Model Predictive Control (NMPC) in addition to the use of behavioural cloning to mimic the NMPC using a feedforward neural network. It is shown that behavioural cloning does successfully transfer general flight characteristics but that the performance is degraded with respect to the NMPC. Additionally, a sensitivity analysis was performed to investigate the effects of improper parameter estimation on controller performance. The most interesting result from this analysis is the strong sensitivity of both controllers to changes in centre of gravity location and mass.**

***Index Terms*—UAV, Tailsitter, Tilt-rotor, Acceleration Control, Nonlinear Model Predictive Control, Behavioural Cloning**

I. INTRODUCTION

There has been a surge in the use of micro air vehicles (MAVs) over the last few years with uses spanning a spectrum of civilian and military applications including inspection services, crop monitoring and surveillance tasks [1]. Some of these applications necessitate the combination of features seen in fixed-wing MAVs, such as long-range flight and high endurance, with the hovering and vertical take-off and landing abilities typically associated with rotorcraft. Hybrid MAVs cover these requirements by combining the ability to hover with a wing to have an efficient forward flight. A specific type of Hybrid MAV called tailsitter rotates their whole body 90 degrees to transition between flight phases. According to [2], this reduces the mechanical complexity of the system and adds minimal dead weight thanks to the use of fixed rotors and aerodynamic control surfaces. However, the use of aerodynamic control surfaces limits the ability to control roll and pitch at lower airspeeds. To resolve this issue, the addition of tilt-rotor mechanisms to the propellers and the removal of the ailerons has been proposed in [3], removing the dependence on airspeed for pitch and roll control.

Controlling such a system is difficult in part due to the large flight envelope that includes hover, transition and forward flight which means that the flight dynamics vary substantially. Adding to this, a nonlinear tilt-rotor control mechanism that has to control both attitude and position, the selection of a control method should take these challenges into account.

In [4], a fused-PID velocity controller was developed to control a tilt-rotor Vertical Take-Off and Landing (VTOL) drone. Two different PID controllers had to be designed to deal with hover and forward flight and were fused to deal with transition flight. This highlights a weakness of PID control where highly nonlinear systems require the design of multiple PID controllers to deal with the system dynamics.

In [5], the outer-loop control of a tilt-rotor drone was designed using a state-dependent LQR trajectory controller that controls the angular rates and thrust. The low-level control is made of multiple PID controllers requesting moments to achieve these angular rates. Although this method achieved moderately aggressive trajectories, the use of PID controllers in the low-level control would be challenging for this highly nonlinear platform.

Incremental Nonlinear Dynamic Inversion (INDI) is a good candidate for acceleration control thanks to its successful implementation on multiple similar fixed-rotor tailsitters. In [6], two INDI controllers controlling attitude and velocity individually were combined in a cascaded structure to successfully control a fixed-rotor tailsitter in all phases of forward flight. Additionally, in [7], an adaptation of INDI using differential flatness was successfully implemented to perform accurate and aggressive uncoordinated flight manoeuvers. The use of INDI was tested, but after several trials, a cascaded INDI approach showed difficulties in controlling the drone adequately. It cannot be guaranteed that it is impossible to use such an approach but the difficulties encountered were most likely caused by the high coupling between translational and rotational control on a tilt-rotor control mechanism. For example, using the tilt-rotor mechanism to fulfil an immediate acceleration requirement may be possible but could negatively affect the attitude of the drone in the future. It therefore became apparent that a solution that can deal with this coupling is needed to control this drone. Model Predictive Control is one such method that can optimise the control outputs to follow acceleration requirements without jeopardising its ability to do so in the future. The system being highly nonlinear warrants the use of Nonlinear Model Predictive Control (NMPC) [8].

To the best knowledge of the author, NMPC has not been used for the control of a tilt-rotor tailsitter before. NMPC is an elegant solution to the problem of controlling a tilt-rotor tailsitter thanks to its ability to follow objectives such as acceleration defined in a cost function. The nonlinear solver then minimises the value of the cost function to find the required control inputs. Through some initial testing, it became apparent that the NMPC may have difficulties running in real-time on hardware that can be placed on the drone.

Imitation learning is a method that can be used to solve this issue as explained in [9] where the behaviour of an expert NMPC controller was replicated using a neural network. A neural network can run substantially faster than the NMPC on much less powerful hardware. This is called imitation learning. Imitation learning at its core aims to replicate the behaviour of an expert controller which in this case is the NMPC mentioned previously. In a survey summarising the different imitation learning methods [10], three different techniques were identified: Behavioural Cloning, Direct Policy Learning and Inverse Reinforcement Learning.

Behavioural Cloning is the most straightforward method where the expert controller is used to generate input-output data pairs which the neural network will then try to replicate through training. Direct Policy Learning starts identically as Behavioural Cloning by training an initial policy but can request additional expert controller input-output pairs in situations where the original policy performs poorly. Inverse Reinforcement Learning consists in learning a reward function from expert demonstrations which in this case is the NMPC to then apply this reward function in a normal reinforcement learning framework.

For this work, Behavioural Cloning will be used as a framework for imitation learning thanks to its simplicity. This means that a feedforward neural network will be built and trained on input-output pairs generated by an NMPC controller.

The contribution of this paper is the development of an acceleration controller using nonlinear model predictive control for a tilt-rotor tailsitter simulation. Additionally, due to its high computational requirements, behavioural cloning is proposed and tested to mimic the behaviour of NMPC allowing it to be run on hardware that can be placed on the tilt-rotor tailsitter.

This paper is structured as follows. To start, the design of the vehicle simulation along with some definitions and conventions will be detailed in section II. Then, the reasoning behind the design of the general control structure is outlined in section III. The design of the NMPC and the NN used for behavioural cloning are outlined in section IV and section V respectively. Next, the performances of both controllers are assessed in section VI. Following this, a sensitivity analysis is performed in section VII. To finish, the results are discussed in section VIII and conclusions made in section IX.

II. Vehicle Simulation

The vehicle simulation will base itself on the tilt-rotor tailsitter shown in Figure 1 which was developed in [3]. The model will be used to test the developed control methods and will be composed of three major components: the aerodynamics model, the thrust model and the actuator dynamics.

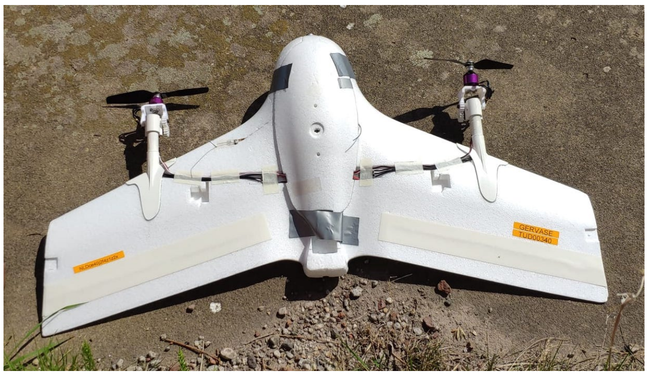


Fig. 1. Tilt-rotor tailsitter MAV [3].

A. Definitions and Conventions

Before diving into the modelling of the system, some definitions and conventions need to be defined.

Firstly, the inertial reference frame that is used is the North-East-Down (NED) reference frame. The cube in Figure 2 illustrates this, the positive X-axis points North, the positive Y-axis points East and the positive Z-axis points Down.

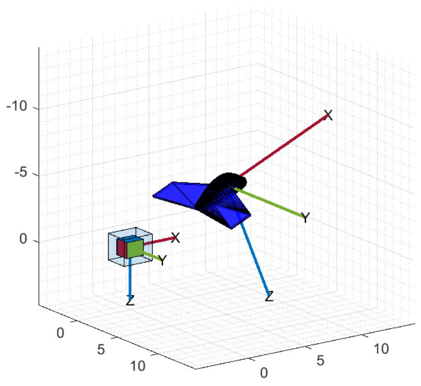


Fig. 2. Modelling reference frames: inertial reference frame (bottom left cube) and body reference frame (drone model).

Secondly, the body reference frame of the drone is centred at its origin with the positive direction of its X-axis pointing towards the nose of the drone and the positive direction of the Y-axis pointing perpendicularly along the wing. The Z-axis completes the right-handed axes. These axes are represented in Figure 2.

Finally, the representation of Euler angles in plots will not use the standard ”ZYX” rotation sequence but the ”ZXY” rotation sequence to avoid issues with representing attitude during hovering phases of flight.

B. Aerodynamics Model

In [7], a fixed-rotor tail-sitter was successfully controlled using INDI and differential flatness. The aerodynamic model used in this paper called ϕ theory is a singularity-free model that has been expressly developed for the ”algorithmic flight control of tail sitters" [11]. The objective of this model is not to be more accurate than other models but instead, trade some accuracy for a more consistent and numerically stable model. The authors of [11] also conclude that this model effectively captures the main dynamic characteristics throughout the whole range of flight conditions. This last point is important as it implies that if the ϕ theory model parameters are properly identified, the model will depict the main flight characteristics of the drone. The aerodynamic model

$$\vec{F}_{a_B} = \frac{1}{2}\rho S V_\infty \Phi^{(fv)} \vec{V}_B, \tag{1}$$

where $\vec{F}_{a_B}$ is the aerodynamic force vector in the body reference frame, ρ is the air density, S the wing surface area, V_∞ the free-stream velocity, $\Phi^{(fv)}$ the aerodynamics coefficients matrix and $\vec{V}_B$ the air velocity in the body reference frame.

The aerodynamic coefficients matrix $\Phi^{(fv)}$

$$\Phi^{(fv)} = \begin{bmatrix} C_{d_0} & 0 & 0 \\ 0 & C_{y_0} & 0 \\ 0 & 0 & C_{L_\alpha} + C_{d_0} \end{bmatrix}, \tag{2}$$

where C_{L_α} is the lift coefficient slope, C_{d_0} is the minimum drag coefficient and C_{y_0} the minimum lateral force coefficient.

C. Thrust Model

The thrust model consists of three steps. First, the static thrust T_0 is calculated using

$$T_0 = 5 \cdot 10^{-6}\omega^2 - 0.0008\omega + 0.1034, \tag{3}$$

where the propeller angular velocity ω is the input. This equation was determined experimentally by G.H.L.H. Prescod in [3].

Static propeller thrust is not what is experienced during actual flight as the inflow velocity of the air reduces the capacity of the propeller to produce force. Modelling this effect can be done by using the relationship shown in Equation 4 which is adapted from [12]. The effective thrust T_c is calculated using

$$T_c = T_0 \frac{v_e - v}{v_e}, \tag{4}$$

where T_0 is the static thrust discussed previously, v_e the translational velocity and v the inflow velocity. The translational velocity v_e shown in

$$v_e = \frac{\omega}{2\pi} P_P, \tag{5}$$

depends on the propeller angular velocity ω and propeller pitch P_P. It can be described as the velocity at which the propeller would progress in a viscous liquid. Finally, inflow velocity v is the velocity of the airflow perpendicular to the plane of rotation of the propeller. With the estimation of the effective thrust performed, the force vector in the body reference frame $\vec{F}_{T_{B_{l,r}}}$ can be calculated using

$$\vec{F}_{T_{B_{l,r}}} = \begin{bmatrix} T_{B_{l,r}} \cos(\alpha_{l,r}) \\ 0 \\ -T_{B_{l,r}} \sin(\alpha_{l,r}) \end{bmatrix}, \tag{6}$$

where $T_{B_{l,r}}$ is the effective thrust of the propeller and $\alpha_{l,r}$ is the angle of the propeller nacelle. l and r subscripts denote the left and right propellers. Figure 3 shows the positive definitions of $T_{B_{l,r}}$ and $\alpha_{l,r}$ in a simplified side view of the drone. The final thrust force $\vec{F}_{T_B}$ is calculated using

$$\vec{F}_{T_B} = \vec{F}_{T_{B_l}} + \vec{F}_{T_{B_r}}. \tag{7}$$

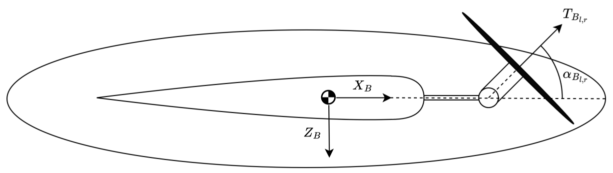


Fig. 3. Simplified drone side view: definition of $T_{B_{l,r}}$ and $\alpha_{l,r}$ in the body reference frame.

D. Actuator Dynamics

The actuators in this system are composed of the propeller motors of which the angular velocities are denoted ω and the tilting mechanism servos whose angles are denoted α. Including actuator dynamics in the simulation is important as in real life, control inputs are not replicated instantly in the system. Simple first-order dynamics will be used to model the actuators using the transfer function $H(s)$ shown in

$$H(s) = \frac{1}{\tau s + 1}, \tag{8}$$

where τ is the time constant. This also means that the actuators are now part of the state vector and the control inputs become commands for these actuators to follow. As an example, the calculation of the derivative of the left propeller angular velocity $\dot{\omega}_l$ using Equation 8 is shown in

$$\dot{\omega}_l = \frac{1}{\tau}(\omega_{l_c} - \omega_l) \tag{9}$$

where ω_{l_c} is the commanded angular velocity and ω_l is the current angular velocity of the propeller. The same method is used for the remaining actuators ω_r, α_l and α_r. As the focus of this work is not on complete modelling accuracy with the real-world drone, all actuators will have the same first-order dynamics for this proof of concept. The equation for these dynamics (Equation 8) uses τ which is the time constant and for this simulation is set to 0.04 seconds. This means that its settling time, which is the time it takes for the state to reach 95% of the commanded value, is 0.12 seconds.

E. Integrated Model

To summarise, both the aerodynamics and thrust model forces are combined to produce the equations of motion of the system. The translational equations of motion are defined in

$$\begin{bmatrix} \dot{V}_x \\ \dot{V}_y \\ \dot{V}_z \end{bmatrix} = \frac{1}{m} R_{IB}(\vec{F}_{T_B} + \vec{F}_{a_B}) + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}, \tag{10}$$

where $\vec{F}_{T_B}$ and $\vec{F}_{a_B}$ are the thrust forces and aerodynamics forces in the body reference frame, R_{IB} the reference frame transformation matrix from body reference frame to inertial reference frame and g the Earth's acceleration. The rotational equations of motion are defined in

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \frac{1}{I}(\vec{D}_{CP} \times \vec{F}_{a_B} + \sum_{i=l,r} \vec{D}_{T_i} \times \vec{F}_{T_{iB}} - \vec{\Omega} \times I\vec{\Omega}), \quad (11)$$

where I is a 3-by-3 matrix containing the moments of inertia of the drone around the x-, y- and z-axis on the diagonal of the matrix and $\vec{\Omega}$ the angular velocity vector $\langle p, q, r \rangle$. Equation 11 can be simplified to

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \frac{1}{I}(\vec{D}_{CP} \times \vec{F}_{a_B} + \sum_{i=l,r} \vec{D}_{T_i} \times \vec{F}_{T_{iB}}), \quad (12)$$

where the term $\vec{\Omega} \times I\vec{\Omega}$ is ignored as its contribution compared to the other terms is much smaller due to the low angular velocities expected in the simulation. Additionally, the following notation $\vec{D}_A$ indicates a vector starting at the centre of gravity and ending at position A. Performing the cross product of this distance vector with the force applied at the end of it generates vector moments in the respective axes. CP stands for Center of Pressure. In general, the Center of Pressure is not a fixed point, but the aerodynamics model [11] used in this paper calls for a fixed point at which the aerodynamic forces are applied and thus $\vec{D}_{CP}$ is constant. T_i designates the points at which the propellers generate their force. T_l refers to the left propeller and T_r the right propeller.

For further reference, the full state $\vec{X}$ and control inputs $\vec{U}$ of the system are defined

$$\begin{aligned} \vec{X} &= \langle \vec{x}, \vec{V}, \vec{q}, \vec{\Omega}, \omega_l, \omega_r, \alpha_l, \alpha_r \rangle \\ \vec{U} &= \langle \omega_{l_c}, \omega_{r_c}, \alpha_{l_c}, \alpha_{r_c} \rangle \end{aligned}, \quad (13)$$

where $\vec{x}$ is the position vector $\langle x, y, z \rangle$, $\vec{V}$ the velocity vector $\langle V_x, V_y, V_z \rangle$, $\vec{q}$ the quaternion vector $\langle q_0, q_x, q_y, q_z \rangle$ and finally $\langle \omega_l, \omega_r, \alpha_r, \alpha_r \rangle$ the actuator states. As for the control inputs, ω_{l,r_c} denotes the propeller angular velocity and α_{l,r_c} the propeller nacelle tilt angle. Additionally, the c subscript indicates that the value is the commanded value and not the actual state of the actuator.

F. Unmodelled effects and forces

As the objective of this work is to investigate the ability of the developed control methods to handle the main dynamic properties of a tilt-rotor tailsitter, some effects and forces were left unmodelled. The addition of these forces to the controllers in the future will be beneficial for modelling accuracy. This section will discuss these forces and how not modelling them affects the results that will be shown later.

1) Prop-wash: Determining the effects of prop-wash for a tilt-rotor tailsitter is difficult due to its novel nature and thus a lack of experimental data from the literature. The forces generated by the prop-wash differ greatly depending on the angle of the propeller nacelle. This can be visualised in

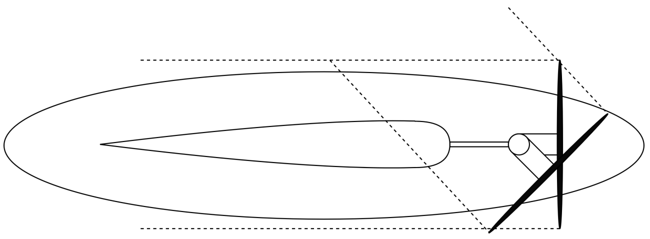


Fig. 4. Simplified prop-wash side view visualisation (dashed lines represent region affected by prop-wash).

Figure 4. The figure depicts a side view of one of the tilt-rotor propellers with two different angles of the propeller nacelle. The dashed lines represent approximately the regions where the flow has been accelerated based on propeller momentum theory [12]. The two different propeller nacelle angles generate vastly different regions affected by prop-wash which will interact with the wing differently. Modelling of prop-wash-induced forces is thus recommended for future work.

2) Propeller Angular Acceleration: Propeller angular acceleration generates torques that are not modelled in this simulation. The effect on the overall system is expected to be low due to the small inertia of the propellers. Therefore for this paper, it can be left unmodelled.

3) Propeller Nacelle Angular Acceleration: The rotation of the propeller nacelle is what makes this drone a tilt-rotor drone. In the model developed in this paper, the torque required to rotate the propeller nacelles is not modelled and therefore no torques are applied to the body of the drone when rotating the nacelles. This torque should be added to the model in further work.

4) Inflow Velocity to Drone Angular Velocity: Currently, the inflow velocity to the propellers is only calculated based on the point mass velocity of the drone. For steady-state flight, this does not cause any errors. However, because the propellers are not positioned at the centre of gravity of the drone when the drone rotates, this causes the propellers to move at different velocities than the centre of gravity. This effect should be modelled if aggressive manoeuvres are performed but this is not the case in this paper.

III. Control Structure

As the objective of this work is to develop an acceleration controller, the general control structure should be designed around this. The structure should also allow for easy switching between the NMPC controller and the neural network designed to mimic the NMPC. The proposed general structure is given in Figure 5. To calculate the target acceleration $\vec{a}_{ref}$, a proportional controller calculates the error between the target velocity $\vec{V}_{ref}$ and the actual velocity $\vec{V}$ and multiplies it by a gain $K_V = 0.8$ which was determined through trial and error. The components of $\vec{a}_{ref}$ are limited between -5 and 5 m/s^2 to avoid large acceleration requirements from large velocity errors. The acceleration controller then takes this target acceleration along with additional states from the drone $\vec{X}$ to calculate the control actuation required to achieve that acceleration. This is a very modular structure which allows for different acceleration controllers to be implemented as long as they have the same inputs and outputs.

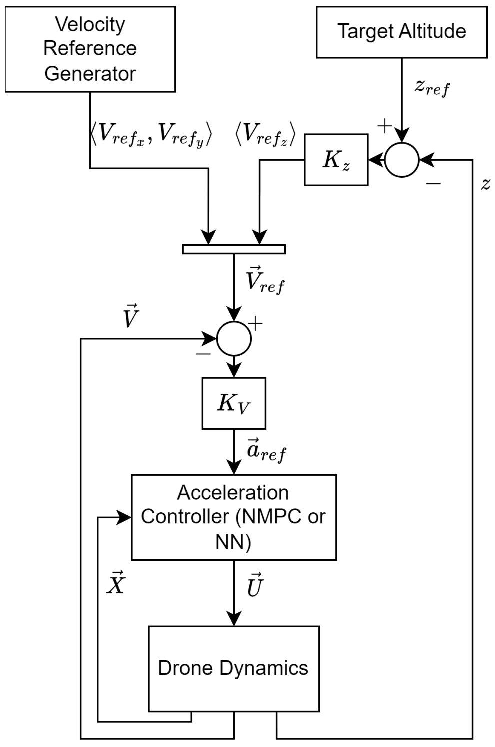

Velocity Reference Generator
Target Altitude
z_{ref}
$\langle V_{ref_x}, V_{ref_y} \rangle$
$\langle V_{ref_z} \rangle$
K_z
+
−
z
$\vec{V}_{ref}$
$\vec{V}$
+
−
K_V
$\vec{a}_{ref}$
Acceleration Controller (NMPC or NN)
$\vec{X}$
$\vec{U}$
Drone Dynamics

Fig. 5. General overview of control structure.

The velocity reference $\vec{V}_{ref}$ is generated by combining two different controllers. On the right in Figure 5 an altitude controller calculates the error between the drone z position and the target altitude z_{ref} which is then multiplied by a gain $K_z = 0.2$ (determined through trial and error) to get V_{ref_z}. To generate the velocity references $\langle V_{ref_x}, V_{ref_y} \rangle$, a waypoint-based method is used. Figure 6 illustrates how this is done. A line is drawn between the drone position and the waypoint location. Then, a unit vector can be built on this line and scaled depending on the desired target velocity which finally creates $\langle V_{ref_x}, V_{ref_y} \rangle$.

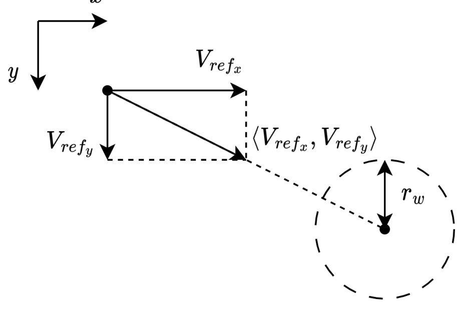


Fig. 6. Velocity reference generator.

The velocity reference generator switches to the next waypoint once the drone is within r_w distance of the current waypoint. Additionally, first-order dynamics in the same form as Equation 8 were added to the velocity reference $\vec{V}_{ref}$ to avoid large sudden changes. The time constant used for these dynamics is 0.5 seconds. One of the benefits of this overall structure is that minimal planning is needed for a mission. All that is needed is a target velocity and the location and order of the waypoints.

IV. Nonlinear Model Predictive Control (NMPC)

To define the Nonlinear Model Predictive Controller, a few elements are needed. The dynamics model, the cost function, the constraints of the system and the prediction parameters (Prediction horizon and sample time). The dynamics model used has been previously defined in section II. The cost function value J is defined as

$$J = K_a \cdot a_e + K_{pqr} \cdot \Omega_e - K_{cross} \cdot H_e, \tag{14}$$

where a_e is a term penalising acceleration error, Ω_e penalises excessive angular rates and H_e ensures the drone aligns its body XZ-plane with the reference velocity vector. Terms starting with K_x are simple weights used to balance the error terms. a_e is calculated using

$$a_e = \sum_{i=1}^{p} (a_{x_{ref_i}} - a_{x_i})^2 + (a_{y_{ref_i}} - a_{y_i})^2 + (a_{z_{ref_i}} - a_{z_i})^2, \tag{15}$$

where the x,y, and z accelerations are subtracted to their respective reference values denoted $a_{x_{ref}}$ then squared. The subscript i denotes the specific timestep and p is the prediction horizon which in this case is 10. The term Ω_e is calculated using

$$\Omega_e = \sum_{i=1}^{p} (p_i^2 + q_i^2 + r_i^2), \tag{16}$$

where the angular rates p, q and r are squared and summed over all timesteps in the prediction horizon. Ω_e was required in the cost function as without it, the NMPC has no incentive to find a stable attitude. This term being a secondary objective is weighted less than the acceleration error. The final term called H_e and which is defined in

$$H_e = \sum_{i=1}^{p} \left\| \frac{diag(1,1,0) \cdot \vec{V}_{ref}}{\left\| diag(1,1,0) \cdot \vec{V}_{ref} \right\|} \times \frac{proj_{xy}(\vec{Y}_B)_i}{\left\| proj_{xy}(\vec{Y}_B)_i \right\|} \right\|, \tag{17}$$

was designed to make the drone align its XZ-plane in the body reference frame with the current reference velocity vector. To make sure the alignment calculation works in all circumstances, a vector-based method is devised. Two vectors are built to calculate H_e. First, on the left side of the cross product in Equation 17, the x- and y-components of the reference velocity vector are extracted, and then this vector is normalised. On the right side of the cross-product, the y-axis of the drone in its body reference frame is projected onto the inertial xy-plane and then normalised. The cross product of both these vectors is taken then the norm of the results is summed for each timestep. The norm of the cross product of both these vectors produces the largest value of 1 when they are perpendicular and 0 when they are parallel. As this value is subtracted from the cost function, the solver will aim to keep the vectors perpendicular which is the desired behaviour. This current implementation can be adapted to take into account wind by modifying the $diag(1,1,0) \cdot \vec{V}_{ref}$ term with the x and y wind components.

It should be noted that no terms governing pitch or roll were added to the cost function. The reasoning behind this is that forcing the drone into specific attitudes may be less optimal than leaving the freedom to the solver to determine the best attitudes to minimise the cost function.

The next step to describe the NMPC is to set constraints on the system. Being able to set explicit constraints on both the state and control inputs is a big reason for the success of NMPC [12]. In this case, setting constraints on the control inputs is essential as there are physical limits to the capabilities of the actuators.

$$\begin{aligned} \omega_{l,r} \in &\quad [100, 1400], (rad/s) \\ \dot{\omega}_{l,r} \in &\quad [-500, 500], (rad/s^2) \\ \alpha_{l,r} \in &\quad [-\pi/4, \pi/4], (rad) \\ \dot{\alpha}_{l,r} \in &\quad [-\pi/9, \pi/9], (rad/s) \end{aligned} \tag{18}$$

Equation 18 summarises the constraints applied to the actuators. This minimal set of constraints also has its use in terms of improving computational performance. This has been shown in [13] where an initial set of nonlinear constraints was aggregated into a smaller number of nonlinear constraints which had the effect of saving significant computational time.

Finally, the prediction parameters which are the prediction horizon and the sample time are discussed. The main driver of computational complexity for model predictive control is the amount of decision variables [14]. The amount of decision variables is calculated by multiplying the control horizon c with the number of control inputs which in this case is 4. As in this work, the control horizon is equal to the prediction horizon, p can be used to calculate the amount of decision variables. Selecting the prediction parameters is therefore a balancing act between the controller and computational performance.

Through trial and error, a prediction horizon of 10 and a sample time of 0.1 seconds struck a good balance between computational performance and controller performance bringing the number of decision variables to 40.

A. Sub-stepping

It has been mentioned previously that the sample time for the NMPC was set to 0.1 seconds. In its current implementation, this means that the open-loop prediction of the NMPC is simulated at 10 Hz, which in this case causes issues related to the stability of the dynamics which causes the solver to crash in certain cases. This was confirmed by keeping the 0.1-second sample time for the NMPC but sub-stepping the integration steps at 0.01 seconds. Solver crashes were not detected after implementing this change. Further references to the NMPC using sub-stepping will be called Sub-stepping NMPC. It is important to note that all simulations were run at 100 Hz and that it only is the internal prediction model of the NMPC that was run at 10 Hz.

B. Real-time Performance

Applying this controller in real-time requires that the processing time of the whole control algorithm shown in Figure 5 must at least be less than the sample time δ of the NMPC. In practice, the objective is to get the processing time much smaller than the sample time. This is because the control input that is calculated is optimal for the system state at time t but due to the processing time, the state has changed and the control input is no longer optimal.

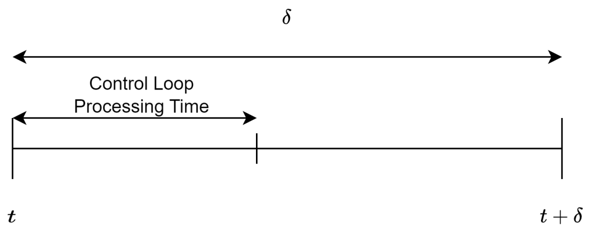
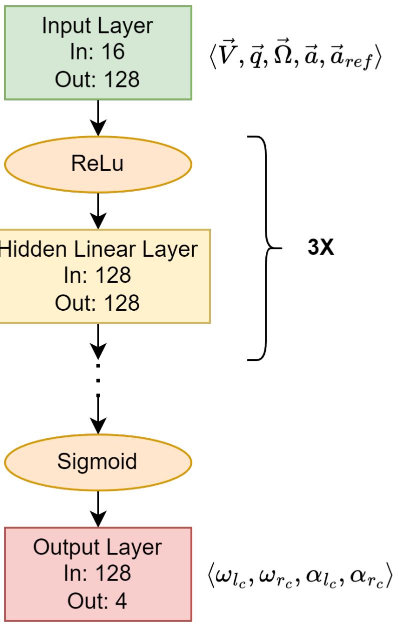


Fig. 7. Control loop processing time as a fraction of sample time overview.

The average optimisation time for the NMPC takes about 0.7 seconds with maximums reaching about 1 second. The hardware used is a desktop Intel® i5-12400 CPU. It is thus expected for hardware that can be placed on the drone to have difficulties running the NMPC in real-time. As mentioned in the introduction, a successful method of solving this issue is explained in [9] where the behaviour of an expert NMPC controller was replicated using a neural network. This method is called imitation learning and will be discussed next.

V. Imitation Learning Neural Network

The neural network (NN) used in this work is a feedforward network with three hidden layers. An overview of the structure is shown in Figure 8. The design choices will be divided into three parts: the NN inputs, the activation functions and the NN size.



Input Layer
In: 16
Out: 128
$\langle \vec{V}, \vec{q}, \vec{\Omega}, \vec{a}, \vec{a}_{ref} \rangle$
ReLu
Hidden Linear Layer
In: 128
Out: 128
3X
Sigmoid
Output Layer
In: 128
Out: 4
$\langle \omega_{l_c}, \omega_{r_c}, \alpha_{l_c}, \alpha_{r_c} \rangle$

Fig. 8. Neural network structure diagram.

1) NN Inputs: The input used in the NN consists of states of the drone whose values affect the forces applied to the drone. This includes the velocity vector $\vec{V}$, the quaternion attitude vector $\vec{q}$ and the angular rates $\vec{\Omega}$. The reasoning behind this is that the NN needs to have access to the states of the drone that affect the forces applied for it to be able to control acceleration. This is why position is not an input to the NN as it does not influence the acceleration in the model. Additionally, as this is an acceleration controller, the current acceleration $\vec{a}$ and reference acceleration $\vec{a}_{ref}$ are added to the input. One would be tempted to subtract the acceleration $\vec{a}$ from the reference acceleration $\vec{a}_{ref}$ to reduce the number of inputs of the NN. However, this was implemented and hurt controller performance. This is possibly explained by the loss of information that using the acceleration error causes as the absolute values of the acceleration and reference acceleration are lost.

2) Activation Functions: Looking at Figure 8 it can be seen that two different activation functions were used in the NN. For the hidden layers, ReLU activation was used as it is the first go-to activation function for hidden layers [15]. As for the activation of the output layer, a required property of the activation function is that it should be bounded. This is to ensure that the outputs of the NN do not exceed what the actuators can physically achieve. Therefore, a Sigmoid activation function was used and its output was scaled to fit the range of values the actuators may take.

3) Hidden Layers and Neurons: Choosing the number of hidden layers and neurons was performed through trial and error and ended up with three fully connected hidden layers with 128 neurons each.

A. Training data generation

The initial training data generation objective was to generalise as much as possible. This meant randomly sampling the drones' possible states as well as the reference accelerations that are used as input. The NMPC would then calculate the optimal control output for this situation and a data point would be generated.

Provided enough samples of the state and references to follow are generated, it was assumed that the network could learn from this and generalise. This was however not the case, the network could not learn from this random sampling method. A possible reason for this is that generalised random sampling over all states and reference accelerations may require substantially more training data than was generated for the neural network to be able to generalise. Therefore, it was decided to create a method that generates more specific training data to reduce the amount of data generation required. This method involved the generation of randomised flight scenarios. The basic design of these scenarios is for the drone to start in a randomised orientation and follow a randomised velocity vector. The drone is given a set amount of time to reach this velocity, then the velocity reference is set to zero for which it has the same amount of time to reach. By generating a large number of scenarios, the neural network should be able to generalise and perform scenarios that it has not encountered in the training data.

TABLE I
TRAINING DATA GENERATION SCENARIO INFORMATION.

	Velocity Range 1	Velocity Range 2
Target Velocity Range [m/s]	[0.5, 5]	[5, 11]
Number of Scenarios [-]	4454	3895
Number of Generated Input-Output Pairs [-]	356560	389500
Scenario Time [s]	8	10
Starting Yaw Range [rad]	$[-\pi/2, \pi/2]$	$[-\pi/2, \pi/2]$
Starting Pitch Range [rad]	$[\pi/4, 3\pi/4]$	$[\pi/4, 3\pi/4]$
Starting Roll Range [rad]	[0,0]	[0,0]
Reference Flight Path Angle Range [rad]	$[-\pi/2, \pi/2]$	$[-\pi/8, \pi/8]$

Table I summarises the scenario parameters used to generate the data. Overall, the training data covers target velocities starting at 0.5 m/s up to 11 m/s by combining two different ranges of training data shown in Table I. The training data does not go beyond 11 m/s for practical reasons linked to the time it takes to generate the training data. The training data summarised in Table I took about 80 hours to generate.

What both target velocity ranges have in common are the initial attitude ranges. The idea behind randomising the initial attitudes is simply to generate more diverse training data for the NN to train on.

The use of different reference flight path ranges depending on the target velocity has to do with the flight envelope used for the performance tests presented later in this paper. All tests have a constant altitude hold which means that the flight path angle for all tests is technically zero. In practice, the NN controller needs to be able to correct for vertical error so training data where the flight path angle is non-zero needs to be generated. At slow velocities in the range of 0.5 to 5 m/s, the limits on flight path angle are $[-\pi/2, \pi/2]$ radians which essentially means no limits. This was done to allow the NN to move the drone in any direction during hover. For the range of 5 to 11 m/s, the flight path angle range was reduced to $[-\pi/8, \pi/8]$ radians to improve training data generation efficiency.

B. *Training*

With the NN defined and the training data generated, the NN can now be trained. Setting up the training of a neural network requires the definition of multiple settings. These are the loss function, optimiser, learning rate, batch size and number of epochs. The loss function used to calculate the error between the prediction of the NN and the output of the NMPC is the Mean Square Error (MSE) defined in

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{19}$$

with n being the number of samples, Y the predicted output and $\hat{Y}$ the true output. It was selected due to its common use in feedforward neural networks used for regression tasks [16, 17, 18].

After calculating the MSE, the weights and biases of the neural network are updated using the Adam optimiser which is a gradient-based optimiser known to be able to deal with problems involving large amounts of data whilst remaining efficient computationally [19]. Additionally, [19] recommends a learning rate of 0.001 as a good starting point.

The batch size and number of epochs were determined through trial and error. A summary of these training settings is presented in Table II.

TABLE II
NEURAL NETWORK TRAINING SETTINGS.

	Value
Loss Function	Mean Square Error (MSE)
Optimiser	Adam [19]
Learning Rate	0.001
Batchsize	512
Number Epochs	50

The final step before training the model is to divide the data into training and test datasets. According to [20], an 80%/20% split is commonly used in machine learning and thus is used as a good starting point. Splitting the data is done to verify the generalisation ability of the network throughout training. If the test average loss starts to diverge from the training average loss it is a possible sign that the NN is starting to overfit the training data. The resulting loss plot for the NN used in this paper is shown in Figure 9.

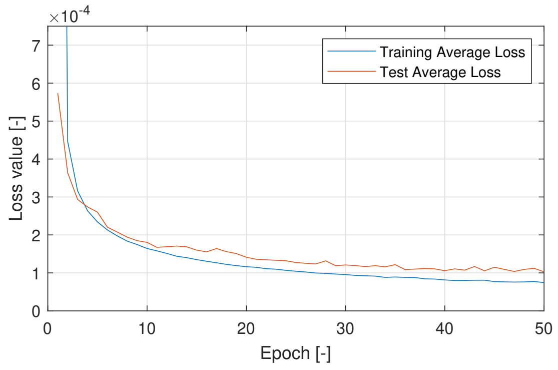


Fig. 9. Average training and test loss plot.

C. *Real-time performance of onboard hardware*

The development of this neural network was done in the interest of real-time performance. Therefore, it is necessary to test its real-time performance on hardware that could be placed onboard a drone. In this case, the neural network was tested on a Raspberry Pi® 4 Model B with 4 Gigabytes of RAM.

The model in its PyTorch format is first converted to ONNX (Open Neural Network Exchange) format for performance improvements. At this time, the ONNX Runtime module for Python has better-optimised inference for the ARM-type processor of the Raspberry Pi 4 than PyTorch itself. The converted model is then run 1,000,000 times and the computational time for each inference is recorded. The results of this test are presented in Table III.

TABLE III
NEURAL NETWORK REAL-TIME PERFORMANCE TESTING RESULTS ON RASPBERRY PI 4B.

	Value	Unit
Mean	9.4029e−5	s
Standard Deviation	1.6026e−3	s
Maximum	1.1261e−3	s

Typical control loops run at about 100-200 Hz which gives 0.005 to 0.01 seconds as maximums for computational time. The maximum computational time of 1.1261e−3 seconds which is the worst-case scenario is still smaller than these values. This leaves a safety factor and room for the neural network to grow in size if needed. It should be noted that the tests were performed with no other processes being run on the Raspberry Pi. If other tasks need to be run alongside the inference, this testing should be performed again as the additional load might affect performance.

VI. PERFORMANCE COMPARISON

This section will assess the differences in performance between the expert NMPC and NN controller. The performance of the Sub-stepping NMPC will be succinctly discussed for each test but not compared in depth with the NMPC and NN as it was not used for the generation of the training data. Additionally, test failures will be classed into three types: Solver, Performance and Crash failure. Solver failure is only relevant to the NMPC and indicates that the test failed because the solver did not converge to a solution. Performance failure indicates that a waypoint was missed or that the drone is circling around waypoints before reaching them. Crash failure indicates that the controller no longer had control of the drone and crashed. For each test, a table with the maximum velocities each controller could perform the test up to is presented. An additional column contains the failure mode of the controller when that maximum velocity is surpassed.

A. *Maximum Forward Velocity*

This first simple test will assess the maximum forward velocity in the X-direction the controllers can achieve. The results are summarised in Table IV.

TABLE IV
MAXIMUM FORWARD VELOCITY EXPERIMENT: NN, NMPC AND SUB-STEPPING NMPC RESULTS WITH FAILURE MODES PAST THE MAXIMUM VELOCITIES

	Velocity [m/s]	Failure Mode
NN	8	Crash
NMPC	19	Solver
Sub-stepping NMPC	24.5	None (Actuator Saturation)

It is important to note that the training data generated for the NN only included scenarios with velocities up to 11 m/s which would make it very unlikely for the NN to fly past this velocity. In this maximum velocity test, the NN is able to reach around 70% of this maximum theoretical velocity.

The NMPC and Sub-stepping NMPC reach higher velocities as they are not limited by the training data velocity ranges. The NMPC cannot make it past 19 m/s due to the solver not converging and the Sub-stepping NMPC stops accelerating at 24.5 m/s due to the saturation of the propeller angular velocities.

To assess the transfer or flight control logic between the NMPC and the NN, the maximum velocity of the NN will be used as the target velocity for both. Looking at Figure 10, the NMPC reaches the target velocity faster and settles to the target velocity after 5 seconds. Its attitude also remains very flat with no yaw or roll to be seen. In comparison, the NN takes about 4 seconds to reach the target and remains slightly offset from the target velocity. It also does not fly completely flat with small variations in yaw and roll throughout.

Both controllers show very smooth changes in actuator states as shown in Figure 11. For the NMPC this is not surprising as rate limits are applied when calculating the values for the actuators. For the NN, behaviour is learned through the training data and thus shows that it is able to mimic the behaviour of the NMPC.

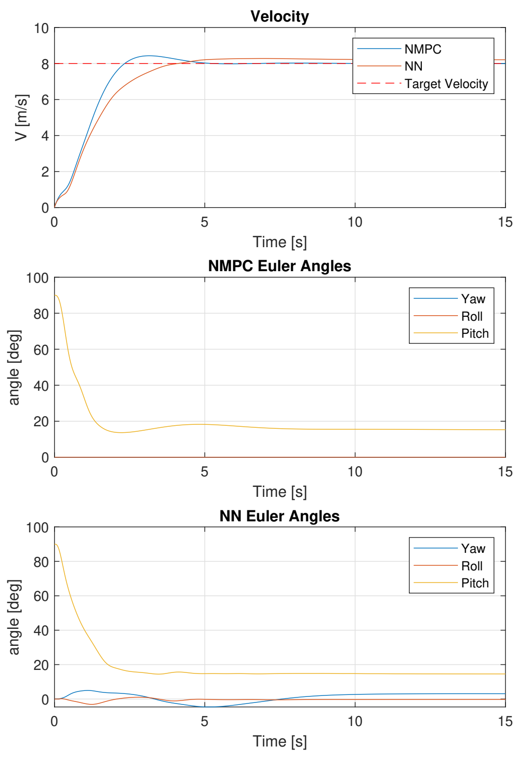


Fig. 10. Maximum forward velocity experiment: velocity and Euler angles of the NMPC and NN.

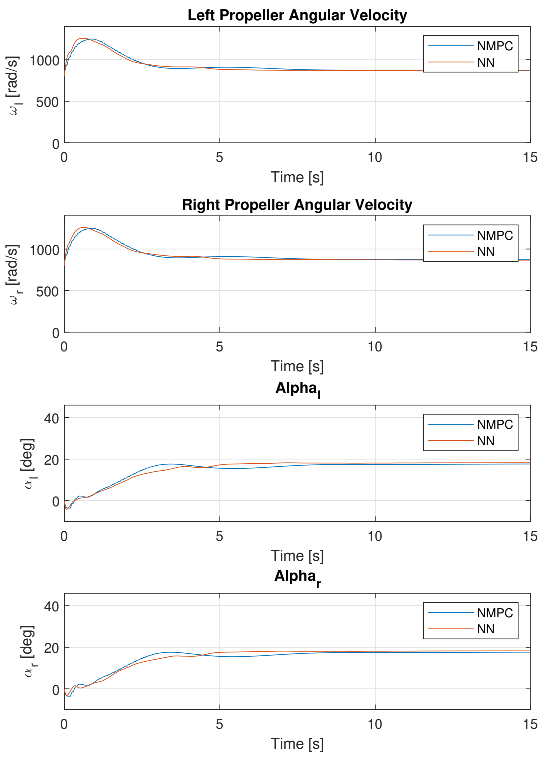


Fig. 11. Maximum forward velocity experiment: control actuator states.

B. *Hover to Forward Flight back to Hover*

This test involves finding the fastest forward velocity at which the controller can accelerate to then slow back down to hover. The results are summarised in Table V.

TABLE V
HOVER TO FORWARD FLIGHT TO HOVER EXPERIMENT: NN, NMPC AND SUB-STEPPING NMPC RESULTS WITH FAILURE MODES PAST THE MAXIMUM VELOCITIES

	Velocity [m/s]	Failure Mode
NN	8	Crash
NMPC	8	Solver
Sub-stepping NMPC	24	None (Actuator Saturation)

Both the NN and the NMPC managed to perform this task up to 8 m/s. This is one of the rare cases where the maximum target velocities are identical. The Sub-stepping NMPC having the advantage of not experiencing solver issues manages to perform this test up to 24 m/s. It does not have a failure mode because it simply cannot reach 25 m/s as previously seen in the maximum forward velocity test.

Just like the maximum forward velocity test, the NN can perform this test over about 70% of the velocity range it was trained on. It can be argued that although its maximum velocity is lower, the fact that it can slow back down to a hover from its maximum velocity is safer than a system such as the NMPC which can fly much faster but not safely slow down.

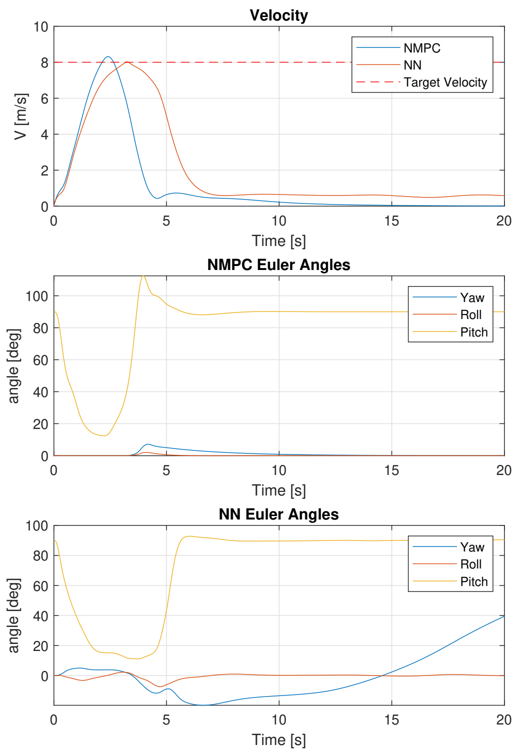


Fig. 12. Hover to forward flight to hover experiment: velocity and Euler angles.

Looking at Figure 12, some assessments on control logic transfer can be made. The NN reached the target velocity with about a 1-second delay compared to the NMPC. This is reflected in Figure 13 where the sharp change in control inputs for the slow-down procedure is delayed over that of the NMPC. Interestingly, the velocity profile and control inputs of the NMPC and NN are similar up until about 2 seconds when the NN starts to deviate and slows down when approaching the target velocity. Additionally, the NN never slows back down to zero velocity after it has reached the target velocity. This slower response with respect to the NMPC shows degradation of the NMPC control logic through the imitation learning process. This strengthens the need to improve the cloning process of the NN.

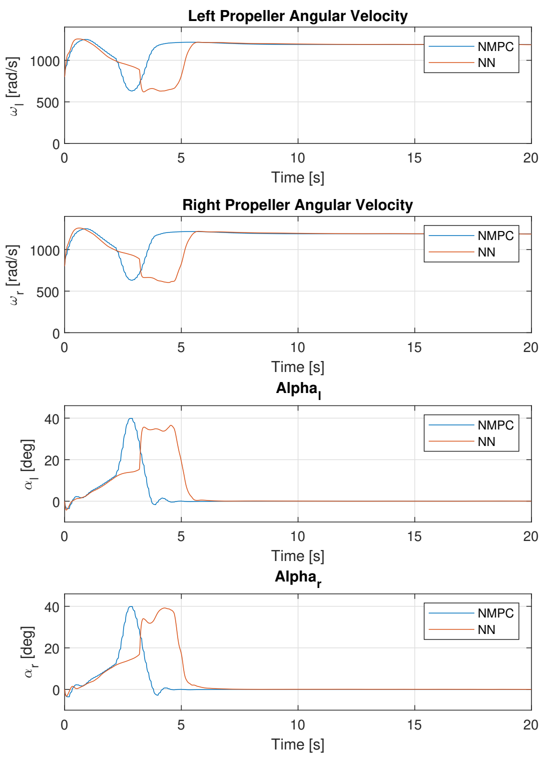


Fig. 13. Hover to forward flight to hover experiment: control actuator states.

C. Circular Waypoint Following

This test will investigate the ability of both controllers to perform a more complex trajectory. This trajectory is a circle with a radius of 40 meters and is composed of 14 waypoints for the drone to follow. The waypoint radius r_w is 5 metres.

TABLE VI
CIRCULAR WAYPOINT FOLLOWING EXPERIMENT: NN, NMPC AND SUB-STEPPING NMPC RESULTS WITH FAILURE MODES PAST THE MAXIMUM VELOCITIES

	Velocity [m/s]	Failure Mode
NN	6	Crash
NMPC	10	Solver
Sub-stepping NMPC	11	Performance

The NN has managed to complete this test up to 6 m/s after which the drone would lose control and crash. As summarised in Table VI, the NMPC and Sub-stepping NMPC performed similarly but past these velocities failed very differently. The NMPC fails by missing a waypoint, after which, the solver fails and the simulation stops. The Sub-stepping NMPC does not experience a solver failure but instead misses the waypoint and circles around it until it reaches it. It then moves to the next waypoint where it does the same.

To properly compare the NMPC and NN, the maximum velocity of the NN will be used as the target velocity for comparison in this test.

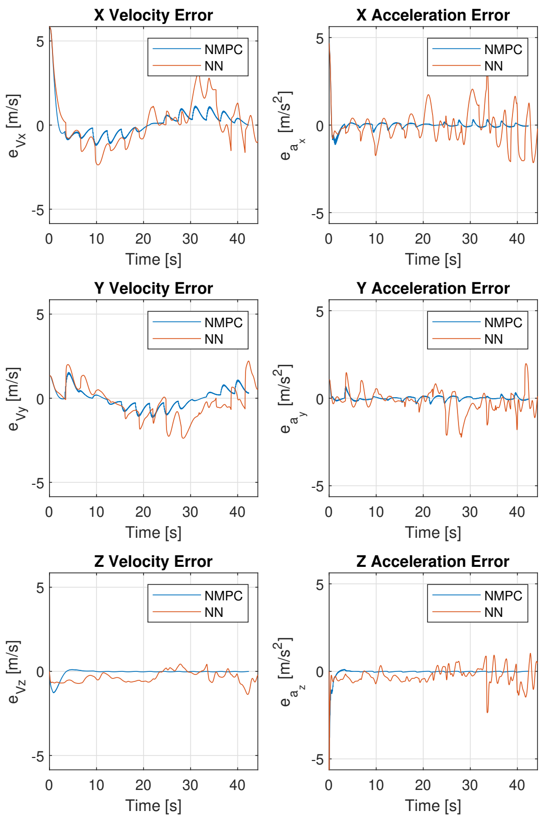


Fig. 14. Circular trajectory experiment: velocity and acceleration errors.

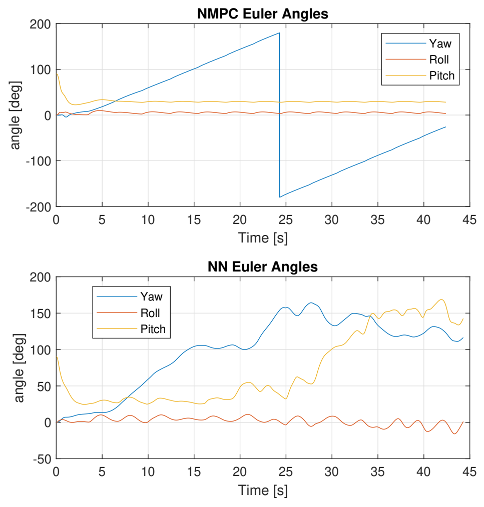


Fig. 15. Circular trajectory experiment: NMPC and NN Euler angles. NN flies upside down after t=25 seconds.

A general characterisation of the NMPC performance is that it has accurate tracking of the succeeding waypoints confirmed by small velocity and acceleration errors seen in Figure 14. Along with a smooth continuous change in yaw around the circle visible in Figure 15 in the top plot.

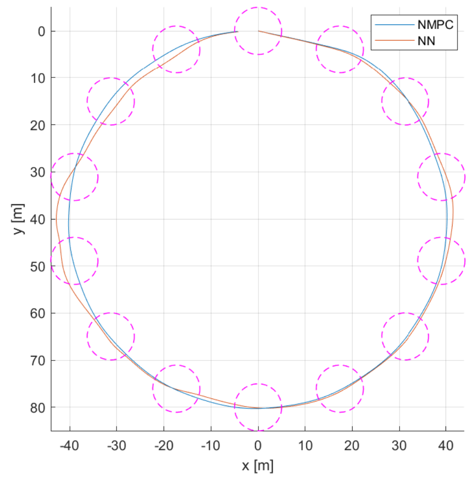


Fig. 16. Circular trajectory experiment top view (target velocity = 6m/s, circle radius = 40m).

The NN, relative to the NMPC, performs worse. It cannot maintain a constant altitude and is not as consistent in tracking the waypoints as visible in Figure 16 and Figure 17. This is characterised by larger velocity and acceleration errors shown in Figure 14. Both controllers present recurrent sudden changes in velocity errors in the X and Y directions. This is likely due to the waypoint velocity reference generation system. With these peaks showing in both controllers and the periods between peaks also being similar between controllers, the switching of the current waypoint to the next waypoint would explain the spikes in velocity errors and the periodic nature of these spikes explained by the equal distance between waypoints.

Interestingly, looking at Figure 15, the progression of the Euler angles for the NN during the test is very different than the NMPC. At the XY coordinates [-25 -70] in Figure 16, the NN has the drone flying in knife-edge flight. It then reaches the leftmost part of the circle where it flips and starts to fly upside down. This can be observed in Figure 15 in the bottom plot. Between 28 and 32 seconds, the pitch is near vertical which is when the drone is flying at knife edge. After 32 seconds, the pitch continues to increase indicating that it is now fully flying upside down.

This type of behaviour is not unexpected from either controller. For the NMPC, this can be explained by the fact that there are no terms in the NMPC governing attitude, the NMPC will simply try to minimise acceleration error. Adding to this that the model is symmetrical, there is no performance benefit for the NMPC to fly ”upright” and thus will fly upside down when needed. The training data used for the NN has an equal amount of ”upright” flight scenarios and upside-down flight scenarios. Thus confirming that the ability of the NN to fly upside down is inherited from the NMPC.

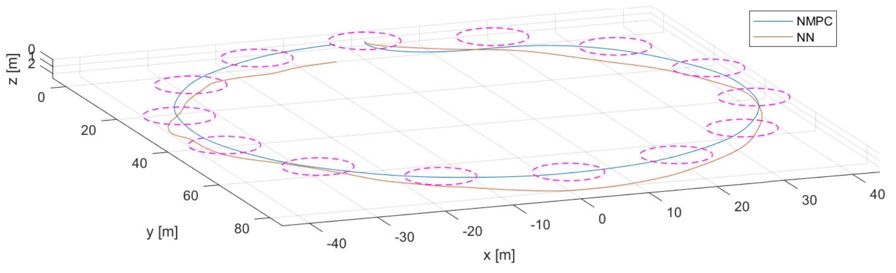


Fig. 17. Circular trajectory side view (target velocity = 6m/s, circle radius = 40m).

D. Rectangular Waypoint Following

This test continues the evaluation of both controllers in more complex trajectories. In this case, a rectangular trajectory to test the ability of the controllers to perform 90-degree turns. The waypoint radius r_w is 5 metres.

TABLE VII
RECTANGULAR WAYPOINT FOLLOWING EXPERIMENT: NN, NMPC AND SUB-STEPPING NMPC RESULTS WITH FAILURE MODES PAST THE MAXIMUM VELOCITIES

	Velocity [m/s]	Failure Mode
NN	5	Crash
NMPC	8	Solver
Sub-stepping NMPC	24	Performance

Looking at the summarised results in Table VII, the NN has only managed to perform this trajectory up to 5 m/s which is less than half the range of velocities included in the training data. The NMPC is again limited by a solver failure that can be mitigated by the use of sub-stepping. Concerning the Sub-stepping NMPC, what is meant by ”Performance” as a failure mode in Table VII is related to the physical limits of the system. It has been calculated that the minimum turning radius the drone can perform at 24 meters per second is around 56 meters. With the distance between waypoints being 40 meters and in perfect circumstances, a minimum turning radius of 56 meters, the tracking of the waypoints is degraded as the distance between waypoints is too small for this velocity.

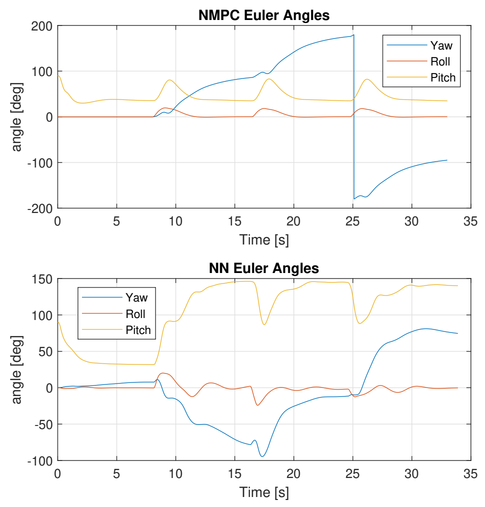


Fig. 18. Rectangular trajectory experiment: NMPC and NN Euler angles. NN flies upside down after t=8 seconds.

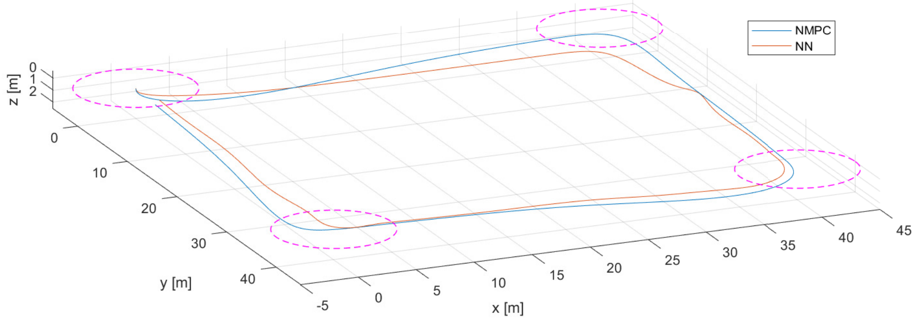


Fig. 20. Square trajectory side view (target velocity = 5m/s, square size = 40m).

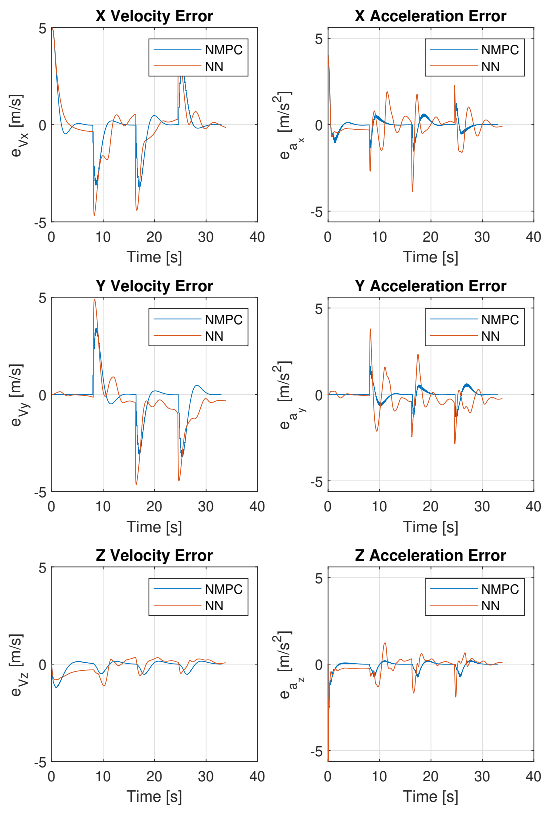


Fig. 19. Rectangular trajectory experiment: velocity and acceleration errors.

Using as target velocity the maximum velocity of the NN for this test, the NMPC and NN are compared. Both controllers present large pitch-up movements at around 8 seconds in Figure 18 which coincide with large X and Y acceleration errors in Figure 19 caused by the change in waypoint as mentioned in the circular test. A possible explanation for this effect is that the drone is not required to maintain the target velocity that is given. This means that to transition to a new velocity vector, the controller may slow down the drone to then turn and accelerate to the desired speed. If smooth turns by banking the drone are required in future work, the NMPC will have to be adapted to perform this new behaviour.

Looking at Figure 20 and Figure 21, the NN does not

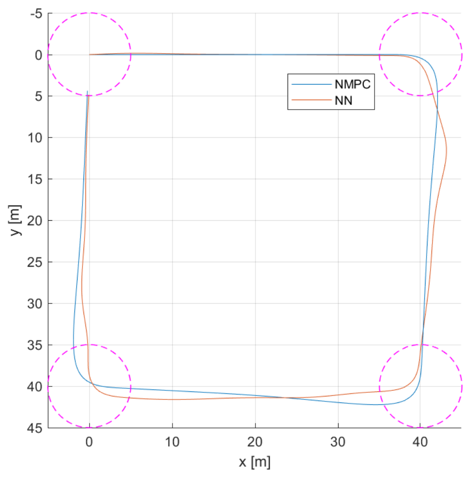


Fig. 21. Square trajectory top view (target velocity = 5m/s, square size = 40m).

smoothly reach each waypoint. This difficulty is reflected in the velocity errors shown in Figure 19. The controller manages to correct large disturbances but has trouble reducing them to zero like the NMPC can and therefore shows some offsets. These are most noticeable in Figure 19 in the Y Velocity Error plot.

Finally, looking at Figure 18 and the pitch angle of the NN, it can be seen flipping to upside-down flight just like the circular test. Most likely for the same reason as the previous test. To perform turns, the controller pitches up the drone significantly. In this case, over 90 degrees to a point where the controller transitions to upside-down flight.

VII. Sensitivity Analysis

The sensitivity analysis will assess the effect of improper parameter estimation on controller performance. One way to perform a sensitivity analysis for a model-based controller such as NMPC would be to change the parameters used in the NMPC internal model and keep the simulation model parameters constant. However, this is not possible for the NN as it is trained with the original parameters from the NMPC. Therefore, the simulation model parameters will be changed instead to assess the effects on performance. A flight path including a 90-degree turn and half a circle path will be used to compare performance between the simulations. Simulations where there are no changed parameters are called nominal. A top view of the waypoints defining this flight path is presented in Figure 22 depicted with magenta-coloured circles along with the nominal trajectories of the NMPC and NN.

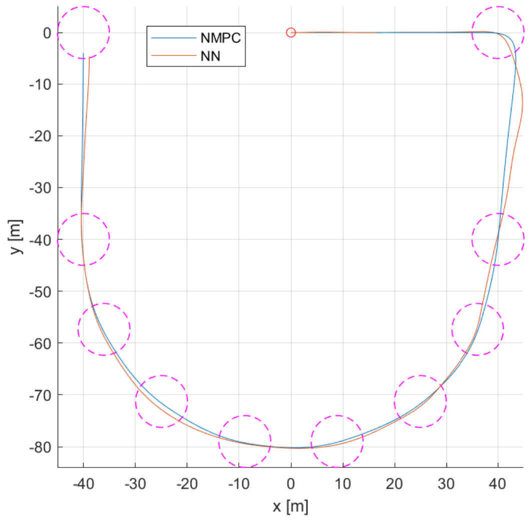


Fig. 22. Waypoint top view with nominal trajectories.

The parameters that will be analysed are summarised in Table VIII with their initial value, the percentage change that will be applied (if applicable) and the equivalent value of this percentage change. These parameters are the aerodynamic coefficients C_{L_α}, C_{d_0} and C_{y_0}, the wing surface area S, the mass m, the moments of inertia I_{xx}, I_{yy} and I_{zz} and the x-position of the center of gravity CG_x (in the body reference frame).

TABLE VIII
SENSITIVITY ANALYSIS PARAMETERS.

Parameter	Value	% Change	Value Change
C_{L_α} [-]	4	±10	±0.4
C_{d_0} [-]	0.05	±10	±0.005
C_{y_0} [-]	0.05	±10	±0.005
S $[m^2]$	0.26	±5	±0.013
m $[kg]$	1.27	±5	±0.0635
I_{xx} $[kg.m^2]$	0.065	±10	±0.0065
I_{yy} $[kg.m^2]$	0.009	±10	±0.0009
I_{zz} $[kg.m^2]$	0.0662	±10	±0.00662
CG_x $[m]$	0	N.A.	±0.01

To assess the results of these simulations, two tools will be used. First, visual results are presented as two figures depicting the trajectories of each simulation, the first of which Figure 23 shows an overview of all the NMPC trajectories and the second Figure 24 shows all of the NN trajectories. Secondly, a performance metric based on the error between the desired velocity and the actual velocity of the drone will be calculated for each simulation and presented in Table IX. Using a metric based on position would not be consistent across tests due to the waypoint-based method of generating velocity references. First, a Mean Absolute Error (MAE) velocity vector $\vec{V}_{MAE}$ for each simulation is calculated using

$$\vec{V}_{MAE} = \frac{\sum_{i=1}^{n} \left| \vec{V}_{ref_i} - \vec{V}_i \right|}{n}, \tag{20}$$

where n is the total amount of timesteps in the simulation, $\vec{V}_{ref}$ the reference velocity vector and $\vec{V}$ the drone velocity vector. From these MAE vectors, a velocity error metric e_V can be calculated using

$$e_V = \sum_{i=1}^{3} (\vec{V}_{MAE_t} - \vec{V}_{MAE_m})_i, \tag{21}$$

where the sum from $i = 1$ to 3 simply means adding the components of the vector, the subscript t in $\vec{V}_{MAE_t}$ denotes the MAE for the simulation with the modified parameters and the subscript m in $\vec{V}_{MAE_m}$ denotes the simulation where there is no difference between the controller parameters and simulation parameters.

The velocity error metric for each parameter change is documented in Table IX for both the NMPC and NN. As each cell containing the error metric represents a simulation, a colour will be assigned to each cell based on the success or failure of the simulation. Each colour will now be explained:

1) Success: Mission completed. All waypoints were reached and performance remained acceptable.
2) Performance Failure: Reaches all waypoints but in a ”non-direct” way. An example of this type of failure is the purple trajectory in Figure 24.
3) Failure: Drone stays in flight but does not complete waypoints
4) Total Failure: Crash / Drop out of the sky
5) Solver Failure: The NMPC solver cannot converge to a solution and fails

TABLE IX
SENSITIVITY ANALYSIS RESULTS.

Parameter	Value	NMPC e_V	NN e_V
C_{L_α} [-]	3.6	0.5036	-0.0959
C_{L_α} [-]	4.4	0.3905	0.1176
C_{d_0} [-]	0.045	0.0108	0.0315
C_{d_0} [-]	0.055	0.0068	-0.0280
C_{y_0} [-]	0.045	-0.0010	0.0013
C_{y_0} [-]	0.055	0.0014	-0.0014
S $[m^2]$	0.273	0.2018	0.0358
S $[m^2]$	0.247	0.2186	-0.0404
m $[kg]$	1.3335	0.3672	7.7529
m $[kg]$	1.2065	0.4475	2.7402
I_{xx} $[kg.m^2]$	0.0715	-0.0002	-0.0002
I_{xx} $[kg.m^2]$	0.0585	2.93504E-05	-0.0008
I_{yy} $[kg.m^2]$	0.0099	0.0052	0.0058
I_{yy} $[kg.m^2]$	0.0081	-0.0051	-0.0068
I_{zz} $[kg.m^2]$	0.07282	-0.0006	0.0007
I_{zz} $[kg.m^2]$	0.05958	0.0006	-0.0006
CG_x $[m]$	+0.01	3.3991	48.8413
CG_x $[m]$	-0.01	-	4.7458

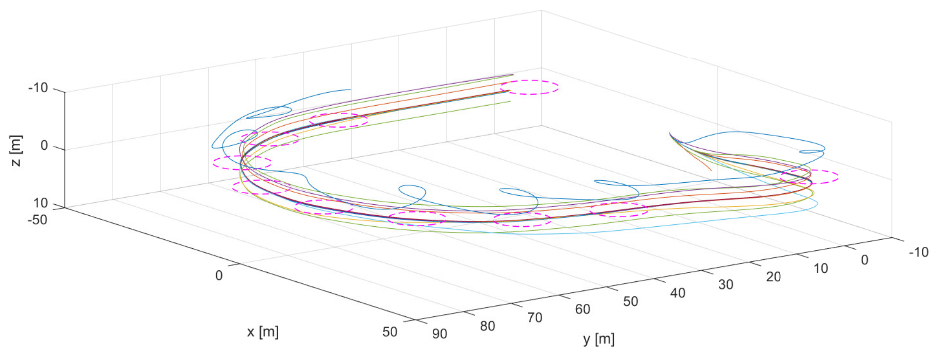


Fig. 23. Overview of all NMPC trajectories. Different coloured lines represent simulations with different parameters changed.

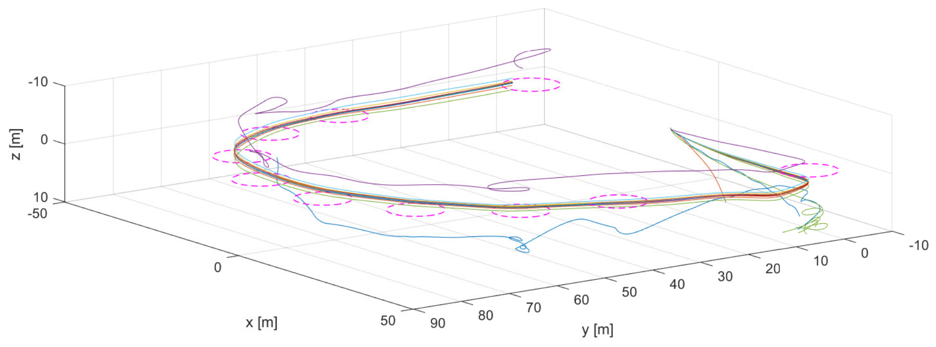


Fig. 24. Overview of all NN trajectories. Different coloured lines represent simulations with different parameters changed.

A first look at Figure 23 and Figure 24 shows that out of the simulations that succeeded, the vertical variation in trajectories for the NN is much smaller than for the NMPC compared to their respective nominal trajectories. This is reflected in Table IX where it can be seen that the e_V for NN is either similar to that of the NMPC or substantially smaller in the case of C_{L_α} and S. However, although the NMPC shows more impact on performance when changing parameters, it succeeds in having the least failures overall.

Both controllers have similar low sensitivity to changes in C_{d_0}, C_{y_0} and the moments of inertia. However, some interesting results show up when looking at C_{L_α}, S and m. The NMPC is substantially more sensitive to C_{L_α} and S which directly affect the aerodynamic lift force but the NN fails the tests where the mass is changed. Finally, both the NMPC and NN are very sensitive to changes in the centre of gravity due to both centre of gravity changes causing failures.

VIII. Discussion

The current formulation of the NMPC provides a simple solution to the acceleration controller. However, the NMPC used to generate the training data used an integration timestep equal to the sample time of 0.1 seconds. This had as consequence that the main limits in performance for the NMPC were caused by solver crashes due to instability in the prediciton model. This was confirmed by introducing sub-stepping integration timesteps of 0.01 seconds which stopped solver failures and increased performance in all experiments. This Sub-stepping NMPC should be used in future work to generate the training data.

Considering the flight characteristics of the NMPC, if these need to be changed, an updated formulation of the cost function would be needed. In the rectangular trajectory experiment, the drone pitches up substantially and slows down to make turns. Enforcing a target velocity in the cost function could possibly push the NMPC to perform a smoother turn.

Concerning the NN, behaviours seen in the NMPC such as the pitch up to slow down for turns and the ability to fly upside down are transferred successfully from the NMPC to the NN. However, the NN shows some performance degradation with respect to the NMPC. The NN manages to follow general velocity requirements but does not settle for the actual requested value. This is reflected in the hover test where once it slows back down after having reached the target velocity, it does not manage to slow down to 0 m/s completely, or in the maximum velocity test where it does not settle to the target 8 m/s. The more complex circular and rectangular trajectory tests show the ability of the NN to perform turns which is a task not included in the training data. These trajectories were achieved up to about half the maximum velocity included in the training data (11 m/s). Increasing the maximum velocity these trajectories can be performed at could be done by increasing the velocity range included in the training data and/or adding more specific training data that includes turns.

From the sensitivity analysis, the NN shows reduced sensitivity to changes in parameters that affect lift generation (C_{L_α} and S) over the NMPC. It is difficult to say, however, if this reduced sensitivity will remain once the performance of the NMPC and the training of the NN have been improved.

Looking at both controllers as a whole, the trade-off the NN proposes is very attractive. Velocity and acceleration errors are relatively worse but it solves the main issue of the NMPC which is its difficulty to run in real-time. From the inference speed test results shown in Table III, the NN can comfortably run onboard the drone.

One of the main problems when talking about validating the controllers on the real drone is that the NMPC has difficulties running in real-time. This is why a NN was trained to mimic the behaviour of the NMPC. This makes identifying reasons for issues in the validation results very difficult. Poor performance of the NN controller on the real drone could result from issues at any stage of the design process. This issue is amplified by the time it takes to perform a design iteration.

The training data generation took about 80 hours and the training of the neural network around 10 minutes. Any change to the model parameters requires a re-generation of all the training data and retraining of the neural network. The main limiting factor for the amount of time it takes to perform a design iteration is therefore the generation of the training data. An idea to get around this issue is to set up a reinforcement learning framework to continue the training of the neural network after it has been trained on the NMPC training data. This is called pretraining the network and has been shown to significantly improve reinforcement learning performance for an autonomous driving application in [21].

The validation process may also require improvements to the underlying dynamics model. The sensitivity analysis gives an initial idea of the parameters that will require the most attention. Mainly the position of the centre of gravity, the mass m, the wing surface S and the lift coefficient slope C_{L_α}. The centre of gravity location is the most sensitive out of these mentioned parameters. This is most likely a consequence of the NMPC being a model-based controller. Changing the position of the center of gravity changes the moment arm lengths of all the applied forces creating a substantial mismatch between the controller internal model and the simulation model.

For future work, before testing the NN in real flight, it is recommended to perform an additional design iteration using the Sub-stepping NMPC to generate the training data. Additionally, it would be beneficial to expand the velocity ranges included in the training data and add scenarios which include turns. With this newly trained NN, real flight tests should be performed and compared to simulation to start the validation process of the controller.

IX. Conclusion

This paper demonstrates the ability to transfer complex acceleration control logic from an expert controller using Nonlinear Model Predictive Control to a neural network using a process called behavioural cloning. In addition, it provides insights on how to improve the framework and improve controller performance. In its current formulation, the NMPC that was used to generate the training data shows an ability to control this complex platform up to 19 m/s in forward horizontal flight and perform circular and rectangular trajectories up to 10 and 8 m/s respectively. These performances were mainly limited by an issue in the prediction model where a large integration timestep was used which caused model instability. The use of sub-steps to reduce the integration timesteps fixed this issue and extended the performance of the NMPC.

Behavioural Cloning of the NMPC acceleration controller using a feedforward neural network has shown successful transfer of general flight behaviours but does show performance degradation with respect to the NMPC in all tests performed. An expansion of training data and variety of the training data is required to improve the learning transfer.

Although the NMPC controller performance has probably not reached its limits, if new formulations require substantially more computational time, a practical limit may be reached with regards to the time it takes to generate training data. With the current training data taking around 80 hours to generate, design iterations are slow and are substantial decisions to make. A proposal to mitigate this issue is to use the neural network trained with the NMPC data as a pre-trained network for a reinforcement learning framework.

References

[1] A.S. Saeed et al. “A survey of hybrid Unmanned Aerial Vehicles”. In: *Progress in Aerospace Sciences* 98 (2018), pp. 91–105. ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2018.03.007. URL: https://www.sciencedirect.com/science/article/pii/S0376042117302233.

[2] G.J.J. Ducard and M. Allenspach. “Review of designs and flight control techniques of hybrid and convertible VTOL UAVs”. In: *Aerospace Science and Technology* 118 (2021), p. 107035. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2021.107035. URL: https://www.sciencedirect.com/science/article/pii/S1270963821005459.

[3] G.H.L.H. Lovell-Prescod, Z. Ma, and E.J.J. Smeur. “Attitude Control of a Tilt-rotor Tailsitter Micro Air Vehicle Using Incremental Control”. In: *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2023, pp. 842–849.

[4] L. Bauersfeld and G. Ducard. “Fused-PID Control for Tilt-Rotor VTOL Aircraft”. In: (2020), pp. 703–708. DOI: 10.1109/MED48518.2020.9183031.

[5] J. Willis, J. Johnson, and R.W. Beard. “State-Dependent LQR Control for a Tilt-Rotor UAV”. In: (2020), pp. 4175–4181. DOI: 10.23919/ACC45564.2020.9147931.

[6] E.J.J. Smeur, M. Bronz, and G.C.H.E. de Croon. “Incremental Control and Guidance of Hybrid Aircraft Applied to a Tailsitter Unmanned Air Vehicle”. English. In: *Journal of Guidance, Control, and Dynamics: devoted to the technology of dynamics and control* 43.2 (2020), pp. 274–287. ISSN: 0731-5090. DOI: 10.2514/1.G004520.

[7] E. Tal and S. Karaman. “Global Trajectory-tracking Control for a Tailsitter Flying Wing in Agile Uncoordinated Flight”. In: *AIAA AVIATION 2021 FORUM* (Aug. 2021). DOI: 10.2514/6.2021-3214.

[8] L. Grüne and J. Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer London, 2011.

[9] M. Wang et al. “A Lightweight Control Method for Fast and Agile Quadrotor Using NMPC-Imitation Learning”. In: *Proceedings of 2022 International Conference on Autonomous Unmanned Systems (ICAUS 2022)*. Ed. by W. Fu, M. Gu, and Y. Niu. Singapore: Springer Nature Singapore, 2023, pp. 2940–2953. ISBN: 978-981-99-0479-2.

[10] L. Mero et al. “A Survey on Imitation Learning Techniques for End-to-End Autonomous Vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 23 (Sept. 2022), pp. 1–20. DOI: 10.1109/TITS.2022.3144867.

[11] L. Ribeiro Lustosa, F. Defaÿ, and J. Moschetta. “Global Singularity-Free Aerodynamic Model for Algorithmic Flight Control of Tail Sitters”. In: *Journal of Guidance, Control, and Dynamics* 42 (Dec. 2018), pp. 1–14. DOI: 10.2514/1.G003374.

[12] B. Volker. “Chapter 2 - Propellers”. In: *Practical Ship Hydrodynamics (Second Edition)*. Ed. by B. Volker. Second Edition. Oxford: Butterworth-Heinemann, 2012, pp. 41–72. ISBN: 978-0-08-097150-6. DOI: https://doi.org/10.1016/B978-0-08-097150-6.10002-8. URL: https://www.sciencedirect.com/science/article/pii/B9780080971506100028.

[13] M. de Freitas Virgilio Pereira, I.V. Kolmanovsky, and C.E.S. Cesnik. “Nonlinear Model Predictive Control with aggregated constraints”. In: *Automatica* 146 (2022), p. 110649. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2022.110649. URL: https://www.sciencedirect.com/science/article/pii/S0005109822005131.

[14] R. Cagienard et al. “Move blocking strategies in receding horizon control”. In: *Journal of Process Control* 17.6 (2007), pp. 563–570. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2007.01.001. URL: https://www.sciencedirect.com/science/article/pii/S0959152407000030.

[15] T. Szandała. “Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks”. In: *Bio-inspired Neurocomputing*. Ed. by A.K. Bhoi et al. Singapore: Springer Singapore, 2021, pp. 203–224. ISBN: 978-981-15-5495-7. DOI: 10.1007/978-981-15-5495-7_11. URL: https://doi.org/10.1007/978-981-15-5495-7_11.

[16] G. Civelekoglu et al. “Prediction of Bromate Formation Using Multi-Linear Regression and Artificial Neural Networks”. In: *Ozone: Science & Engineering* 29.5 (2007), pp. 353–362. DOI: 10.1080/01919510701549327. URL: https://doi.org/10.1080/01919510701549327.

[17] A. Saberian et al. “Modelling and Prediction of Photovoltaic Power Output Using Artificial Neural Networks”. In: *International Journal of Photoenergy* 2014 (Apr. 2014), pp. 1–10. DOI: 10.1155/2014/469701.

[18] K.C. Hikmet and A. Murat. “Generalized regression neural network in modelling river sediment yield”. In: *Advances in Engineering Software* 37.2 (2006), pp. 63–68. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2005.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0965997805000888.

[19] D.P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[20] S. Raschka. *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. 2020. arXiv: 1811.12808 [cs.LG].

[21] T. Wang and D.E. Chang. “Improved Reinforcement Learning through Imitation Learning Pretraining Towards Image-based Autonomous Driving”. In: *2019 19th International Conference on Control, Automation and Systems (ICCAS)*. 2019, pp. 1306–1310. DOI: 10.23919/ICCAS47443.2019.8971737.

3 Literature Study

This section summarises the initial assessment of the literature concerning the different control methods applied to hybrid MAVs.

3.1. Literature Study Research Questions

The literature study will aim to answer the following research questions.

1. What are the different outer-loop control methods for hybrid MAVs available for following kinematic references generated by the trajectory tracking controller?
    (a) What are the benefits of using this method?
    (b) What are the limitations of this method?
    (c) Is the method feasible to implement in the time frame of the thesis?
    (d) Does the method require data that cannot be acquired in the framework of this thesis?
2. Tail-sitters have a large flight envelope which makes aerodynamic modelling challenging. What are the current methods employed for modelling these aerodynamics?
    (a) What are the benefits of using this model?
    (b) What are the limitations of the model?
3. What is a suitable method of modelling the thrust generated by the propellers of the drone in the context of this project?
    (a) What are the main methods of modelling propeller thrust?
    (b) It is known that wind speed and direction affect propeller performance, what are the current methods of modelling these changes?
4. What are currently the different trajectory tracking methods available for following a three-dimensional trajectory for MAVs?
    (a) What are the benefits of using this method?
    (b) What are the limitations of this method?
    (c) Is the method feasible to implement in the time frame of the thesis?
    (d) Does the method require data that cannot be acquired in the framework of this thesis?

3.2. State of the Field

This chapter will discuss the state of the field for the topics mentioned in the literature study part of the research questions. The order in which topics are discussed in this chapter will be the same as the order they appear in the research questions.

3.2.1. Outer-loop Control of Hybrid MAVs

This section will explore different outer-loop control methods that can be applied for control hybrid of MAVs and how they were implemented in the literature.

PID Control

PID stands for Proportional-Integral-Derivative and is a controller that translates errors in a system to control inputs via a combination of proportional, integral and derivative terms. Its simplest form is depicted in Equation 3.1.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{3.1}$$

The error $e(t)$ is defined as $e(t) = r(t) - y(t)$ where $r(t)$ is the reference and $y(t)$ the current state of the system. $K_p/K_i/K_d$ are gains that can be tuned to improve the performance of the system and $u(t)$ is the control output that the equation calculates and that should be applied to the system to reduce the error. PID control has the advantage of not requiring any modelling of the system to function. However, its original form in Equation 3.1 performs very poorly for highly non-linear systems. Countermeasures can be taken such as gain scheduling but that is a time-intensive task.

Chiappinelli et al. [3] implemented a PI controller that translated the forward velocity error into a required forward force which is the control input. In [4], the authors proposed a fused-PID velocity control system for their tilt-rotor VTOL (Vertical Take-Off and Landing) drone. The use of this method highlights one of the problems with PID control which is the difficulty of dealing with nonlinear systems. Indeed in [4], two different PID controllers are developed to deal with both hover and horizontal flight separately. The transition between flight conditions is then dealt with by fusing the control outputs of both controllers adding more importance to one or the other depending on velocity.

LQR Control

LQR or Linear Quadratic Regulator control is a form of optimal state feedback control that calculates the optimal gains required to have a stable and high-performance closed-loop system. Different forms of LQR are possible, for simplicity the formulation that will be detailed here concerns a continuous time and infinite horizon formulation of LQR. First a time-invariant state-space system in the form shown in Equation 3.2 is needed.

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}, \qquad \boldsymbol{x}(0) = \boldsymbol{x_0} \tag{3.2}$$

Then as the objective of the feedback control is to calculate the optimal control input, $\boldsymbol{u}$ is calculated using Equation 3.3. This clearly shows that LQR is a state feedback method as u is a linear combination of the states x of the system.

$$\boldsymbol{u} = K\boldsymbol{x} \tag{3.3}$$

To start calculating K, a cost function is defined in Equation 3.4.

$$J_{cost} = \int_0^\infty (\boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{u}^T R \boldsymbol{u}) dt \tag{3.4}$$

Where Q and R are square positive matrices of size of the number of states and the number of control inputs respectively. The optimal control feedback gain matrix K^* can then be defined as shown in Equation 3.5.

$$K^* = -R^{-1} B^T P \tag{3.5}$$

Where P is a positive matrix and can be found by solving the Ricatti equation given by Equation 3.6.

$$Q + A^T P + PA - PBR^{-1} B^T P = 0 \tag{3.6}$$

LQR has successfully been implemented as mentioned in [5] as a low-level velocity controller for the AR.Drone quadcopter combined with a nonlinear inverse kinematic controller that provides the velocity references for the LQR controller to follow.

In [6], a state-dependent LQR trajectory controller was developed as outer-loop control for a tilt-rotor drone. Limitations from this method involve not being able to guarantee the stability of the controller as the LQR controller is state-dependent and concerning the flight performance, only moderately aggressive trajectories could be achieved.

The main problem with LQR is that it is model-dependent control and thus the quality of the model affects the effectiveness of the control severely if there are model inaccuracies or external disturbances. Additionally, LQR can only guarantee stability for linear systems. This means that if it is used for nonlinear systems and is made state-dependent, the stability of the controller cannot be guaranteed just as mentioned in [6].

INDI Control

Incremental Nonlinear Dynamic Inversion (INDI) is a modification of Nonlinear Dynamic Inversion (NDI) that reduces the dependency of the control method on the model and knowledge of the system dynamics. NDI creates a linear mapping of inputs to outputs from the nonlinear dynamics that can be used by a linear control law [7, 8, 9]. After being applied to multiple aircraft for flight control [10, 11, 12, 13, 14], NDI was deemed not robust enough as performance of exact dynamic inversion suffers severely from sensor noise and modelling uncertainty [15]. The INDI form adds incremental changes to the inputs based on linear and angular acceleration measurements. This provides better robustness over NDI as external disturbances and modelling uncertainties are mitigated through the inertial measurements.

To understand the basic principles of INDI, the derivation in [16] for a general nonlinear system will be detailed. To start, a general nonlinear system is defined in Equation 3.7.

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \tag{3.7}$$

Where $\boldsymbol{x}$ is the state vector and $\boldsymbol{u}$ is the control input vector. A Taylor expansion can then be performed around $[\boldsymbol{x_0}, \boldsymbol{u_0}]$ where a 0 subscript indicates the latest available information about the states and the control inputs and no subscript for $\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}$ indicates the values in the next time-step. The expansion is presented in Equation 3.8.

$$\begin{aligned} \dot{\boldsymbol{x}} &\approx f(\boldsymbol{x_0}, \boldsymbol{u_0}) + \frac{\partial f(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}}\bigg|_{\boldsymbol{x}=\boldsymbol{x_0},\boldsymbol{u}=\boldsymbol{u_0}} (\boldsymbol{x} - \boldsymbol{x_0}) + \frac{\partial f(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}}\bigg|_{\boldsymbol{x}=\boldsymbol{x_0},\boldsymbol{u}=\boldsymbol{u_0}} (\boldsymbol{u} - \boldsymbol{u_0}) + h.o.t. \\ \dot{\boldsymbol{x}} &\approx \dot{\boldsymbol{x}}_0 + F(\boldsymbol{x_0}, \boldsymbol{u_0})(\boldsymbol{x} - \boldsymbol{x_0}) + G(\boldsymbol{x_0}, \boldsymbol{u_0})(\boldsymbol{u} - \boldsymbol{u_0}) \end{aligned} \tag{3.8}$$

The notable simplifications in Equation 3.8 are the removal of the higher order terms ($h.o.t.$) and using Equation 3.7, $f(\boldsymbol{x_0}, \boldsymbol{u_0})$ can be rewritten as $\dot{\boldsymbol{x}}_\boldsymbol{0}$. To further simplify Equation 3.8, some additional assumptions can be taken. Based on the assumption that the control system is run at a small sampling time and the assumption of instantaneous control effectors [16] this means that it is assumed that $\boldsymbol{x} \approx \boldsymbol{x_0}$ and that $\boldsymbol{u} \neq \boldsymbol{u_0}$. This results in Equation 3.9.

$$\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}_\boldsymbol{0} + G(\boldsymbol{x_0}, \boldsymbol{u_0})(\boldsymbol{u} - \boldsymbol{u_0}) \tag{3.9}$$

Equation 3.9 can then be rearranged to obtain $\boldsymbol{u}$. Additionally, $\dot{\boldsymbol{x}}$ is replaced with $\boldsymbol{v}$ which is the virtual control input that has to be designed and is essentially the reference the system has to follow.

$$\boldsymbol{u} = \boldsymbol{u_0} + G^{-1}(\boldsymbol{x_0}, \boldsymbol{u_0})(\boldsymbol{v} - \dot{\boldsymbol{x}}_\boldsymbol{0}) \tag{3.10}$$

To the best knowledge of the author, there is no literature concerning the acceleration control of a tilt-rotor tail-sitter specifically. Therefore, mentions of INDI implementations will also note the drone configuration that was used in the paper. In [17], INDI was implemented for a fixed rotor tail-sitter where two INDI controllers were designed to control attitude and velocity using the general method described earlier which makes use of the control effectiveness matrix.

An alternative implementation of INDI is implemented in [18] that removes the need to locally linearise the dynamics and invert the control effectiveness matrix of the tail-sitter. This was done by deriving a differential flatness transform of the tail-sitter's nonlinear dynamics which included a simplified aerodynamics model [19]. This implementation showed accurate trajectory tracking for aggressive uncoordinated flight manoeuvres.

3.2.2. Aerodynamics and Thrust Modelling

Two key components for robust control are the accurate modelling of the aerodynamics and thrust. This section will cover methods used in the literature and discuss their advantages and disadvantages.

Aerodynamics Modelling

Due to the large flight envelope tail-sitters encounter, it becomes challenging to model the aerodynamic forces of all flight phases. This section will look at the current methods employed to model the aerodynamics of tail-sitters and discuss their benefits and limitations.

Static Force Aerodynamic Model

In [17], a very simple approach to the aerodynamics model is taken. It is assumed that the flight path angle is always zero. This means that only gravity needs to be countered through the use of lift and thrust. A simple function shown in Equation 3.11 can then be used to estimate lift.

$$L(\theta) = -9.81\,\text{sin}(-\theta)m, \theta \in [-\frac{\pi}{2}, 0] \tag{3.11}$$

Where θ is pitch angle and m is the mass of the drone. As for thrust, it is assumed that it always compensates for drag which results in Equation 3.12.

$$T(\theta) = -9.81\,\text{cos}(\theta)m, \theta \in [-\frac{\pi}{2}, 0] \tag{3.12}$$

These equations provide a very simple model to implement for the aerodynamics of the system and in [17], were sufficient to perform horizontal flight for a fixed rotor tail-sitter. However, this method poses some limitations. The assumption that the flight path angle is zero means that the model is very inaccurate when the trajectory performed by the drone is not horizontal. Furthermore, the assumption that the thrust always counters the drag also leads to significant inaccuracies when the drone is accelerating. For the tests performed in [17], this model was good enough to perform horizontal flight. However, it is expected that such a simple model would not be sufficient for following a three-dimensional trajectory.

Singularity-Free Aerodynamic Model

This method was developed in [19] to provide a simple singularity-free aerodynamic model for control purposes of tail-sitters. It is important to note that the main focus of the model is not to provide the most accurate aerodynamic model but to propose a model that is accurate enough whilst being easy to work with. The general proposed method is shown in Equation 3.13 where ρ is air density, S the wing surface area, V_∞ the free-stream velocity, $\vec{v}_B$ the velocity vector in the body reference frame and $\Phi^{(fv)}$ the aerodynamics coefficients matrix created with the model where $\Phi^{(fv)} \in \mathbb{R}^{3\times3}$.

$$\vec{F}_{B_{aero}} = \frac{1}{2}\rho S V_\infty \Phi^{(fv)} \vec{v}_B \tag{3.13}$$

For a xz-symmetrical aircraft, $\Phi^{(fv)}$ can be defined by Equation 3.14.

$$\Phi^{fv} = \begin{bmatrix} \Phi_{11} & 0 & \Phi_{13} \\ 0 & \Phi_{22} & 0 \\ \Phi_{31} & 0 & \Phi_{33} \end{bmatrix}, \Phi_{13} = \Phi_{31} \tag{3.14}$$

Using thin-airfoil theory, additional coefficients can be removed and the remaining coefficients can be estimated without the use of large amounts of wind tunnel test data. The effective matrix $\Phi^{(fv)}$ that can be used is shown in Equation 3.15 where the only coefficient that is needed is C_{d_0} which is the minimum drag coefficient. C_{y_0} is not needed as it can be assumed that lateral aerodynamic forces are minimal compared to longitudinal forces for aircraft that have no vertical control surfaces such as the tilt-rotor tailsitter.

$$\Phi^{fv} = \begin{bmatrix} C_{d_0} & 0 & 0 \\ 0 & C_{y_0} & 0 \\ 0 & 0 & 2\pi + C_{d_0} \end{bmatrix} \tag{3.15}$$

The resulting model derived in [19] provides a robust method of calculating longitudinal aerodynamic forces and while the modelling accuracy is lower than other methods, it has already been proven successful by its use in [18] where a tail-sitter performed agile and aggressive manoeuvres which included knife-edge flight and transition from knife-edge flight to horizontal flight.

Parameter Learning Aerodynamics Model
Another option to model the aerodynamic forces is through parameter learning of a model. This has been done in [20] where parametrised functions are used to describe the lift and drag aerodynamic forces. Then through gathering data from onboard sensors, a parameter learning scheme can be applied to estimate the best value for these coefficients. Other learning methods were considered, however, parameter learning was chosen for this application thanks to its low computational and memory requirements compared to the other methods [21].

The model defines the aerodynamic forces in the body reference frame $\vec{F}_{B_{aero}}$ as is shown in Equation 3.16 where F_{lift} and F_{drag} are the parametrised functions for lift and drag respectively. Both are functions of angle of attack α, free-stream velocity V_∞ and what is called the average propeller force f_a which is defined as $f_a = (f_l + f_r)/2$ where f_l and f_r are the left and right propeller forces. It is important to note that the presence of the average propeller force in the model does not mean this method includes thrust force, it means the aerodynamic forces induced by the thrust are taken into account.

$$\vec{F}_{B_{aero}} = \begin{bmatrix} -F_{lift}(\alpha, V_\infty, f_a) & 0 & F_{drag}(\alpha, V_\infty, f_a) \end{bmatrix}^T \tag{3.16}$$

The parameterised functions F_{lift} and F_{drag} are defined in Equation 3.17 where the coefficients that need to be learned are denoted $k_{...}$.

$$\begin{aligned} F_{lift}(\alpha, V_\infty, f_a) &= (k_{l1}\sin(\alpha)\cos(\alpha)^2 + k_{l2}\sin(\alpha)^3)V_\infty^2 + k_{l3}f_a, \\ F_{drag}(\alpha, V_\infty, f_a) &= (k_{d1}\sin(\alpha)^2\cos(\alpha) + k_{d2}\cos(\alpha))V_\infty^2 + k_{d3}f_a \end{aligned} \tag{3.17}$$

The implementation of this model in [20] where a fixed rotor tail-sitter was successfully flown and showed good tracking performance supports the potential use of this model for the controller of the drone. This is further supported by the fact that wind tunnel test data is not required as flight data from onboard sensors can be used to train the model. Implementation of this model would however require the addition of a term related to the angle of the propellers in the parameterised functions as it is expected that this angle significantly affects the aerodynamic forces applied to the drone.

Thrust Modelling
In the context of the tilt-rotor tailsitter drone, accurate modelling of the available thrust force is important for effective control of the system as this is the sole control method of the system. For this reason, this section will detail different thrust modelling methods that may be used.

In [18], a very simple model for thrust was used and is shown in Equation 3.18 where T is thrust, ω the propeller angular velocity and c_T the thrust coefficient which was obtained by performing a force balance bench test.

$$T = c_T\omega^2 \tag{3.18}$$

The simplicity of this model implies that a lot of aerodynamic effects are being ignored and thus that its accuracy is limited. However, this model was successfully implemented in [18] where a fixed rotor tail-sitter performed agile uncoordinated flight such as knife's edge flight which implies that such a simple model might be sufficient for this research. In the case that this model is not sufficient for the purposes of this research, additional models are proposed.

Separate Thrust and Propeller Aerodynamics Modelling
Thrust coefficients are typically calculated based on bench tests where the flow of air is axial to the propeller rotation axis. Therefore, the accuracy of these coefficients diminishes when the flow becomes non-axial as the non-axial velocity components generate additional aerodynamic forces that are not taken into account in these coefficients. The method proposed in [22] uses just like in [18], a simple thrust model but adds corrective forces when the airflow relative to the propellers becomes non-axial. The thrust model used is shown in Equation 3.19 where T is the thrust force, c_T the thrust coefficient, ρ the air density, ω the propeller angular velocity and D the propeller diameter.

$$T = c_T \rho \omega^2 D^4 \tag{3.19}$$

In [22], the thrust coefficient c_T varies based on the advance ratio J which is defined in Equation 3.20 where V_N is the normal velocity component of the flow with respect to the propeller disc. This means that c_T is a function of J so $c_T(J)$. Finding the coefficients c_T as a function of J is done in [22] by using blade element momentum theory and in particular using the PROPID code [23].

$$J = \frac{V_N}{\omega D} \tag{3.20}$$

Now that the thrust is modelled for axial flow, the method proposed to correct for non-axial flow is presented Equation 3.21 where Z_{fwd} is the corrective force in the same axis as the propeller rotational axis, σ is what is called the propeller solidity which is the ratio of blade area over disk area, $\bar{q}$ is dynamic pressure, A_{disc} is the propeller disc surface area, a is the airfoil lift-curve slope which is set to 2π, C_d the drag coefficient, α_{prop} the angle of the flow with respect to the propeller hub and $\bar{C}_l$ is the averaged lift coefficient which is defined in Equation 3.22

$$Z_{fwd} = -\frac{\sigma \bar{q} A_{disc}}{2}(\bar{C}_l + \frac{aJ}{2\pi}\ln(1 + (\frac{\pi}{J}^2)) + \frac{\pi}{J}C_d)\alpha_{prop} \tag{3.21}$$

$$\bar{C}_l = (\frac{3J}{2\pi})(\frac{2}{\sigma \bar{q} A_{disc}}\frac{J}{\pi}T + C_d) \tag{3.22}$$

In hover and near-hover conditions, Equation 3.21 cannot be used which is why when the drone transitions to hovering flight, Equation 3.21 is progressively switched to Equation 3.23 where K_{hvr} is an empirical coefficient that corrects for reduced flow turning efficiency and is equal to 0.8, $V_{\infty T}$ is the freestream velocity component in the disk plane which can be calculated with $V_{\infty T} = V_\infty \sin \alpha_{prop}$ and w_0 is the induced velocity caused by the propeller during hovering and is defined in Equation 3.24.

$$Z_{hvr} = -K_{hvr}(\rho w_0 A_{disc})V_{\infty T} \tag{3.23}$$

$$w_0 = \sqrt{\frac{T}{2\rho A_{disc}}} \tag{3.24}$$

The main limitation of this method is acquiring the data required to map the thrust coefficient as a function of the advance ratio. A solution is proposed by using the PROPID code but it is currently difficult to estimate how much time it would take or if it would be precise enough. If it proves too difficult to use the PROPID code, another method will now be proposed that is physics-based.

Physics Based Thrust Model
To avoid extensive testing to accurately model the thrust of the propellers for UAVs, the authors of [24] propose a physics-based model to accurately predict the thrust generated by the propellers. The model was compared to experimental test results and only had an error of 4.7% for the thrust. To be able to calculate the thrust using this model, a few values need to be known with respect to the radial location of the propeller blades and these are shown in Table 3.1.

Table 3.1: Required Propeller Data

Name	Symbol	Function of
Chord	c	Radial Position (r)
Chord line Pitch Angle	θ	Radial Position (r)
Zero-Lift Angle-of-Attack	α_0	Radial Position (r)
2D Lift Curve Slope	a	Radial Position (r)
2D Drag Coefficient	C_d	Radial Position (r)

It is important to note that the variables in Table 3.1 are dependent on the radial position of the propeller but this will not be shown in further equations. To calculate the thrust, Equation 3.25 needs to be numerically integrated starting at the radius of the propeller hub r_h and ending at the propeller tip r_p. The variables not mentioned in Table 3.1 are air density ρ, the number of blades N, the propeller angular velocity ω and the inflow angle ϕ_{in}. The inflow angle ϕ_{in} is defined as $\tan\phi_{in} = (V_x + V_i)/(\omega r)$ where V_x is the velocity component of the freestream normal to the propeller disc plane and V_i is the induced velocity.

$$T = \frac{1}{2}\rho N\omega^2 \sum_{r=r_h}^{r_p} (\frac{r^2}{\cos(\phi_{in})} c(a(\theta - \alpha_0 - \phi_{in}) - C_d \tan(\phi_{in}))\Delta r \tag{3.25}$$

All variables in Equation 3.25 should be known already except for ϕ_{in}. ϕ_{in} is solved for each section of the propeller radius by solving the nonlinear equation presented in Equation 3.26 and solutions for ϕ_{in} should respect the following boundaries $0 < \phi_{in} < \pi/2$.

$$\omega Nc(a(\theta - \alpha_0 - \phi_{in}) - C_d \tan\phi_{in}) = 8\pi \sin\phi_{in}(\omega r \tan\phi_{in} - V_x) \tag{3.26}$$

Whilst this method proposes a way to estimate generated thrust without extensive experiments, it still requires measurement and estimation of the propeller characteristics which if this information is not available from the manufacturer would take a substantial amount of time to obtain.

3.2.3. Trajectory Tracking

Trajectory tracking involves ensuring that the system that is being controlled is properly following a designed trajectory. This means that methods involved in trajectory tracking at least cover the kinematics of the system but are not limited to this and can additionally take into account the dynamics of the system. For this reason, purely kinematic trajectory tracking methods will be covered first (Waypoint Tracking and L_1 Guidance) then, methods involving both kinematics and dynamics will be covered (Differential Flatness and Model Predictive Control).

Waypoint Tracking

Waypoint tracking [25] is probably the simplest form of trajectory tracking that there is and is a good starting point for this section. It is considered simple as it does not take into account the dynamics of the vehicle and typically requires little information to function.

Vector Field Guidance

As the name implies, vector field guidance generates a field of vectors that typically depend on vehicle position. In [25] for example, each position in space gives a bearing value that the vehicle should follow and in [26], each position in space outputs the desired velocity vector to follow a loitering pattern. As there is no standard formulation for generating a field, the formulation from [25] will be presented in Equation 3.27. The definitions of the symbols in Equation 3.27 can be found in Table 3.2.

$$\begin{aligned} \delta &= \sqrt[K_R]{|D_{CT} K_C(\varphi_T - \varphi_R)|} * sign(D_{CT}) \\ \gamma &= min(1, D_T) \\ \varphi_d &= \varphi_T + \delta\gamma \end{aligned} \tag{3.27}$$

Table 3.2: Definition of symbols in Equation 3.27 [25]

Symbol	Definition
D_{CT}	Cross track error
D_T	Distance from target
φ_d	Desired bearing
φ_T	Bearing from the UAV to the destination waypoint
φ_R	Bearing from the previous waypoint to the destination waypoint
K_D	Path reaching gain
K_C	Cross track error gain
γ	Waypoint bearing gain

From Equation 3.27, waypoint following can be achieved and generates a vector field that is shown in Figure 3.1 [25] which starts on the left and ends on the right waypoint.

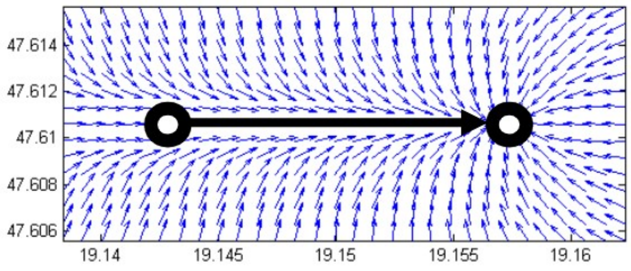


Figure 3.1: Waypoint Vector Field [25]

Alternative trajectories can be generated such as loitering around a point with a certain radius. To achieve this in [25], the equation for φ_d in Equation 3.27 was replaced by Equation 3.28.

$$\varphi_d = \varphi_T + \frac{\pi}{2} + min(|D_{CT}K_C|, \frac{\pi}{2}) * sign(D_{CT}) \tag{3.28}$$

In [26], the loitering vector field is generated by default by the formulation. This means no modifications are needed to achieve loitering and waypoint navigation can be achieved by switching the loitering circle positions. This can be done as the generated vector field is globally convergent to the loitering pattern. An example of a loitering pattern vector field can be seen in Figure 3.2 [26].

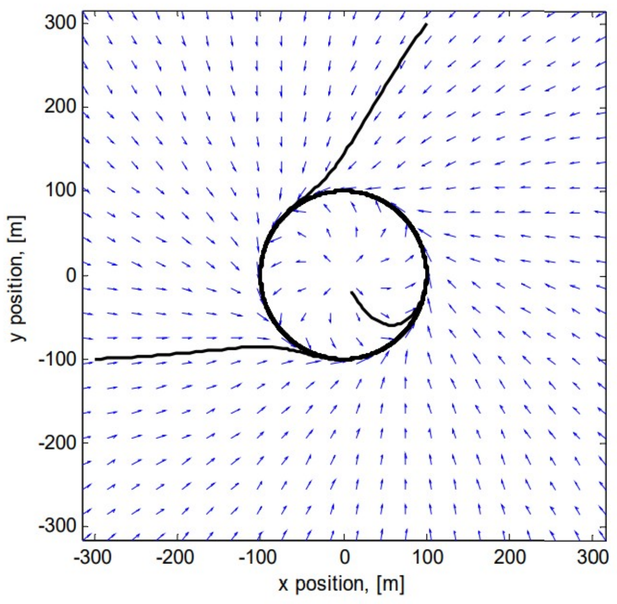


Figure 3.2: Loitering Vector Field [26]

L_1 Guidance

L_1 guidance [27] is a non-linear guidance method that generates a lateral acceleration command based on a reference point on the desired trajectory. The general logic is shown in Equation 3.29 and a visual representation of the logic is shown in Figure 3.3 [27].

$$a_{s_{cmd}} = 2\frac{V^2}{L_1}\sin\eta \tag{3.29}$$

Where $a_{s_{cmd}}$ is the commanded lateral acceleration, V the vehicle inertial velocity, L_1 the vector between the vehicle and the reference point and η is the angle between the vector L_1 and the velocity vector V.

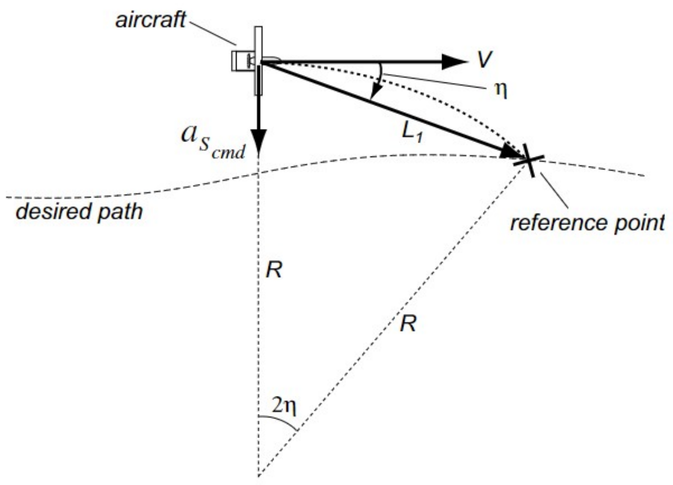


Figure 3.3: Guidance Logic Diagram [27]

This nonlinear method offers several advantages over traditional PID controllers. First, the use of η provides three advantages:

1. η provides a heading correction

2. For small angles, the controller behaves like a PD controller for cross-track error
3. As the L_1 vector essentially looks ahead of the desired trajectory, the method provides an anticipatory acceleration for turns in the reference

Second, the method does not require any knowledge of the dynamics of the system. Finally, the fact that the controller uses the inertial velocity V of the vehicle means that it can deal with external disturbances such as wind.

L_1 is a well-established guidance method and is implemented in multiple open source autopilots such as *PX4*[1] and *Ardupilot*[2]. Additionally, there are multiple papers addressing issues with the method and extending its capabilities.

For example, [28] proposes solutions for small, slow-flying fixed-wing UAVs to be able to handle small loiter radii and high winds. To handle small loiter radii, the L_1 distance was adaptively calculated based on the knowledge that to follow a circle of radius R, the L_1 distance should respect $L_1 \leq R$. As for high winds, when the wind velocity exceeds the airspeed of the UAV, the feasibility of following a desired L_1 trajectory depends on the bearing of the wind. This is why a bearing feasibility estimator was developed. The logic behind it is that when the desired trajectory is deemed feasible, the L_1 controller will function normally. But when the desired bearing is not feasible, the L_1 guidance will change its objective to what is called in [28] the ”Safety Objective” which is in the feasibility region.

In [29], several improvements are performed so that the guidance can be used in the real world. This involves adding additional logic for when the L_1 intercept cannot be defined as the distance from the vehicle to the trajectory is larger than the L_1 distance. Furthermore, to improve stability when flying on downwind trajectories, the length of L_1 was adaptively calculated based on its ground speed following Equation 3.30 where T^* is essentially a look-ahead time in seconds and V_g is the ground speed. It was concluded in [29] that for an adequate transient response, the value of T^* should be at least 3-4 times the roll response lag.

$$L_1 = T^* V_g \tag{3.30}$$

Some limitations of L_1 guidance will now be discussed. First, the method does not take into account the dynamics of the system it is controlling. This has two implications, if the system has slow jerk characteristics it could overshoot as the controller assumes that the commanded acceleration it provides is applied instantaneously. The second implication is that as the dynamics of the system are not taken into account when generating the trajectory, it is possible to generate trajectories that are impossible to follow properly.

Second, the method does not control the complete velocity vector of the vehicle, it only provides a perpendicular acceleration to the velocity vector. Furthermore, when velocity is zero, the control law in Equation 3.29 says that the lateral acceleration is zero. This means that a separate velocity controller needs to be implemented.

Finally, the method needs additional logic to deal with problems such as when the distance between the vehicle and the closest part of the trajectory is further than the length L_1.

Trajectory Tracking Using Differential Flatness

Differential flatness [30] is in itself not a control method but a characteristic of nonlinear systems that simplifies their control. A definition given by [31] will be detailed. A general nonlinear system shown in Equation 3.31 is differentially flat if there exists outputs called flat outputs of the form shown in Equation 3.32 such that $\boldsymbol{x}$ and $\boldsymbol{u}$ can be expressed as shown in Equation 3.33.

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}), \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{u} \in \mathbb{R}^m \tag{3.31}$$

[1]https://px4.io/

[2]https://ardupilot.org/

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}, \dot{\boldsymbol{u}}, ..., \boldsymbol{u}^{(p)}), \boldsymbol{y} \in \mathbb{R}^m, p \in \mathbb{Z}_{\geq 0} \tag{3.32}$$

$$\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{y}, \dot{\boldsymbol{y}}, ..., \boldsymbol{y}^{(q)}), \boldsymbol{u} = \boldsymbol{u}(\boldsymbol{y}, \dot{\boldsymbol{y}}, ..., \boldsymbol{y}^{(q)}), q \in \mathbb{Z}_{\geq 0} \tag{3.33}$$

The implications of such a system are very advantageous. In [18], the flat output is defined as (x, y, z, ϕ) which means that defining this flat output is the same as defining the desired trajectory. From this, a unique set of states and inputs can be calculated to achieve this trajectory. This enables more agile and aggressive manoeuvres to be performed. Most notably in [18], this method enabled the fixed rotor tail-sitter to perform a stable knife edge flight and transition from this knife edge flight to coordinated flight whilst also performing a 1.6g turn. It should be noted that the ability to perform such manoeuvres is not completely thanks to differential flatness but its combination with INDI for linear and angular acceleration control that can deal with external disturbances and model inaccuracies.

The ability to perform aggressive and agile flight is further supported by [32] where differential flatness and INDI were applied to a quadrotor that achieved an RMSE of 6.6cm for position tracking and reached accelerations of up to 2.1g.

Using differential flatness for control does prove to have some limitations. For example, ensuring that the system is differentially flat cannot be guaranteed. This is because finding functions that relate the states and inputs to the flat outputs can be difficult or impossible (In which case the system cannot be differentially flat).

Additionally, looking at Equation 3.33, It should be noted that there are limitations on the type of trajectory that can be designed as calculating the states $\boldsymbol{x}$ and $\boldsymbol{u}$ requires the derivatives of y up to q. So a defined trajectory must be differentiable q times.

Model Predictive Control

An MPC is an advanced control method that typically makes use of a simplified model of the system to calculate the optimal control inputs the system needs to perform over a certain prediction horizon. Figure 3.4 depicts a simplified MPC scheme where the system consists of one input and one output. Starting at $t = k$, the MPC will calculate, through the use of an objective function, the optimal control input to minimise the error between the predicted output in orange and the reference trajectory in red. The output of the optimisation is the predicted control input in light blue. From this predicted control input, the vehicle will apply the control input of $k + 1$ to the real system then one sample time after the first optimisation it will perform the same process over again.

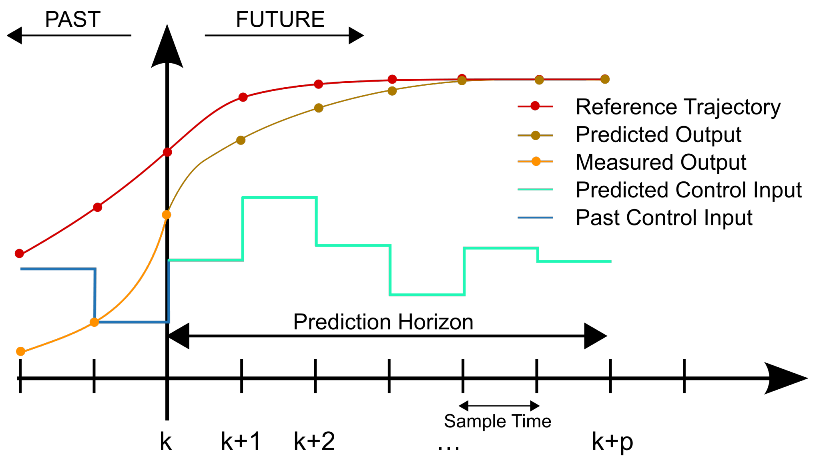


Figure 3.4: Single Input, Single Output MPC Example [33]

An additional concept that is not depicted in Figure 3.4 is the control horizon. The control horizon must always be equal to or smaller than the prediction horizon p. The control horizon dictates the number of time steps over which the control inputs may vary. Past the control horizon, the inputs must remain the same as the last input of the control horizon. This translates to an equality constraint in the formulation of the optimisation problem.

In [34], a version of Model Predictive Control called Successive Linearisation Model Predictive Control (SLMPC) was used to follow a reference trajectory for a quadrotor tail-sitter in hovering flight. Successive linearisation indicates that the model is linearised at each timestep to be able to perform linear optimisation even though the system is nonlinear. The implementation of SLMPC in [34] does show some weaknesses. First, the method described is not a general controller for all flight phases of the tail-sitter and would require extensions for transition and horizontal flight. Secondly, the wind disturbances are assumed to be known in the simulations that were performed which means that wind needs to be estimated during the flight in real-world conditions.

Another potential method of dealing with nonlinear systems is to make use of nonlinear model predictive control. With a proper model, this ensures that model inaccuracies are kept minimal. However, the nonlinear nature of the system means that a nonlinear solver must be used which implies that a globally optimal solution cannot be guaranteed and also significantly increases the computational load.

3.3. Research Gap & Conclusion

From the research performed in section 3.2 and to the best knowledge of the author, the autonomous guidance of a twin tilt-rotor tail-sitter has not yet been achieved. Implementations on similar platforms such as fixed-rotor tail-sitters have been successful and point towards the feasibility of achieving autonomous guidance for a tilt-rotor platform. As the novelty lies in achieving autonomous guidance, the body of literature explored in section 3.2 will serve as guidance for the best way to achieve that objective. The literature research explored the topics of hybrid MAV control, aerodynamics & thrust modelling and trajectory tracking. Each topic had multiple methods available which is why in this section, a method from each topic will be selected to be implemented. Trajectory tracking will be covered first, then the control method that will be combined with this and finally, the aerodynamics and thrust models that will be used in the control method will be chosen.

Trajectory Tracking

To the best knowledge of the author, none of the trajectory tracking methods detailed previously have been applied to the specific twin tilt-rotor tail-sitter platform. However, certain methods have been applied to very similar platforms such as differential flatness which was applied to a twin fixed-rotor tail-sitter and model predictive control which was applied to a quad fixed-rotor tail-sitter. The application of such methods to very similar platforms is a good argument for their potential use further on. However, each has a principal problem which leads to choosing another option.

Differential flatness has proven to be excellent for agile manoeuvres for a fixed-rotor tail-sitter. The main issue concerning the application of this method is time. When considering the time it has taken to simply implement INDI attitude stabilisation on the platform (A full thesis was dedicated to this), it is not realistic to assume the full implementation of this method could take a maximum of 8 months.

Concerning MPC, the two available options are SLMPC and nonlinear MPC. The first has been shown to perform well but would require model extensions for transition and horizontal flight which may take too much time. The second, as it is solved by a nonlinear solver would potentially encounter two problems. The computational power required could be too high for such a complex system or, the solved solutions are not optimal which could severely affect the performance of the system.

There now remains Waypoint Tracking and L_1 Guidance. Waypoint Tracking is essentially a PID controller which for a heavily nonlinear system would most likely require gain scheduling with a lot of tuning required for the system to perform correctly. For this reason, this method will not be chosen. This means that despite the practical adaptations the method needs to be implemented, L_1 Guidance will be used for trajectory tracking due to the simplicity and robustness of its control logic.

Outer-loop Control Method
With L_1 guidance providing a reference acceleration to follow, this reference now needs to be translated to effective control inputs that will achieve the required acceleration. From the research performed in subsection 3.2.1, three methods are available: PID, LQR and INDI.

PID control does not require any model information which for simple linear systems is very practical as the controller only needs to be tuned to the desired performance. Unfortunately, this simplicity does not carry over to highly nonlinear systems such as a tilt-rotor tail-sitter and thus would not be an effective method of translating the acceleration requirement to control inputs.

The performance of LQR control suffers mostly when there are model inaccuracies or the system is subject to external disturbances. For the purposes of a tilt-rotor tail-sitter, this method is not optimal. Firstly due to the limited accuracy of dynamics models for tail-sitters and secondly, outdoor flight requires robustness to external disturbances. This method will therefore not be used.

Finally, INDI is considered. Its successful implementation in two different fixed-rotor tail-sitters [18, 17] shows promising performance for controlling such a nonlinear system and by design can deal with external disturbances.

Aerodynamics and Thrust Models
For the aerodynamic and thrust models, as they are not the focus of this research, the simplest or most practical implementations will be used at first. If their performance is not sufficient to ensure the proper functioning of the system, the proposed alternatives will be implemented.

For the aerodynamics, this means that the singularity-free model will first be implemented as it requires no experimental or simulation data and as the name implies it is also singularity-free which is very advantageous for a tail-sitter MAV.

Finally, concerning the thrust model, the simplest form shown in Equation 3.18 will be used initially as it has successfully been used for the agile control of a fixed-rotor tail-sitter. In the potential scenario it is not sufficient, one of the two other alternatives may be implemented.

Conclusion
With methods from each topic selected, the control structure is complete. An overview of the control structure that will be developed is presented in Figure 3.5. Starting from the right is the inner loop control which consists in attitude stabilisation using INDI and was developed by Gervase Lovell-Prescod [1]. As can be seen, the inner loop takes as inputs a required thrust and attitude which is then translated to actual actuator commands in the inner loop. These inputs are provided by the INDI acceleration controller that will be developed. In combination with the 3D L_1 controller, the INDI acceleration controller forms the outer loop of the control structure and should allow the drone to follow a desired three-dimensional trajectory. In Appendix A, an explanation of the adapted L_1 kinematic controller is given and a partial derivation of the INDI acceleration controller is derived.

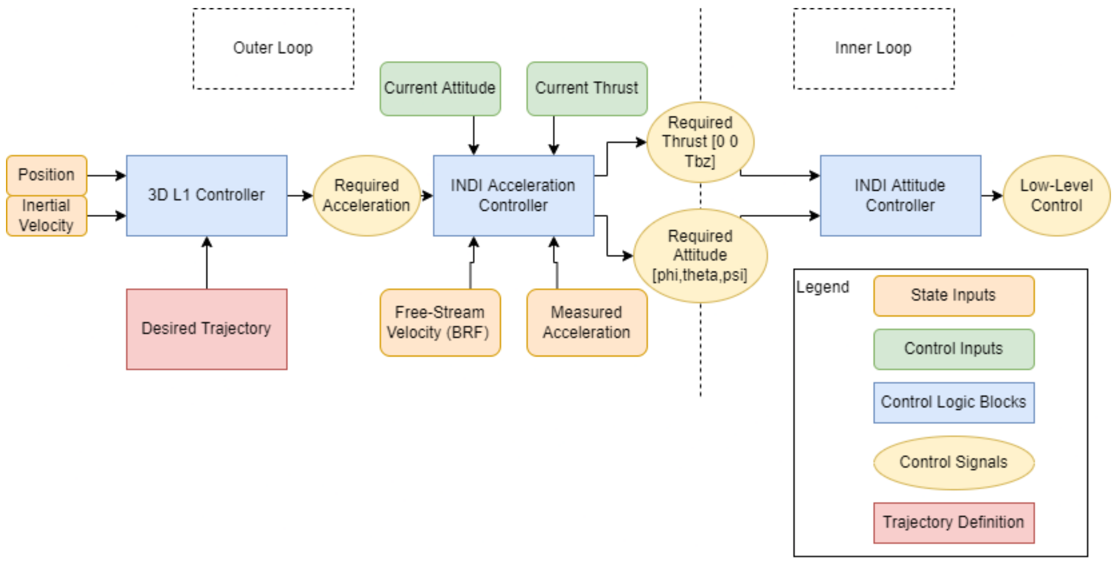


Figure 3.5: Control Structure Overview

4 Verification

This chapter is dedicated to verifying the simulation model used to simulate the behaviour of the tilt-rotor tailsitter MAV. This will be done in two steps. First, the calculation of the state derivatives will be verified by comparing manually calculated derivatives with the derivatives calculated by the developed function. Finally, some full simulation verifications will be performed where specific simulation scenarios will be performed and the outputs analysed to check that the expected behaviours and outputs are observed.

4.1. Model Verification

The states and control inputs of the model will be referred to multiple times in this chapter therefore to avoid repetition, the units for each symbol will be summarised here in the following tables. The units for the states are shown in Table 4.1, the units for the corresponding state derivatives are presented in Table 4.2 and finally, the units for the inputs of the model are shown in Table 4.3.

$$\begin{gathered}\vec{X} = \langle x, y, z, V_x, V_y, V_z, q_0, q_x, q_y, q_z, p, q, r\rangle \\ \vec{U} = \langle \omega_l, \omega_r, \alpha_l, \alpha_r\rangle\end{gathered} \tag{4.1}$$

Table 4.1

Symbols	x	y	z	V_x	V_y	V_z	q_0	q_x	q_y	q_z	p	q	r
Units	$[m]$			$[m/s]$			$[-]$				$[rad/s]$		

Table 4.2

Symbols	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{V}_x$	$\dot{V}_y$	$\dot{V}_z$	$\dot{q}_0$	$\dot{q}_x$	$\dot{q}_y$	$\dot{q}_z$	$\dot{p}$	$\dot{q}$	$\dot{r}$
Units	$[m/s]$			$[m/s^2]$			$[-]$				$[rad/s^2]$		

Table 4.3

Symbols	ω_l	ω_r	α_l	α_r
Units	$[rad/s]$		$[rad]$	

4.1.1. Manual Verification of State Derivatives

A simple first verification that can be done is to check that the function calculating the state derivatives is producing the right values. To do so, some simple states can be generated. The derivatives can then be calculated manually and then compared to the output of the state derivatives function. For each test, the inputs to the function will be described in an equation similar to Equation 4.1 except the symbols will be replaced with numerical values, then a table will detail the ”Expected” values calculated manually and the values produced by the ”Simulation” function. An error will also be calculated if there is any.

Test 1

This first test has the drone flying in the x-direction at 10 m/s with zero yaw, pitch and roll. This means it is completely flat and aligned with the velocity vector. Additionally, both propellers are spinning at 1000 rad/s and tilted upwards by 10 degrees. An overview of these values is shown in Equation 4.2.

$$\begin{aligned}\vec{X} &= \langle 0,0,0,10,0,0,1,0,0,0,0,0,0\rangle \\ \vec{U} &= \langle 1000, 1000, \tfrac{\pi}{18}, \tfrac{\pi}{18}\rangle\end{aligned} \tag{4.2}$$

The results of this test in Table 4.4 show no error between the expected and simulation values. Furthermore, the system behaves as expected, there is an acceleration in the x-direction $\dot{V}_x$ caused by the propellers and a reduction of gravity's acceleration in the z-direction $\dot{V}_z$ due to the tilting of the propeller by 10 degrees. The tilting of the propellers also generates a moment around the y-axis which generates a positive angular acceleration $\dot{q}$.

Table 4.4: Verification Test 1 Results

Derivatives	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{V}_x$	$\dot{V}_y$	$\dot{V}_z$	$\dot{q}_0$	$\dot{q}_x$	$\dot{q}_y$	$\dot{q}_z$	$\dot{p}$	$\dot{q}$	$\dot{r}$
Expected	10	0	0	2.7953	0	9.2066	0	0	0	0	0	11.4956	0
Simulation	10	0	0	2.7953	0	9.2066	0	0	0	0	0	11.4956	0
Error	0	0	0	0	0	0	0	0	0	0	0	0	0

Test 2

For this test, the drone will have a yaw and pitch of 90 degrees with zero degrees of roll. Values for position have been given to simply check that they do not affect calculations of the derivatives. A velocity of 5 m/s will be given in the y-direction and no velocity in other directions. As for the propellers, the only change is that α_r will now be negative. These values are shown in Equation 4.3.

$$\begin{aligned}\vec{X} &= \langle 10,10,10,0,5,0,0.5,-0.5,0.5,0.5,0,0,0\rangle \\ \vec{U} &= \langle 1000, 1000, \tfrac{\pi}{18}, -\tfrac{\pi}{18}\rangle\end{aligned} \tag{4.3}$$

Before looking at the results, some predictions about the derivatives can be made. First, because the drone is perpendicular to the velocity, a large deceleration should be expected in the y-direction due to the force produced by the aerodynamics model. This deceleration will be slightly complemented by a differential in actual thrust produced by the propellers. This is caused by the difference in propeller inflow velocity experienced by each propeller. Indeed, α_r being negative points the propeller into the flow increasing the inflow velocity and reducing its thrust whereas α_l points away from the flow having an equal but opposite inflow velocity increasing its thrust. The result is an additional deceleration in the y-direction. Secondly, a reduction of gravitational acceleration should be expected. Thirdly, $\dot{p}$ should be positive due to the opposite angles of the propellers, $\dot{q}$ should be large and negative due to the large aerodynamic moment and finally $\dot{r}$ should not be zero but positive because a moment is generated due to the differential thrust caused by the different inflow velocities.

Table 4.5: Verification Test 2 Results

Derivatives	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{V}_x$	$\dot{V}_y$	$\dot{V}_z$	$\dot{q}_0$	$\dot{q}_x$	$\dot{q}_y$	$\dot{q}_z$	$\dot{p}$	$\dot{q}$	$\dot{r}$
Expected	0	5	0	0	-12.7467	3.136	0	0	0	0	6.8979	-25.9105	1.65
Simulation	0	5	0	0	-12.7467	3.136	0	0	0	0	6.8979	-25.9105	1.65
Error	0	0	0	0	0	0	0	0	0	0	0	0	0

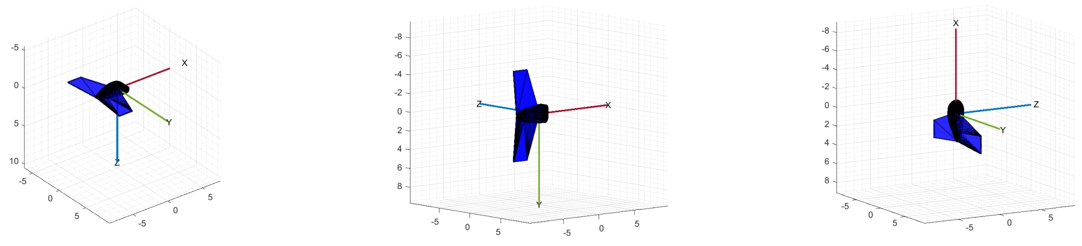
Looking at Table 4.5, the results match up exactly when rounded to the fourth digit. In practice, the results still match past four digits but for presenting the results four has been deemed sufficient.

Results of State Derivatives Verification

The results of both tests performed in this section give sufficient proof that the calculation of the state derivatives is performed correctly and thus is verified.

4.1.2. Full Simulation Verification

This section is dedicated to verifying the implementation of the full simulation. This entails verifying that the flight characteristics and behaviour of the drone in different scenarios is what is expected. The tests will consist of two drop tests with no thrust and a vertical spin test with thrust. The first two tests with no thrust aim to verify the correct behaviour of the aerodynamics model and the vertical spin test aims to verify the correct behaviour of the thrust force calculations. Before the results of each test, the expected flight characteristics of each test will be detailed and then compared to the actual results in the figure. For a better understanding of the tests, the starting orientations of the drone in each test are shown in Figure 4.1.



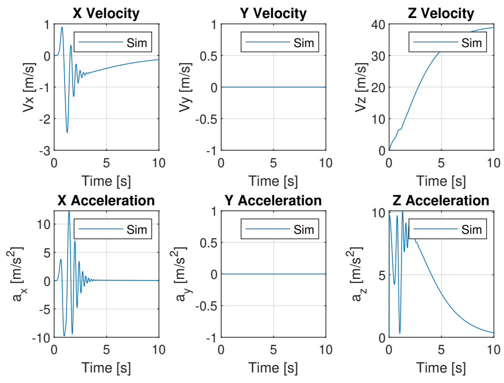
(a) Flat Drop **(b)** Sideway Drop **(c)** Vertical Spin

Figure 4.1: Starting Orientations of the Different Verifications

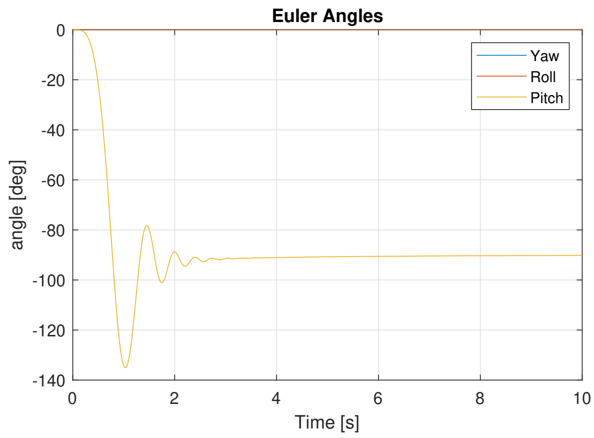
Flat Drop Test

The flat drop test is performed to verify the aerodynamic behaviour of the drone, more specifically, the lift and drag implementation from the ϕ aerodynamic theory model from [19]. The starting orientation of the drone is depicted in Figure 4.1a. As the centre of pressure is behind the centre of gravity, the drone is expected to drop and start pitching downwards once it has attained some downwards velocity. This pitching movement should oscillate a bit but should stabilise to -90 degrees which is pointing straight down. Additionally, due to the drag force, the Z acceleration should tend to zero with increasing vertical velocity.

Looking at the results in Figure 4.2 the drone indeed pitches forward and within three seconds is pointing straight down with a pitch value of -90 degrees. No velocity or acceleration in the y-direction can be observed which is expected as no forces are produced in that direction. Furthermore, the acceleration in the z-direction can be seen approaching zero after 10 seconds. The velocity at which the drone should no longer accelerate in this situation or in other words its terminal velocity is around 39.5559 m/s. Simulating this test up to 20 seconds, the velocity in the z-direction has a value of $39.551\ m/s$ and an acceleration of $0.00249\ m/s^2$ which slowly pushes the velocity to the expected terminal velocity confirming the proper implementation of the drag element of the aerodynamics.



(a) fig 1



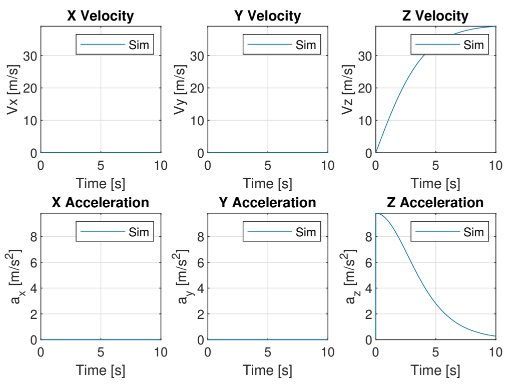
(b) fig 1

Figure 4.2: Flat Drop Test

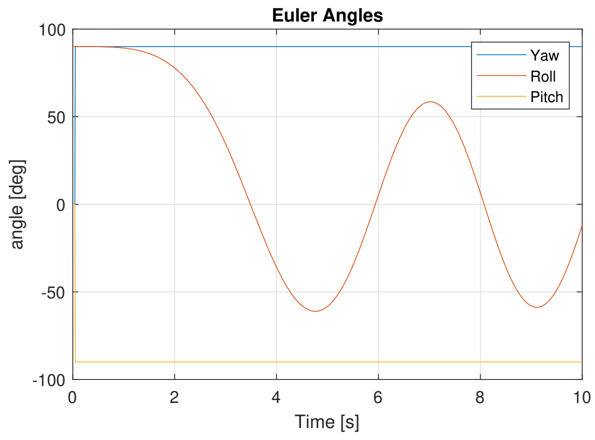
Sideway Drop Test

The flat drop test has verified the implementation of the aerodynamic lift and drag force, this sideway drop test is designed to verify the proper implementation of the lateral aerodynamic force implementation. The starting orientation of the drone is shown in Figure 4.1b. With no thrust during this test, the drone is expected to start falling and then with the velocity increasing the lateral force should start to rotate the drone around its z-axis as depicted in Figure 4.1b. The lateral force coefficient is much smaller than the coefficient generating lift, therefore slower and larger oscillations around the z-axis of the drone are expected.

The results of this test in Figure 4.3 indeed show an increase in velocity in the z-direction along with a large oscillation and slow oscillation of roll. The reason that it is roll that is oscillating and that yaw and pitch have large jumps in values at the start is due to the ambiguity of the definitions of these Euler angles when the drone is positioned with 90 degrees of roll as shown in Figure 4.1b. Changing the rotation sequence back to the more popular ZYX rotation sequence would not help much in terms of clarity as this test also has the drone rotate directly downwards which is ambiguous for this sequence as well. To clarify, it has been verified that this oscillation in roll translates to an oscillation around the body z-axis depicted in Figure 4.1b which is what is expected.



(a) fig 1



(b) fig 1

Figure 4.3: Sideway Drop Test

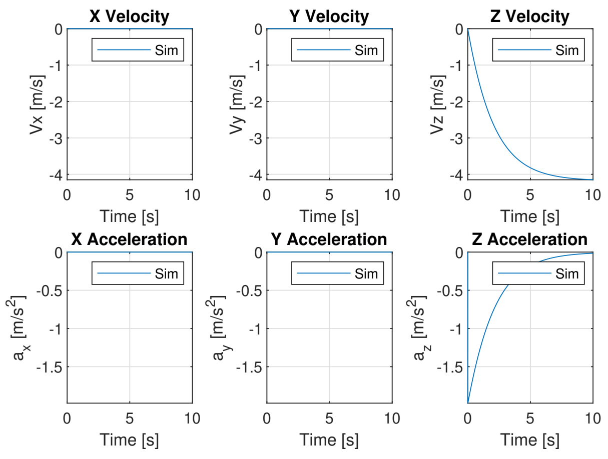
Vertical Spin Test

This vertical spin test is designed to verify the proper behaviour of the thrust model. The propeller angular velocities will be set to $1300\ rad/s$ with each nacelle pointing 5 degrees in the opposite direction of each other. These values are shown in Equation 4.4 using the previously mentioned format.

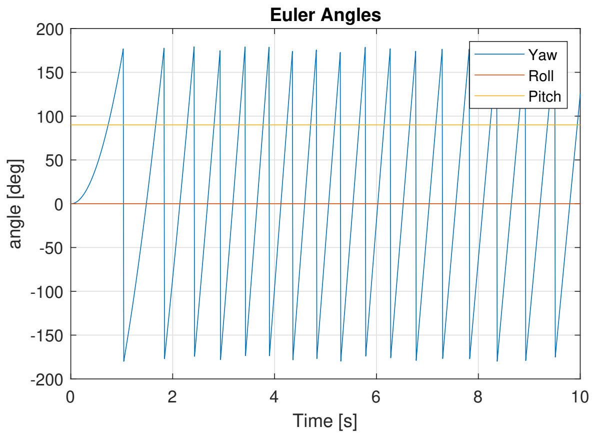
$$\vec{U} = \langle 1300, 1300, -\frac{\pi}{36}, \frac{\pi}{36} \rangle \tag{4.4}$$

With near full throttle, the drone is expected to start flying up even though the propellers are slightly angled. Furthermore, this vertical acceleration is expected to tend to zero as the inflow velocity will increase, decreasing the thrust. Due to the propeller nacelles being angled in opposite directions, the drone is expected to start spinning around its x-axis shown in Figure 4.1c. No aerodynamic forces due to drone rotation have been implemented, therefore there will be no resistance to rotation so the angular velocity of the drone should linearly increase. Furthermore, the angular velocity of the drone is not taken into account when calculating the inflow velocity of the propellers. This means no reduction in thrust should be expected due to the increasing spin of the drone.

Looking at Figure 4.4, the previously made predictions are correct. The drone slowly stops accelerating vertically and reaches a stable velocity and also starts spinning. However, the spin behaviour shown in Figure 4.4b is odd. The yaw rate slowly starts increasing progressively faster but then starts to slow down when it is expected to linearly increase. The slowing down of the yaw rate can be visualised by the slope of the yaw plot or by the increasing distance between the sharp peaks.



(a) fig 1



(b) fig 1

Figure 4.4: Vertical Spin Test

To investigate this issue, Figure 4.5 was created to compare the angular rate calculated by the equations of motion p_{req} and the angular rate actually displayed by the quaternion attitude p_{quat}. The left plot in Figure 4.5 shows both values evolve over the simulation time and shows the origin of the angular rate slowdown observed in Figure 4.4b. The right plot shows the error evolves with increasing p_req. This error has been identified as a quaternion dynamics integration error. This has been verified by reducing the timestep and observing a reduction in the error. Although in this scenario the errors get very large, it should be noted that such large angular velocities are typically never reached. To give a better idea of the magnitude of the error at more reasonable speeds, the error at 360 degrees per second is about 5 degrees per second.

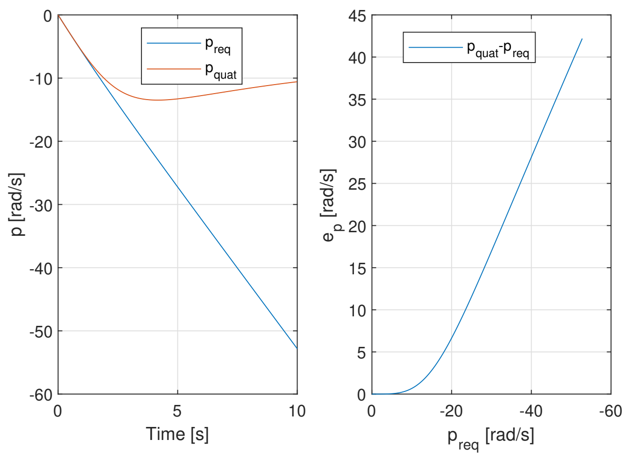


Figure 4.5: Quaternion Integration Angular Rate Error

It can be concluded that although in this specific scenario, the error gets very large, in practice, these angular rate values are never reached and therefore the error stays minimal. Additionally, the aerodynamic drag caused by the spinning of the drone is not modelled which means that during real flight, the angular velocity would never reach such high values. Therefore, no changes have to be made but this source of integration error should be checked if changes to the integration scheme are made.

5 Conclusion

Tailsitter hybrid MAVs offer the ability to fly in hover and horizontal flight allowing them to perform tasks that require hovering and the increased energy efficiency of using a wing for horizontal flight. A novel tilt-rotor tailsitter using leading-edge tilting propellers has been proposed in [1]. The replacement of ailerons by tilt-rotors is to tackle the reduced control authority of ailerons for pitch and roll control at lower to zero velocities. The work focused on attitude control of the platform using Incremental Nonlinear Dynamic Inversion which successfully followed commanded attitudes over the entire flight envelope. Using a simulation of this same platform, this thesis aims to develop a method of controlling the acceleration of this tilt-rotor tailsitter MAV which can then be used for position control of the drone.

Based on the previous success of [1] to control attitude using INDI and [17] using a cascaded INDI structure to control the acceleration of a fixed-rotor tailsitter, initial developments investigated the use of a cascaded INDI structure to control the platform. Issues were encountered when implementing this method which led to a change of methodology and was not further investigated. The new proposed method to control acceleration is a Nonlinear Model Predictive Controller. Using a model of the system and some high-level objectives defined in a cost function, this controller can calculate the optimal control input that minimises the value of the cost function.

Several simulations were performed to assess the performance of the NMPC. The first simulation tested the maximum forward velocity which the NMPC managed to get to 19 m/s. The second simulation tested the ability of the controller to slow back down to hover from a certain forward velocity. It managed this up to a velocity of 8 m/s which is significantly lower than the maximum forward velocity from hover. More complex tests were simulated namely a circular and square trajectory. The maximum velocities achieved for both tests were 10 and 8 m/s respectively. These performances were mainly limited by an issue in the prediction model where a large integration timestep was used which caused model instability. The use of sub-steps to reduce the integration timesteps fixed this issue and extended the performance of the NMPC. After the implementation of this fix, the Sub-stepping NMPC, as it will be referred to, flew to a maximum of about 24.5 m/s in the maximum forward velocity test where the controller could no longer accelerate due to the saturation of the propeller motors. The hover to forward flight to hover test was performed up to 24 m/s and was limited by the previously mentioned maximum forward velocity. As for the circular and square trajectories, maximum velocities increased for both up to 11 and 24 m/s respectively. Most importantly, failures past these velocities were no longer caused by the solver crashing but by performance failures. Based on the limitations of the Sub-stepping NMPC being caused by actuator saturation for the maximum forward velocity and the hover to forward flight to hover test, the Sub-stepping NMPC does show global effectiveness for acceleration control.

In terms of practical limitations, it was determined that in its current form, the NMPC would have difficulties running on hardware that is small enough to be placed on the drone such as a Raspberry Pi. To mitigate this, a feedforward neural network that can run much faster on a Raspberry Pi was trained using a process called behavioural cloning. Using training data from the NMPC, the neural network managed to approximate the complex control logic used by the NMPC. When comparing the NMPC and the NN using the same target velocities in the simulation, performance was reduced in all tests but the neural network showed similar general behaviour to that of the NMPC. This points to behavioural cloning being an effective method of transferring computationally intensive control logic to a neural network that can run faster.

References

[1] G.H.L.H. Lovell-Prescod, Z. Ma, and E.J.J. Smeur. “Attitude Control of a Tilt-rotor Tailsitter Micro Air Vehicle Using Incremental Control”. In: *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2023, pp. 842–849.

[2] E.J.J. Smeur, Q. Chu, and G.C.H.E. de Croon. “Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles”. In: *Journal of Guidance, Control, and Dynamics* 39 (Dec. 2015), pp. 1–12. DOI: `10.2514/1.G001490`.

[3] R. Chiappinelli et al. “Modeling and Control of a Passively-Coupled Tilt-Rotor Vertical Takeoff and Landing Aircraft”. In: (2019), pp. 4141–4147. DOI: `10.1109/ICRA.2019.8793606`.

[4] L. Bauersfeld and G. Ducard. “Fused-PID Control for Tilt-Rotor VTOL Aircraft”. In: (2020), pp. 703–708. DOI: `10.1109/MED48518.2020.9183031`.

[5] L.V. Santana, A.S. Brandão, and M. Sarcinelli-Filho. “Outdoor waypoint navigation with the AR.Drone quadrotor”. In: (2015), pp. 303–311. DOI: `10.1109/ICUAS.2015.7152304`.

[6] J. Willis, J. Johnson, and R.W. Beard. “State-Dependent LQR Control for a Tilt-Rotor UAV”. In: (2020), pp. 4175–4181. DOI: `10.23919/ACC45564.2020.9147931`.

[7] J.J.E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice-Hall International Editions. Prentice-Hall, 1991. ISBN: 9780130400499. URL: `https://books.google.nl/books?id=HddxQgAACAAJ`.

[8] A. Isidori. *Nonlinear Control Systems*. Communications and Control Engineering. Springer London, 1995. ISBN: 9783540199168. URL: `https://books.google.nl/books?id=fPGzHK%5C_pto4C`.

[9] S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Interdisciplinary Applied Mathematics. Springer New York, 2013. ISBN: 9781475731088. URL: `https://books.google.nl/books?id=j%5C_PiBwAAQBAJ`.

[10] S.A. Snell, D.F. Enns, and W.L. Garrard. “Nonlinear inversion flight control for a supermaneuverable aircraft”. In: *Journal of Guidance, Control, and Dynamics* 15.4 (1992), pp. 976–984. DOI: `10.2514/3.20932`. eprint: `https://doi.org/10.2514/3.20932`. URL: `https://doi.org/10.2514/3.20932`.

[11] D.J. Bugajski and D.F. Enns. “Nonlinear control law with application to high angle-of-attack flight”. In: *Journal of Guidance, Control, and Dynamics* 15.3 (1992), pp. 761–767. DOI: `10.2514/3.20902`. eprint: `https://doi.org/10.2514/3.20902`. URL: `https://doi.org/10.2514/3.20902`.

[12] J. Hauser, S. Sastry, and G. Meyer. “Nonlinear Control Design for Slightly Non-minimum Phase Systems: Application to V/STOL Aircraft”. In: *Automatica* 28 (Aug. 1992). DOI: `10.1016/0005-1098(92)90029-F`.

[13] D. Enns et al. “Dynamic inversion: an evolving methodology for flight control design”. In: *International Journal of Control* 59.1 (1994), pp. 71–91. DOI: `10.1080/00207179408923070`. eprint: `https://doi.org/10.1080/00207179408923070`. URL: `https://doi.org/10.1080/00207179408923070`.

[14] T.J. Koo and S. Sastry. *Output tracking control design of a helicopter model based on approximate linearization*. Vol. 4. 1998, 3635–3640 vol.4. DOI: `10.1109/CDC.1998.761745`.

[15] D. Lee and S. Sastry. “Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. International Journal of Control, Automation and Systems, 7(3), 419-428”. In: *International Journal of Control, Automation and Systems* 7 (June 2009), pp. 419–428. DOI: `10.1007/s12555-009-0311-8`.

[16] R. van ’t Veld, E. Van Kampen, and Q. Chu. “Stability and Robustness Analysis and Improvements for Incremental Nonlinear Dynamic Inversion Control”. In: *AIAA Guidance, Navigation, and Control Conference* (Jan. 2018). DOI: `10.2514/6.2018-1127`.

[17] E.J.J. Smeur, M. Bronz, and G.C.H.E. de Croon. “Incremental Control and Guidance of Hybrid Aircraft Applied to a Tailsitter Unmanned Air Vehicle”. English. In: *Journal of Guidance, Control, and Dynamics: devoted to the technology of dynamics and control* 43.2 (2020), pp. 274–287. ISSN: 0731-5090. DOI: 10.2514/1.G004520.

[18] E. Tal and S. Karaman. “Global Trajectory-tracking Control for a Tailsitter Flying Wing in Agile Uncoordinated Flight”. In: *AIAA AVIATION 2021 FORUM* (Aug. 2021). DOI: 10.2514/6.2021-3214.

[19] L. Ribeiro Lustosa, F. Defaÿ, and J. Moschetta. “Global Singularity-Free Aerodynamic Model for Algorithmic Flight Control of Tail Sitters”. In: *Journal of Guidance, Control, and Dynamics* 42 (Dec. 2018), pp. 1–14. DOI: 10.2514/1.G003374.

[20] R. Ritz and R. D’Andrea. “A The controller for flying wing tailsitter vehicles”. In: (2017), pp. 2731–2738. DOI: 10.1109/ICRA.2017.7989318.

[21] G. Chowdhary and R. Jategaonkar. “Aerodynamic parameter estimation from flight data applying extended and unscented Kalman filter”. In: *Aerospace Science and Technology* 14.2 (2010), pp. 106–117. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2009.10.003. URL: https://www.sciencedirect.com/science/article/pii/S1270963809000650.

[22] M. Selig. “Modeling Propeller Aerodynamics and Slipstream Effects on Small UAVs in Realtime”. In: *AIAA Atmospheric Flight Mechanics Conference 2010* (Aug. 2010). DOI: 10.2514/6.2010-7938.

[23] M. Selig. *PROPID - Software for Horizontal-Axis Wind Turbine Design and Analysis*. 1995. URL: https://m-selig.ae.illinois.edu/propid.html.

[24] W. Khan and M. Nahon. “Toward an Accurate Physics-Based UAV Thruster Model”. In: *IEEE/ASME Transactions on Mechatronics* 18.4 (2013), pp. 1269–1279. DOI: 10.1109/TMECH.2013.2264105.

[25] D. Stojcsics. “Autonomous Waypoint-based Guidance Methods for Small Size Unmanned Aerial Vehicles”. In: *Acta Polytechnica Hungarica* 11 (Jan. 2014), pp. 215–233.

[26] D. Lawrence, E. Frew, and W. Pisano. “Lyapunov Vector Fields for Autonomous Unmanned Aircraft Flight Control”. In: *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM* 31 (Sept. 2008), pp. 1220–1229. DOI: 10.2514/1.34896.

[27] S. Park, J Deyst, and J. How. “A New Nonlinear Guidance Logic for Trajectory Tracking”. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*. DOI: 10.2514/6.2004-4900. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2004-4900. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2004-4900.

[28] T. Stastny. “L1 guidance logic extension for small UAVs: handling high winds and small loiter radii”. In: (Apr. 2018).

[29] R. Curry et al. “L+2, an improved line of sight guidance law for UAVs”. In: (2013), pp. 1–6. DOI: 10.1109/ACC.2013.6579804.

[30] M. Fliess et al. “Flatness and defect of non-linear systems: introductory theory and examples”. In: *International Journal of Control* 61 (June 1995), pp. 13–27. DOI: 10.1080/00207179508921959.

[31] P. Kotaru, G. Wu, and K. Sreenath. “Differential-flatness and control of quadrotor(s) with a payload suspended through flexible cable(s)”. In: (2018), pp. 352–357. DOI: 10.1109/INDIANCC.2018.8308004.

[32] E. Tal and S. Karaman. “Accurate Tracking of Aggressive Quadrotor Trajectories Using Incremental Nonlinear Dynamic Inversion and Differential Flatness”. In: *IEEE Transactions on Control Systems Technology* 29.3 (2021), pp. 1203–1218. DOI: 10.1109/TCST.2020.3001117.

[33] M. Behrendt. *A basic working principle of Model Predictive Control*. https://commons.wikimedia.org/wiki/File:MPC_scheme_basic.svg. 2009.

[34] W. Zhou et al. “Position control of a tail-sitter UAV using successive linearization based model predictive control”. In: *Control Engineering Practice* 91 (Oct. 2019), p. 104125. DOI: 10.1016/j.conengprac.2019.104125.

[35] H. Gu et al. “Coordinate Descent Optimization for Winged-UAV Design”. In: *Journal of Intelligent & Robotic Systems* 97 (Jan. 2020). DOI: 10.1007/s10846-019-01020-2.

[36] R. Farhadi et al. “Estimation of the lateral aerodynamic coefficients for skywalker x8 flying wing from real flight-test data”. In: *Acta Polytechnica* 58 (Apr. 2018), p. 77. DOI: `10.14311/AP.2018.58.0077`.

[37] M. Faessler, A. Franchi, and D. Scaramuzza. “Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories”. In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 620–626. DOI: `10.1109/LRA.2017.2776353`.

[38] Maplesoft, a division of Waterloo Maple Inc. *Maple 2022*. Waterloo, Ontario.

A Literature Study Initial Proposal

A.1. Proposed Kinematic Controller: Three-Dimensional L_1 Path Following

Traditional L_1 guidance is made to be implemented in the horizontal plane and only gives a reference lateral acceleration that the vehicle should follow. This means that the third dimension which is altitude is controlled separately. To the best knowledge of the author, converting this method to follow a three-dimensional trajectory has not been performed. The advantage of doing so would remove the separate altitude control and result in a simple kinematic controller for three-dimensional trajectories.

This method will be called *3D L_1 Path Following* and will now be explained. Looking at Figure A.1, the 3D variant of L_1 path following makes use of the same concepts as its two-dimensional counterpart. In Figure A.1, the blue dotted line is the desired trajectory the drone needs to follow and in the scenario depicted in the figure, the desired trajectory is at a higher altitude than the drone currently is. Just as in the two-dimensional scenario, the angle η is calculated between the velocity vector $\vec{V}$ and $\vec{L}_1$. Using this, the magnitude of the lateral acceleration $\vec{a}_{cmd}$ can be calculated using Equation A.1. The additional challenge for 3D L_1 comes from generating the $\vec{a}_{cmd}$ vector in the correct orientation.

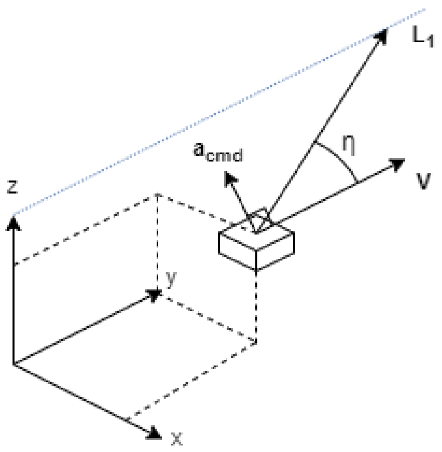


Figure A.1: 3D L_1 Path Following Overview

To start, the magnitude of $\vec{a}_{cmd}$ is calculated in the same way as Equation 3.29 but for clarity, Equation A.1 is given in the context of magnitude of vectors.

$$a_{cmd} = 2\frac{\left\|\vec{V}\right\|^2}{\left\|\vec{L}_1\right\|}\sin\eta \tag{A.1}$$

What is left to calculate is a unit vector in the direction of $\vec{a}_{cmd}$, then multiply it by the required magnitude. There are multiple ways to do this, the method that will be described here uses vector operations and quaternion rotation of vectors. Some important information about $\vec{a}_{cmd}$ is known. First, the vector lies in the plane generated by $\vec{V}$ and $\vec{L}_1$. Second, $\vec{a}_{cmd}$ is always perpendicular to $\vec{V}$ in the plane mentioned previously. Finally, the 90-degree rotation from $\vec{V}$ to $\vec{a}_{cmd}$ is always performed in the direction of $\vec{L}_1$ with respect to $\vec{V}$.

Knowing this, a vector $\vec{R}$ is calculated in Equation A.2. This vector will serve as the vector around which the quaternion rotation will be performed. It should be noted that the vector is normalised for it to work in the quaternion rotation.

$$\vec{R} = \frac{\vec{V} \times \vec{L}_1}{\left\| \vec{V} \times \vec{L}_1 \right\|} \tag{A.2}$$

Next, in Equation A.3, the setup of the different quaternions for the rotation can be found. q_V is built using the x, y, z components of $\vec{V}$. q_R and $q_{R'}$ are built using the x, y, z components of $\vec{R}$ and the magnitude of the desired rotation θ which in this case will be $\frac{\pi}{2}$.

$$\begin{aligned} q_V &= [0, \vec{V}_x, \vec{V}_y, \vec{V}_z] \\ q_R &= [\cos(\tfrac{\theta}{2}), \vec{R}_x \sin(\tfrac{\theta}{2}), \vec{R}_y \sin(\tfrac{\theta}{2}), \vec{R}_z \sin(\tfrac{\theta}{2})] \\ q_{R'} &= [\cos(\tfrac{\theta}{2}), -\vec{R}_x \sin(\tfrac{\theta}{2}), -\vec{R}_y \sin(\tfrac{\theta}{2}), -\vec{R}_z \sin(\tfrac{\theta}{2})] \end{aligned} \tag{A.3}$$

With the quaternions set up, all that is left is to multiply them together using Equation A.4 where H stands for a Hamilton product.

$$q_{final} = H(H(q_R, q_V), q_{R'}) \tag{A.4}$$

Extracting the last three components of q_{final} gives a vector $\vec{U}$ in the desired orientation. However, this vector still has the magnitude of $\vec{V}$ so it must be normalised, then multiplied by a_{cmd}.

$$\vec{a}_{cmd} = \frac{\vec{U}}{\left\| \vec{U} \right\|} a_{cmd} \tag{A.5}$$

A.2. Proposed INDI Controller Derivation

The controller deals with linear accelerations and thus only requires knowledge of the sum of forces applied to the body. To this effect, the sum of forces in the Inertial Reference Frame (IRF) which is $\vec{F}_I$ is given in Equation A.6 where $\vec{T}_I$ is the thrust in the IRF, $\vec{F}_{aero}$ the aerodynamic forces in the IRF and $\vec{F}_g$ is the gravitational force in the IRF.

$$\sum \vec{F}_I = \vec{F}_g + \vec{T}_I + \vec{F}_{aero} \tag{A.6}$$

To get acceleration, both sides are divided by the mass as is shown in Equation A.7.

$$\vec{a} = \vec{g} + \frac{1}{m}(\vec{T}_I + \vec{F}_{aero}) \tag{A.7}$$

Now, both $\vec{T}_I$ and $\vec{F}_{aero}$ will be expanded. First, $\vec{T}_I$ is defined in Equation A.8 where $\vec{T}_B$ is the thrust vector in the body reference frame (BRF) and M_B^I is the transformation matrix from BRF to IRF (See Equation A.11). $\vec{T}_B$ consists of a single component in the z axis. It is important to note that for vectors with subscripts x, y or z this indicates that the value of the respective component is taken and is a scalar value and not a vector. This means that $\vec{T}_{B_z}$ is the scalar value of the z component of $\vec{T}_B$.

$$\vec{T}_I = M_B^I \vec{T}_B = M_B^I \begin{bmatrix} 0 \\ 0 \\ \vec{T}_{B_z} \end{bmatrix} \tag{A.8}$$

Next, $\vec{F}_{aero}$ which is the aerodynamics model is explained. The aerodynamics model is based on ϕ-theory developed in [19] and applied in [18] for a fixed-rotor tail-sitter. The model only requires C_{L_V} and C_{D_V} which are the lift and drag coefficients respectively.

$$\vec{F}_{aero} = M_B^I \begin{bmatrix} (2\pi + C_{d_0})\vec{V}_{B_x} \\ 0 \\ C_{d_0}\vec{V}_{B_z} \end{bmatrix} \left\|\vec{V}_B\right\| \tag{A.9}$$

Finally, $\vec{g}$ is simply the acceleration caused by gravity in the NED inertial reference frame which means the acceleration is in the z-component of the vector as is shown in Equation A.10.

$$\vec{g} = \begin{bmatrix} 0 \\ 0 \\ 9.81 \end{bmatrix} \tag{A.10}$$

$$M_B^I = \begin{bmatrix} c\theta c\psi - s\phi s\theta s\psi & -c\phi s\psi & s\theta c\psi + s\phi c\theta s\psi \\ c\theta s\psi + s\phi s\theta c\psi & c\phi c\psi & s\theta s\psi - s\phi c\theta c\psi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \tag{A.11}$$

With the thrust and aerodynamics forces defined. The model can be expressed in the form $\vec{a} = \vec{f}(\vec{x}, \vec{u})$ and is shown in Equation A.12. Before applying INDI to this system, the inputs and states need to be identified. The variables that are being controlled and thus are inputs are $\vec{T}_{B_z}, \phi, \theta, \psi$ so $\vec{u} = [\vec{T}_{B_z}, \phi, \theta, \psi]^T$. As for the states, these are simply the components of the wind velocity in the BRF. Thus, $\vec{x} = [\vec{V}_{B_x}, \vec{V}_{B_y}, \vec{V}_{B_z}]^T$ and $\left\|\vec{V}_B\right\| = \sqrt{\vec{V}_{B_x}^2 + \vec{V}_{B_y}^2 + \vec{V}_{B_z}^2}$.

$$\vec{a} = \vec{g} + \frac{1}{m} \begin{bmatrix} (c\theta c\psi - s\phi s\theta s\psi)(2\pi + C_{d_0})\vec{V}_{B_x} \left\|\vec{V}_B\right\| + (s\theta c\psi + s\phi c\theta s\psi)(\vec{T}_{B_z} + C_{d_0}\vec{V}_{B_z} \left\|\vec{V}_B\right\|) \\ (c\theta s\psi + s\phi s\theta c\psi)(2\pi + C_{d_0})\vec{V}_{B_x} \left\|\vec{V}_B\right\| + (s\theta s\psi - s\phi c\theta c\psi)(\vec{T}_{B_z} + C_{d_0}\vec{V}_{B_z} \left\|\vec{V}_B\right\|) \\ (-c\phi s\theta)(2\pi + C_{d_0})\vec{V}_{B_x} \left\|\vec{V}_B\right\| + (c\phi c\theta)(\vec{T}_{B_z} + C_{d_0}\vec{V}_{B_z} \left\|\vec{V}_B\right\|) \end{bmatrix} \tag{A.12}$$

From this point, the Taylor expansion of $\vec{a}$ can be performed around $(\vec{x}_0, \vec{u}_0)$ and the general method described in Equation 3.2.1 can be followed.

B Model Identification

Table B.1: Model Constants Required for Simulation with their Source

Symbol	Name	Value	Units	Source
C_{L_α}	Lift Coefficient Slope	4	[-]	[35]
C_{d_0}	Minimum Drag Coefficient	0.05	[-]	[35]
C_{y_0}	Lateral Force Coefficient	0.05	[-]	[36]
S	Wing Surface Area	0.26	$[m^2]$	Drone Spec Sheet
g	Gravity Acceleration	9.81	$[m/s^2]$	-
m	Mass	1.27	$[kg]$	Measured
ρ	Air Density	1.225	$[kg/m^3]$	ISA Sea Level Density
I	Inertia Matrix	$diag(0.065, 0.009, 0.0662)$	$[kg \cdot m^2]$	[1]
P_P	Propeller Pitch	0.127	$[m]$	Propeller Spec Sheet
$\vec{D}_{CP}$	Vector from CG to CP	[-0.015,0,0]	$[m]$	See B.1
$\vec{D}_{T_l}$	Vector from CG to T_l	[0.135,-0.3,0]	$[m]$	Measured
$\vec{D}_{T_r}$	Vector from CG to T_r	[0.135,0.3,0]	$[m]$	Measured

B.1. Distance between centre of gravity and centre of pressure

Estimating the location of the centre of pressure is a complex task as it requires aerodynamic modelling of the drone over multiple flight conditions as the centre of pressure moves with changing angle of attack. As this work is mainly interested in evaluating the ability of controllers to control the main dynamic characteristics of this drone, such accurate modelling is not necessary. The location of the centre of pressure is thus assumed to be fixed at a location of 1.5 centimetres behind the centre of gravity.

C Initial Investigation

This section is dedicated to documenting the other methods that were evaluated/tested to control this drone configuration but failed. These are a cascaded INDI configuration and a combination of differential flatness and INDI. Additionally, a differential flatness derivation was performed to investigate the possibility of generating feedforward control inputs based on a desired trajectory.

C.1. Longitudinal Model

The simplified tail-sitter model used to test the previously mentioned control algorithms is constrained to two dimensions of motion with z pointing down and x being horizontal and positive to the right. Additionally, the tail-sitter may rotate around its centre of gravity which means it can change its pitch angle θ. The states and inputs of the system are detailed in Equation C.1 where x and z are the horizontal and vertical positions respectively, V_x and V_z are the inertial velocities in their respective axes, θ is the pitch angle and ω is the pitch rate. Finally, T and α_T are the thrust and thrust angle respectively.

$$\begin{aligned} \vec{x} &= \langle x, y, V_x, V_z, \theta, \omega \rangle \\ \vec{u} &= \langle T, \alpha_T \rangle \end{aligned} \tag{C.1}$$

Using these states and input, the derivatives of each state can be defined to be able to model the system. These derivatives are shown in Equation C.3.

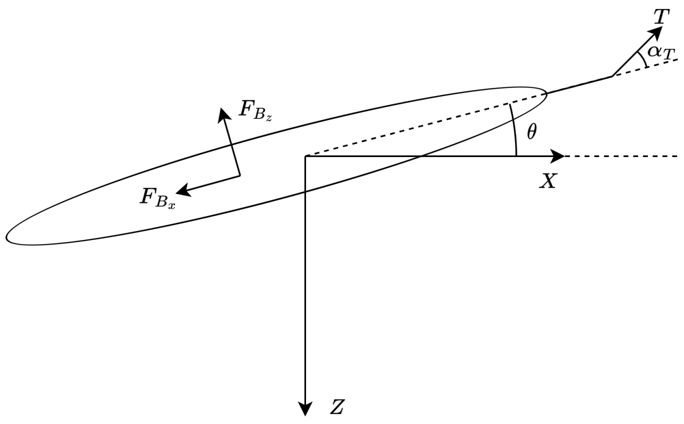


Figure C.1: Diagram of longitudinal tail-sitter model

The forces in Figure C.1 are detailed in Equation C.2 where F_g is the force caused by gravity, m is the mass of the drone, g is the acceleration caused by gravity, F_{B_x} and F_{B_z} are the aerodynamic forces using the singularity-free model developed in [19] expressed in the BRF, ρ is the air density, S the wing surface area, V_x and V_z the inertial velocities, C_{d_0} the zero-lift drag coefficient, C_{L_α} the linear lift-curve slope, V_{B_x} and V_{B_z} the drone velocities in the BRF.

$$\begin{aligned} F_g &= mg \\ F_{B_x} &= -\tfrac{1}{2}\rho S\sqrt{V_x^2+V_z^2}C_{d_0}V_{B_x} \\ F_{B_z} &= -\tfrac{1}{2}\rho S\sqrt{V_x^2+V_z^2}(C_{L_\alpha}+C_{d_0})V_{B_z} \end{aligned} \tag{C.2}$$

$$\begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{V}_x \\ \dot{V}_z \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} V_x \\ V_z \\ \frac{1}{m}(T\cos(\alpha_T+\theta) - F_{B_x}\cos(\theta) - F_{B_z}\sin(\theta)) \\ g + \frac{1}{m}(-T\sin(\alpha_T+\theta) + F_{B_x}\sin(\theta) - F_{B_z}\cos(\theta)) \\ \omega \\ \frac{1}{I}\left(T\sin(\alpha_T)\, d_t - F_{B_z} d\right) \end{bmatrix} \tag{C.3}$$

C.2. Cascaded INDI

Attempts to control the acceleration of the longitudinal model previously mentioned using cascaded INDI have proven to be unsuccessful. In a minority of scenarios, the controller can follow acceleration references but in most cases, the controller fails. An overview of the control loop is presented in Figure C.2.

Starting from the top, an acceleration reference is given to the Linear Acceleration INDI controller. This controller calculates using INDI a command thrust T_c and pitch angle θ_c whilst assuming that the thrust angle α_T is constant for that time-step. The commanded thrust is sent to the actuator and the commanded attitude is given to the Angular Acceleration INDI controller that will calculate the thrust angle α_{T_c} to achieve the commanded pitch angle.

Looking at the control loop overview, the calculation of T and α_T are separate. This is troubling as both controls are extremely coupled. A change in one inherently changes the efficacy of the other. This is further compounded by the separation of Linear and Angular acceleration calculations. As the thrust is calculated with as objective to follow a desired linear acceleration, it is possible that this objective does not always coincide with the objective of the angular acceleration controller that calculates the value of α_T. The suspected issue here is essentially that a multi-objective problem is being solved as two separate single-objective problems whilst having controls that are extremely coupled.

Another proposal to solve the control issue for this drone is to skip the Angular Acceleration INDI and directly calculate the thrust T and thrust angle α_T based on the required linear accelerations. Mathematically, this is possible as there are two states and two control inputs which means the system is solvable. Practically, this does not work due to the following.

Through the fundamentals of INDI, it is possible to deduce that having T and α_T as the output of the Linear Acceleration INDI would mean that the aerodynamics of the system would be completely ignored in the calculation of these control inputs which leads to poor controller performance.

C.3. Differential Flatness combined with INDI

With cascaded INDI not working, the next idea was to adapt the work done in [18] where differential flatness was combined with INDI to control a fixed-rotor tailsitter. Adapting the method used in [18] which in itself is a cascaded approach to the control problem is possible but practically does not work. A critical assumption in the work done in [18] is that the orientation of the total thrust vector remains constant in the Body Reference Frame. This is obviously not the case of the drone where any change of propeller nacelle angle changes the orientation of the total thrust vector in the BRF.

It is technically possible to assume that for each timestep the total thrust vector is constant based on the current nacelle angles. This allows for the desired attitude and total thrust to be calculated. However, to achieve the desired attitude a combination of differential thrust and more importantly a change in propeller nacelle angles are required. This of course breaks the initial assumption that the total thrust vector is constant in the BRF and invalidates the calculations made in the first step.

This highlights the advantage of having control actuators that have minimal overlap over the states that they control. The use of ailerons purely for attitude control is most likely what allows [18] to make the necessary assumptions to use differential flatness in this scenario.

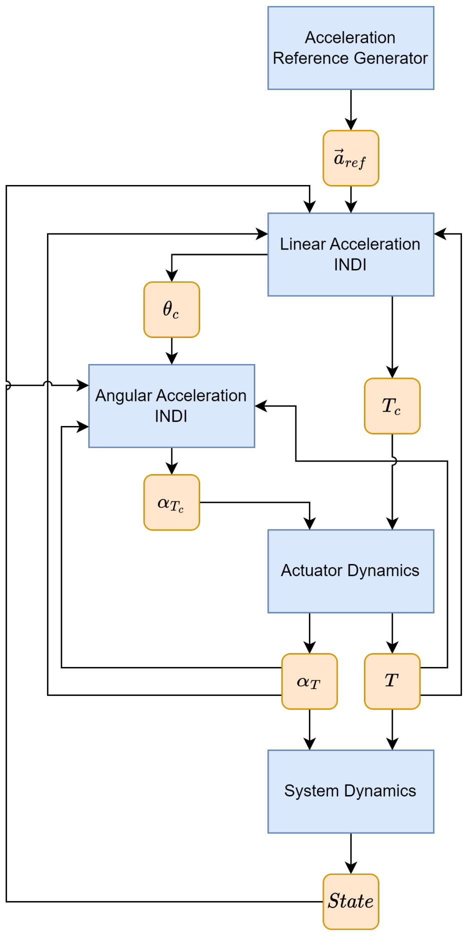


Figure C.2: Cascaded INDI: Control Loop Overview

C.4. Feedforward Control Input Generation Through Differential Flatness

Finally, it was investigated whether feedforward control inputs based on a desired trajectory could be calculated using differential flatness. If possible, a feedback control law could potentially be designed to control the drone as was done in [37].

$$y = [x, z]^T \tag{C.4}$$

Assuming that the flat output y consists of the states x and z in the longitudinal model. Derivatives of this flat output can be taken to make $\dot{y}$.

$$\dot{y} = \begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} V_x \\ V_z \end{bmatrix} \tag{C.5}$$

$$\ddot{y} = \begin{bmatrix} \ddot{x} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \dot{V}_x \\ \dot{V}_z \end{bmatrix} \tag{C.6}$$

At the second derivative of the flat output $\ddot{y}$, expressions for $\ddot{x}$ and $\ddot{z}$ are available from Equation C.3 as these are the equations for linear acceleration. Theoretically, T and α_T could be solved at this point as all other variables are known states. This is not typically done as this method does not impose any constraints on the attitude θ and its derivatives which is undesirable for the general controllability and stability of the drone. Therefore, additional derivatives are taken until the angular acceleration $\ddot{\theta}$ appears from the equations.

$$\ddddot{y} = \left[\begin{array}{c} \ddddot{x} \\ \ddddot{z} \end{array} \right] = f(T, \dot{T}, \ddot{T}, \alpha_T, \dot{\alpha}_T, \ddot{\alpha}_T, V_x, \dot{V}_x, \ddot{V}_x, V_z, \dot{V}_z, \ddot{V}_z, \theta, \dot{\theta}, \ddot{\theta}) \tag{C.7}$$

Looking at Equation C.7 it can be seen that the second derivative of the pitch angle $\ddot{\theta}$ appears which is the angular acceleration. $\ddot{\theta}$ can be calculated using Equation C.8 and then replaced in Equation C.7

$$\ddot{\theta} = \dot{\omega} = \frac{1}{I} \left(T \sin\left(\alpha_T\right) d_t - F_{B_z} d \right) \tag{C.8}$$

With $\ddot{\theta}$ replaced using Equation C.8, Equation C.7 can be solved for T and α_T. This can be done as the derivatives of the states can be measured and estimated and actuator dynamics can be added to T and α_T to get their respective derivatives. Assuming that all the variables of f are calculated properly, the solutions T and α_T should ensure the system follows the flat output and its derivatives.

Implementation

Theoretically, this method can work for this system, however, practically, some issues arise. Taking the time derivative twice of the acceleration equations generates a large expression with many terms. The expression for the fourth derivative is still not too large but once this expression is solved for T and then consequently α_T, the expressions become very large. After solving for T in Maple 2022 [38], the resulting expression contains over 270000 characters. This expression then needs to be used to be able to solve for α_T by replacing all instances of T in the next equation with this large expression. Additional manipulation of this expression will only make it larger and solving for α_T which is contained in either sine or cosine functions will make it very difficult to solve.

Additionally, issues related to the terms calculated also arise such as some of the states being raised to the power of 10. This is not necessarily a problem if it only concerns a few terms, but the combination of a very large amount of terms combined with these large powers could quickly cause overflow errors during the calculations.

Finally, it is very important to notice that this derivation is for the simplified longitudinal system which only contains six states and two inputs. For every extra state and input added to the system, an extra term is added due to its dependency on time. The 6 degree of freedom model used in this paper has 20 states and 4 control inputs which increases the amount of variables dependent on time by 16. With this, it is expected that the already large expressions will become significantly larger.