University of Pisa

MSc in Robotics and Automation Engineering

Course: Process Control - prof. Manolo Garabini

Thrust Vector Rocket

Simone Cirelli - n° 624353

simone.cirelli01@gmail.com

Delivery date: December 2024

Contents

0 Assignment & Project Overview

The course *Processes Control* requires students to prepare a project in which they select and stabilize a system, and implement control strategies for it.

For this purpose, I chose a thrust vector rocket. This paper presents the system's analysis, linearization, and the control techniques applied.

Additionally, a physical implementation has been developed. The rocket features an actuated gimbal to control the thrust direction and uses an electric propeller instead of a combustion engine for safety and ease of supply. A bench test was conducted with a 2-DoF desk-mounted support, allowing the rocket to rotate around its center of mass to simulate in-flight dynamics.

Two case studies were analyzed:

1. **Ascent**: In this phase, the throttle is fixed at its maximum limit to ensure the fastest possible climb. The two available control inputs are the deflection angles of the thrust direction relative to the rocket axis.
2. **Hovering**: Here, the rocket is studied in a hovering condition. This phase provides the most control flexibility, as the throttle becomes an additional input alongside the thrust direction angles.

This presentation is organised as follows:

1. **Introduction**: System overview and state-space equations.
2. **System Analysis**: Analysis, linearization, and system dynamics.
3. **Ascent Phase Control**: Control strategies for ascent, with fixed throttle and adjustable thrust direction.
4. **Hovering Phase Control**: Control techniques for hovering, using both throttle and thrust direction.
5. **Physical Implementation**: Physical model with 2-DoF bench test to simulate dynamics and demonstrate control effectiveness.

1 Introduction

The objective of this project is to implement and analyze the control of a thrust vector rocket. Thrust vectoring is a technique for controlling the trajectory and stability of a rocket by adjusting the direction of the propeller along two axes to generate torque.

The system analysis focuses primarily on the second scenario (i.e., with throttle control), as it is the more complex case. From this, the first scenario can be derived with relative ease.

Different control strategies have been applied to address the two scenarios:

- **PID Controller**: Chosen for the ascent phase due to its simplicity in implementation and tuning.
- **Optimal Control**: Applied during the hovering phase to move the rocket between spatial positions.
- **Model Predictive Control (MPC)**: A robust closed-loop strategy used in the hovering phase, capable of satisfying physical constraints such as upper and lower bounds on throttle and rotation angles.
- **Cascade Control**: A technique to allow to control two variables using only one input.

1.1 The System

The mechanical system under investigation consists of a rocket equipped with a propeller that can be rotated along two orthogonal directions to adjust the rocket's position and maintain its vertical stability.

Although the rocket operates in 3D space, the system is symmetric, and the analysis considers only the linearized system around its equilibrium point. This symmetry allows us to decouple the motion along the two directions, as they can be considered independent of each other. Specifically, the analysis is conducted on the pitch movement, and the resulting controller is subsequently applied to the roll movement as well.

Figure 1 illustrates the 2D schematization of the system. The thrust vector can be deflected by an angle ϕ relative to the rocket's main axis. By adjusting ϕ, the propulsion system generates a torque that stabilizes the rocket and controls its orientation.

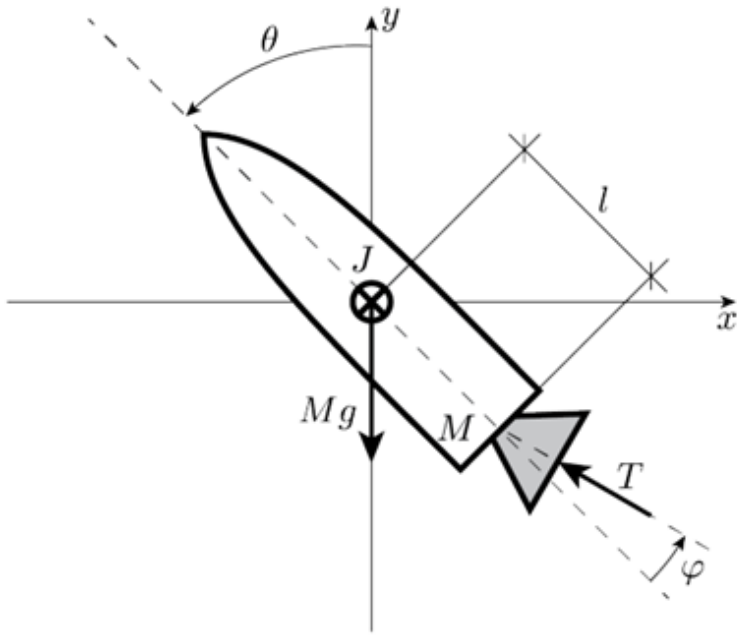


Figure 1: Schematization of the thrust vector rocket

1.2 Variables

Symbol (Unit)	Description	Nominal Value
M (kg)	Mass of the rocket	0.993 kg
J (kg·m^2)	Moment of inertia (relative to the pitch axis)	$0.088kg \cdot m^2$
T	Force provided by the propulsion system	2.3 kg
φ	Angle between T and the axes of the rocket	-
θ	Inclination of the rocket from the y axis	-
b_x	x-component of the aerodynamic drag	0.1 N*s/m
b_y	y-component of the aerodynamic drag	0.1 N*s/m
l	Distance CoM $\Rightarrow$ centre of rotation of the propulsor	0.23 m

The rocket has three degrees of freedom (x,y, θ), and two variables to control (the propeller angle ϕ and the propeller force T).
The equations governing the dynamics are as follows:

$$\begin{cases} M\ddot{x} &= -T\sin(\theta+\varphi) - b_x\dot{x} \\ M\ddot{y} &= T\cos(\theta+\varphi) - Mg - b_y\dot{y} \\ J\ddot{\theta} &= -Tl\sin(\varphi) \end{cases}$$

2 System Analysis

2.1 State-space form

To represent the equations of a thrust vector rocket in state-space form, we need to rewrite them as a set of first-order differential equations by introducing a state vector.

Step 1: Define the State Variables

We define the state vector $\mathbf{x}$ with six state variables and the vector $\mathbf{u}$ with the two inputs:

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ \theta \\ \dot{\theta} \end{bmatrix}, \quad u = \begin{bmatrix} T \\ \varphi \end{bmatrix}$$

where:

- x and y represent the horizontal and vertical positions of the rocket.
- $\dot{x}$ and $\dot{y}$ are the horizontal and vertical velocities.
- θ is the orientation of the rocket.
- $\dot{\theta}$ is the angular velocity.
- T is the thrust.
- φ is the thrust angle.

Now we can rewrite the equations in terms of these state variables.

Step 2: Rewrite as a System of First-Order Differential Equations

To convert each second-order differential equation into a system of first-order equations, we let:

- $\dot{x}_1 = x_2$ (where $x_1 = x$ and $x_2 = \dot{x}$)
- $\dot{x}_3 = x_4$ (where $x_3 = y$ and $x_4 = \dot{y}$)
- $\dot{x}_5 = x_6$ (where $x_5 = \theta$ and $x_6 = \dot{\theta}$)

Then, the equations become:

1. For the horizontal acceleration:

$$M\dot{x}_2 = -T\sin(x_5 + \varphi) - b_x x_2$$

which simplifies to:

$$\dot{x}_2 = -\frac{T}{M}\sin(x_5 + \varphi) - \frac{b_x}{M}x_2$$

2. For the vertical acceleration:

$$M\dot{x}_4 = T\cos(x_5 + \varphi) - Mg - b_y x_4$$

which simplifies to:

$$\dot{x}_4 = \frac{T}{M}\cos(x_5 + \varphi) - g - \frac{b_y}{M}x_4$$

3. For the angular acceleration:

$$J\dot{x}_6 = -Tl\sin(\varphi)$$

which simplifies to:

$$\dot{x}_6 = -\frac{Tl}{J}\sin(\varphi)$$

Step 3: Write the State-Space Representation Now we can write the system in state-space form:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = f(\mathbf{x}, u) = \begin{bmatrix} x_2 \\ -\frac{T}{M}\sin(x_5 + \varphi) - \frac{b_x}{M}x_2 \\ x_4 \\ \frac{T}{M}\cos(x_5 + \varphi) - g - \frac{b_y}{M}x_4 \\ x_6 \\ -\frac{Tl}{J}\sin(\varphi) \end{bmatrix}$$

Here:

- T and φ are inputs to the system (thrust and thrust angle).
- g is the gravitational constant.

2.2 Linearization

In order to linearize the model, we first have to find the equilibrium points. In order to do so, we set the derivatives of the state variables to zero to find equilibrium conditions.

Step 1: Find the Equilibrium Points

For an equilibrium point, the system's state derivatives must be zero:

$$\dot{\mathbf{x}} = 0 \Rightarrow f(\mathbf{x}, u) = 0.$$

Setting each equation in $f(\mathbf{x}, u) = 0$ to zero, we get:

1. $\dot{x}_1 = x_2 = 0$: The horizontal velocity must be zero.
2. $\dot{x}_2 = -\frac{T}{M}\sin(x_5 + \varphi) - \frac{b_x}{M}x_2 = 0$: The horizontal acceleration must be zero.

$$\Rightarrow -\frac{T}{M}\sin(x_5 + \varphi) = 0 \Rightarrow T\sin(x_5 + \varphi) = 0.$$

Assuming $T \neq 0$, this implies $x_5 + \varphi = 0$ or $\theta + \varphi = 0$.

3. $\dot{x}_3 = x_4 = 0$: The vertical velocity must be zero.
4. $\dot{x}_4 = \frac{T}{M}\cos(x_5 + \varphi) - g - \frac{b_y}{M}x_4 = 0$: The vertical acceleration must be zero.

$$\Rightarrow \frac{T}{M}\cos(x_5 + \varphi) - g = 0 \Rightarrow T = \frac{Mg}{\cos(x_5 + \varphi)}$$

So, $T = \frac{Mg}{\cos(x_5+\varphi)}$ if $\cos(x_5 + \varphi) \neq 0$ (which is true, following what is said in point 2).

5. $\dot{x}_5 = x_6 = 0$: The angular velocity must be zero.

6. $\dot{x}_6 = -\frac{Tl}{J}\sin(\varphi) = 0$: The angular acceleration must be zero.

$$\Rightarrow T\sin(\varphi) = 0.$$

Assuming $T \neq 0$, this implies $\varphi = 0$.

In summary, equilibrium points occur when:

$$x_2 = x_4 = x_6 = 0, \quad \theta = 0, \quad \varphi = 0, \quad T = Mg.$$

Step 2: Linearize the System Around the Equilibrium

To linearize, we expand $f(\mathbf{x}, u)$ around the equilibrium point $(\mathbf{x}_e, u_e) = \left(\begin{bmatrix} x \\ 0 \\ y \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} Mg \\ 0 \end{bmatrix} \right)$ using a Taylor series expansion and retain only the first-order terms:

$$\dot{\mathbf{x}} \approx A(\mathbf{x} - \mathbf{x}_e) + B(u - u_e)$$

where $A = \frac{\partial f}{\partial \mathbf{x}}\big|_{(\mathbf{x}_e, u_e)}$ and $B = \frac{\partial f}{\partial u}\big|_{(\mathbf{x}_e, u_e)}$.

Compute the Jacobian Matrices A and B

1. **Matrix** A (Jacobian of f with respect to $\mathbf{x}$):

   Since the equations for $f(\mathbf{x}, u)$ are:

$$\dot{\mathbf{x}} = \begin{bmatrix} x_2 \\ -\frac{T}{M}\sin(x_5 + \varphi) - \frac{b_x}{M}x_2 \\ x_4 \\ \frac{T}{M}\cos(x_5 + \varphi) - g - \frac{b_y}{M}x_4 \\ x_6 \\ -\frac{Tl}{J}\sin(\varphi) \end{bmatrix},$$

   we compute the partial derivatives with respect to each state variable $x_1, x_2, \ldots, x_6$ at the equilibrium point, and we get:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{b_x}{M} & 0 & 0 & -\frac{T}{M}\cos(x_5 + \varphi) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{b_y}{M} & -\frac{T}{M}\sin(x_5 + \varphi) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\Bigg|_{\mathbf{x}=\mathbf{x}_{\text{eq}}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{b_x}{M} & 0 & 0 & -\frac{T}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{b_y}{M} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. **Matrix** B (Jacobian of f with respect to u):

   The inputs are T and φ, so we compute the partial derivatives of $f(\mathbf{x}, u)$ with respect to T and φ at the equilibrium.

$$B = \begin{bmatrix} 0 & 0 \\ -\frac{\sin(x_5+\varphi)}{M} & -\frac{T}{M}\cos(x_5+\varphi) \\ 0 & 0 \\ \frac{\cos(x_5+\varphi)}{M} & -\frac{T}{M}\sin(x_5+\varphi) \\ 0 & 0 \\ -\frac{l\sin(\varphi)}{J} & -\frac{Tl\cos(\varphi)}{J} \end{bmatrix}\Bigg|_{\mathbf{x}=\mathbf{x}_{\text{eq}}} = \begin{bmatrix} 0 & 0 \\ 0 & -\frac{T}{M} \\ 0 & 0 \\ \frac{1}{M} & 0 \\ 0 & 0 \\ 0 & -\frac{Tl}{J} \end{bmatrix}$$

After computing these partial derivatives, the system's linearized form around the equilibrium point is:

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu.$$

This approach allows us to analyze the system's behavior near the equilibrium, such as stability, by examining the eigenvalues of the matrix A.

2.3 Analysis of the linearized system

2.3.1 Reachability Analysis

To determine if the system is reachable, we compute the reachability matrix $\mathcal{R}$, defined as:

$$\mathcal{R} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}$$

where:

- A is the state matrix of the linearized system,
- B is the input matrix of the linearized system,
- n is the number of states in the system.

The system is considered **reachable** if the matrix $\mathcal{R}$ has full rank, i.e., $\text{rank}(\mathcal{R}) = n$.
After computation, we obtain:

$$\mathcal{R} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} =$$

$$= \begin{bmatrix} 0 & 0 & 0 & -g & 0 & \frac{b_x g}{M} & 0 & \frac{Mg^2l}{J} - \frac{b_x^2 g}{M^2} & \dots \\ 0 & -g & 0 & \frac{b_x g}{M} & 0 & \frac{Mg^2l}{J} - \frac{b_x^2 g}{M^2} & 0 & \frac{b_x^3 g}{M^3} - \frac{b_x g^2 l}{J} & \dots \\ 0 & 0 & \frac{1}{M} & 0 & -\frac{b_y}{M^2} & 0 & \frac{b_y^2}{M^3} & 0 & \dots \\ \frac{1}{M} & 0 & -\frac{b_y}{M^2} & 0 & \frac{b_y^2}{M^3} & 0 & -\frac{b_y^3}{M^4} & 0 & \dots \\ 0 & 0 & 0 & -\frac{Mgl}{J} & 0 & 0 & 0 & 0 & \dots \\ 0 & -\frac{Mgl}{J} & 0 & 0 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}$$

The rank of the reachability matrix $\mathcal{R}$ is:

$$\text{rank}(\mathcal{R}) = 6$$

Meaning that **the matrix is fully reachable**.

2.3.2 Observability Analysis

To determine if the system is observable, we compute the observability matrix $\mathcal{O}$, defined as:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

where:

- A is the state matrix of the linearized system,
- C is the output matrix of the linearized system,
- n is the number of states in the system.

The system is considered **observable** if the matrix $\mathcal{O}$ has full rank, i.e., $\text{rank}(\mathcal{O}) = n$.
After computation, the observability matrix $\mathcal{O}$ is:

$$\mathcal{O} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -\frac{b_x}{M} & 0 & 0 & -g & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

The rank of $\mathcal{O}$ is:

$$\text{rank}(\mathcal{O}) = 6$$

Since the rank of the observability matrix equals the number of states $n = 6$, the system is **observable**.

2.3.3 Stability analysis

The Jordan form of the matrix A is:

$$J = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{b_x}{M} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{b_y}{M} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

It is possible to see that there are 4 Jordan blocks:

- J_0^3

- $J^1_{-\frac{b_x}{M}}$
- $J^1_{-\frac{b_y}{M}}$
- J^1_0

We can notice that there are two negative eigenvalues (associated with stable states), but one of the jordan block associated with the 0-eigenvalue have geometric multiplicity 3, meaning that the system is not stable.
NOTE: Specifically, the stable states are $\dot{\mathbf{x}}$ and $\dot{\mathbf{y}}$ as, in the presence of the air viscosity, they decrease exponentially.

3 Ascent Phase

3.1 Non-linear Open-loop System

A non-linear system has been implemented in MATLAB© to simulate the dynamic of the system.

Below is the MATLAB© code (to see it, open the simulink file ***"thrust_vector_rocket2_no_thrust.slx"***).

```
function out = non_linear_dynamic_simplified(in)
% This script implements the non-linear dynamic of the system, considering as the
% only input phi, since thrust T is fixed and not controllable.
x1 = in(1); % x
x2 = in(2); % x'
x3 = in(3); % y
x4 = in(4); % y'
x5 = in(5); % angle
x6 = in(6); % angular velocity
u = in(7);  % angle of the thrust, phi

load('rocketParameters.mat');

dx1 = x2;
dx2 = -T/M*sin(x5 + u) - bx/M*x2;
dx3 = x4;
dx4 = T/M*cos(x5 + u) - g - by/M*x4;
dx5 = x6;
dx6 = -T*L/J*sin(u);

out = [dx1; dx2; dx3; dx4; dx5; dx6];
end
```

This function has been tested on some basic scenarios to verify that it was correctly working. Specifically, it was checked on some basic cases that the range and the maximum height were coherent with the cynematics formulas:

$$h_{max} = \frac{1}{2} a t_{sim}^2$$

$$\Delta x = v_x t + \frac{1}{2}\frac{F_x}{M}t^2$$

3.2 PID Control for Vertical Stabilization

During the climbing phase, the primary objective is to maintain the rocket in a vertical orientation. To achieve this, a PID controller was implemented and tuned using Simulink.

The tuning process was carried out with Simulink's PID Tuner tool, and the controller was tested on both the linearized and nonlinear systems.

Figure 2 illustrates the implemented PID control loop in Simulink. The target angle, θ_{target}, is set to $0°$ (i.e., perfectly vertical), while the initial condition assumes a stationary rocket inclined at $\theta = 5°$.

As shown in Figure 3, the PID controller successfully stabilizes the rocket, bringing it back to the desired vertical position.

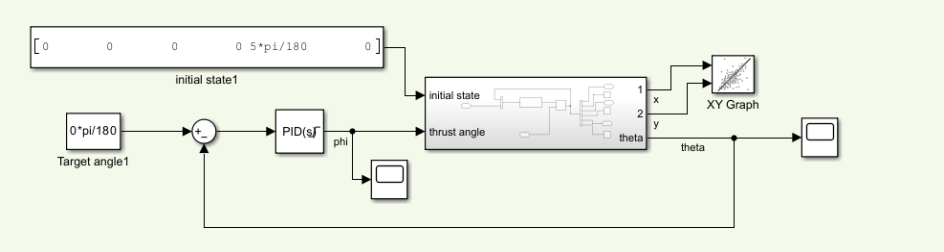


Figure 2: PID loop for vertical stabilization

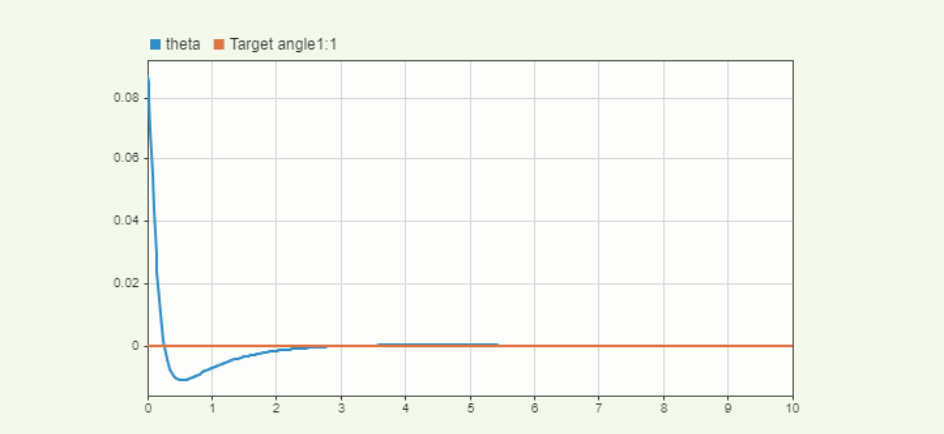


Figure 3: Result of the PID control

3.3 Regulator with Pole Placement

In the course *Control Theory* we studied that it is possible to stabilize a system even without a dynamic controller. Under the hyphothesis of full reachability, it is possible to find a static feedback gain that allows to put the poles of the system at desired locations, using either the Bass-Gura or the Ackermann equations.

At first glance, this approach seems inapplicable to our system, as it is not fully reachable. However, if we sacrify the desire of controlling the y and y' variables, the state-space $[x, x', \theta, \theta']$ becomes fully reachable and this enables the design of a controller for this subsystem.
In practice, this means that we are controlling the rocket in order to maintain a vertical position and ascent stricly along a vertical tracjectory (i.e., eliminating lateral drift during the ascent phase).
The following MATLAB code selects the reachable subsystem and designs a static feedback controller that places all the poles in the left half-plane with negative real parts.

```
reachable = [1,2,5,6]; % Reachable states: [x, x', theta, theta']
Ar = A(reachable, reachable)
Br = B(reachable)
Kr = place(Ar, Br, [-1,-2, -3, -4]')     % Place the poles in [-1,-2,-3,-4]
K = [Kr([1,2]), 0, 0, Kr([3,4])]
A1 = A - B*K
reg = ss(A1, sys.b, sys.c, 0)
```

This controller was implemented and tested in Simulink, as shown in Figure 4. The initial conditions were set to $x = 1m$, $x' = 1m/s$ and $\theta = 5°$. The results demonstrate that the controller can successfully drive those variables to zero, stabilizing the system.

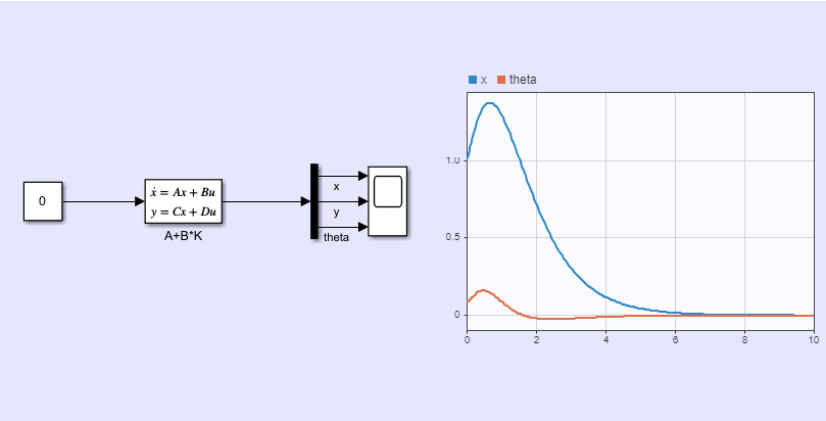


Figure 4: Controller with constant feedback gain

3.4 Optimal Control

Optimal control is a technique to find a control input sequence that minimizes a cost function while ensuring the system reaches the desired state. Since this minimization problem involves the use of the complex calculus of variations, two approximations are used to sensibly simplify the problem:

1. Parametrization of U
2. Discretization of the system.

The problem becomes:

$$\begin{cases} \min\limits_{\mathbf{U}_d} ||\mathbf{U}_d|| \\ \text{subject to:} \\ \quad \mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k \\ \quad x_0 = \overline{x0} \\ \quad x_N = \overline{x_T} \end{cases}$$

And this is now easy to solve. Indeed, using some linear algebra, we can obtain that:

$$x_T = A_d^N \cdot x_0 + R_{d,n} \cdot U_{d_F} \tag{1}$$

where:

- A_d, B_d are the discretized matrices of the system
- $R_{d,N}$ is the reachability matrix after N steps:

$$\mathcal{R} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}$$

- U_{d_F} is the vector containing all the inputs, flipped (i.e., it starts with the last input and ends with the first input):

$$U_{d_F} = \begin{bmatrix} u_{N-1} & u_{N-2} & \dots & u_1 & u_0 \end{bmatrix}^T$$

The solution to the equation 1 is:

$$U_{d_F} = (R_{d,n})^{-1} \cdot (x_T - Ad^N \cdot x_0) \tag{2}$$

In order to apply this technique, the system must be fully reachable. Therefore, we restrict the analysis to the reachable system $[x, x', \theta, \theta']$ as done in the previous section.
The Optimal Control procedure can be summarized as follows:

1. **System Discretization:** The continuous-time system is discretized using the Zero-Order Hold (ZOH) method with a sampling time $T_s = 0.05$ seconds. This results in discrete-time matrices $\mathbf{A}_d$ and $\mathbf{B}_d$.
2. **Reachability Check:** When transitioning from continuous-time to discrete-time, reachability is not always preserved. It is necessary to check that the discretized system remains reachable.
3. **Control horizon** N: This parameter determines the number of steps over which the system is controlled. Increasing N decreases control costs but comes at the cost of slower convergence to the desired state.
4. **Reachability Matrix:** A reachability matrix $\mathbf{R}_{d,N}$ is constructed over N time steps to compute the control inputs that drive the system to the desired state $\mathbf{x}_T$ in N steps.
5. **Control Input Calculation:** The control inputs $\mathbf{u}$ are calculated by solving Equation 2. This provides a sequence of control actions for both thrust and thrust angle.

6. **Final State Adjustment:** The code adjusts the control inputs to ensure the derivative of the state is zero at the final state. This way, after the end of the control, the rocket will remain stable in its last state $\mathbf{x}_T$. To do so, we solve the equation

$$x_T = A_d \cdot x_T + B_d \cdot u_{eq} \quad \Rightarrow \quad u_{eq} = (B_d)^{-1} \cdot (I - A_d) \cdot x_T$$

Below is the MATLAB code to do so.

```
%% OPTIMAL CONTROL
Ts = 0.05;
N = 60; % we are working at 20Hz --> N = 60 means 3 seconds.

sysD = c2d(sys, Ts, 'zoh'); % system discretization using ZOH
Ad = sysD.a;
Bd = sysD.b;
Cd = sysD.c;
Dd = sysD.d;

% Controllability Check
Rd = ctrb(Ad,Bd);
disp(rank(Rd));

% Reachability matrix
Rn = Bd;
for i = 2:N
    Rn = [Bd Ad*Rn];
end

% Control Input Calculation
u = pinv(Rn)*(xf - Ad^N*x0);

% Final State Adjustment
% xf = Ad*xf + Bd*uf --> uf = pinv(Bd)*(I-Ad)*xf
uf = pinv(Bd)*(eye(6)-Ad)*xf;
u1 = [uf'; u];

% Check of the feasibility of the control: the control should be between -5 and +5 degrees
if(min(u1) < -5 || max(u1) > 5)
    disp("Control over the limit!")
else
    disp("The optimal control is feasable")
end
```

Figures 5 and 6 illustrate the results of applying optimal control to both the linear and non-linear systems. While the control performs well in both cases, in the non-linear scenario, even slight nonlinearities cause the system to diverge once the control input is terminated.

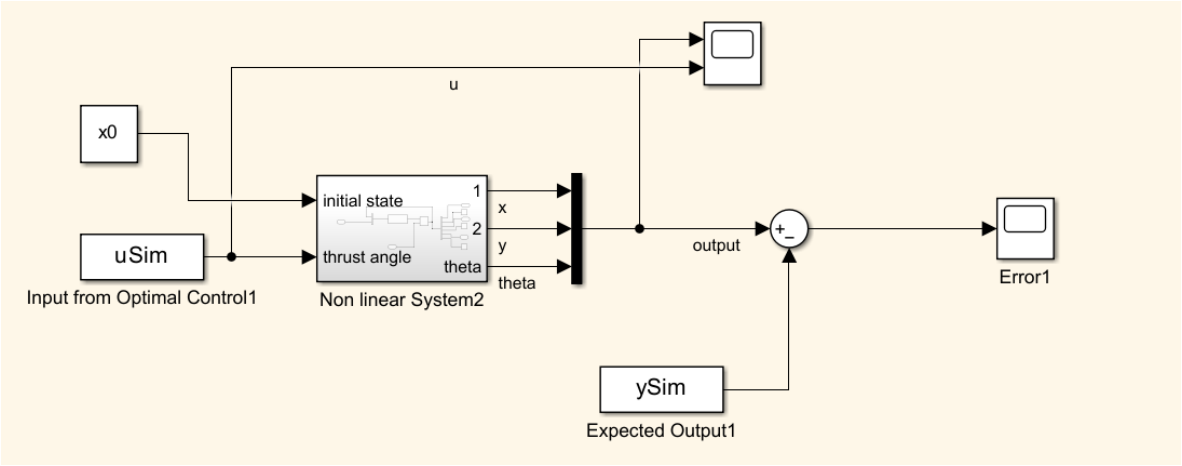
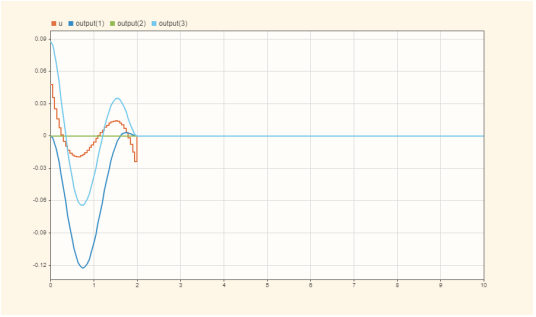
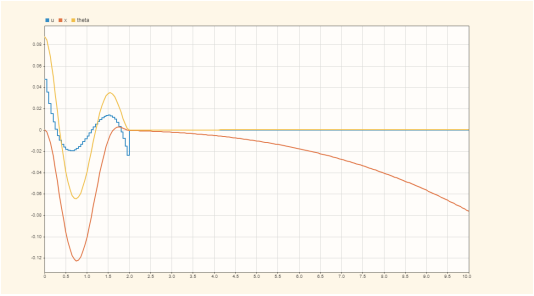


Figure 5: Optimal Control applied to the non-linear system



(a) Linear system



(b) Non-linear system

Figure 6: Results of the optimal control

3.5 Model Predictive Control (MPC)

Model Predictive Control (MPC) is an advanced control strategy that solves an optimization problem in real time to determine the optimal control inputs. At each time step, MPC predicts the future behavior of the system over a finite prediction horizon, using a mathematical model of the system dynamics.

The key steps of MPC are:

- Solve an optimization problem to minimize a cost function, typically balancing performance objectives (e.g., tracking a desired trajectory) and input constraints (e.g., actuator limits). In our case, we want to bring the rocket to the target position $\mathbf{x}_T$
- Apply only the first control input from the optimized sequence to the system.
- Recalculate the optimal control sequence at the next time step, based on updated system states and measurements.

MPC offers the advantage of explicitly handling constraints on states and inputs, while accounting for the dynamic behavior of the system. This makes it particularly well-suited for complex, multivariable systems like rockets, where precise control is critical under constraints such as limited thrust and

angular limits.

We implemented the MPC using the MPC block in *Simulink* to maintain a vertical trajectory. The rocket would start still with an inclination of 5°. Figure 7 shows the control loop. The result of the simulation was used to create an animation (see MATLAB file ***"model_2_animation_MPC.m"***). Figure 8 shows an image of the result, while a video of the animation can be found here.

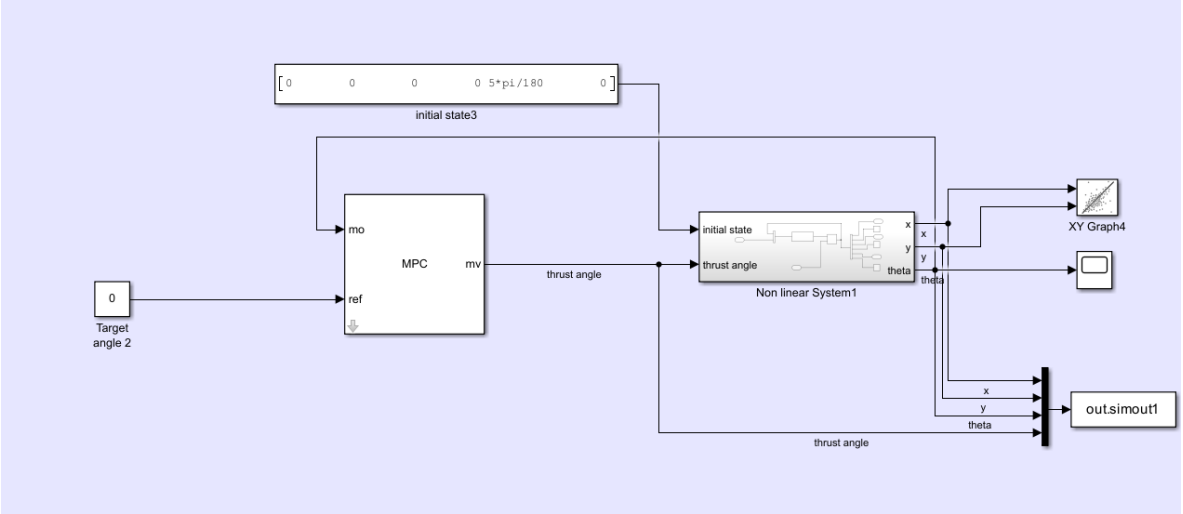


Figure 7: Simulink loop for MPC

3.6 Disturbancies rejection with MPC

To simulate a real-world situation, random disturbances have been added to the system. The result has been tested and compared with the ideal situation (no distubrances). The new Simulink loop is displayed in Figure 9, while the results of the animation can be seen in Figure 10. The video of the animation can be found here.

It can be observed that the rocket successfully ascends in a controlled manner despite the disturbances. However, the ascent is less efficient, as the maximum altitude reached in 10 seconds is 16m, compared to 20m in the disturbance-free scenario.

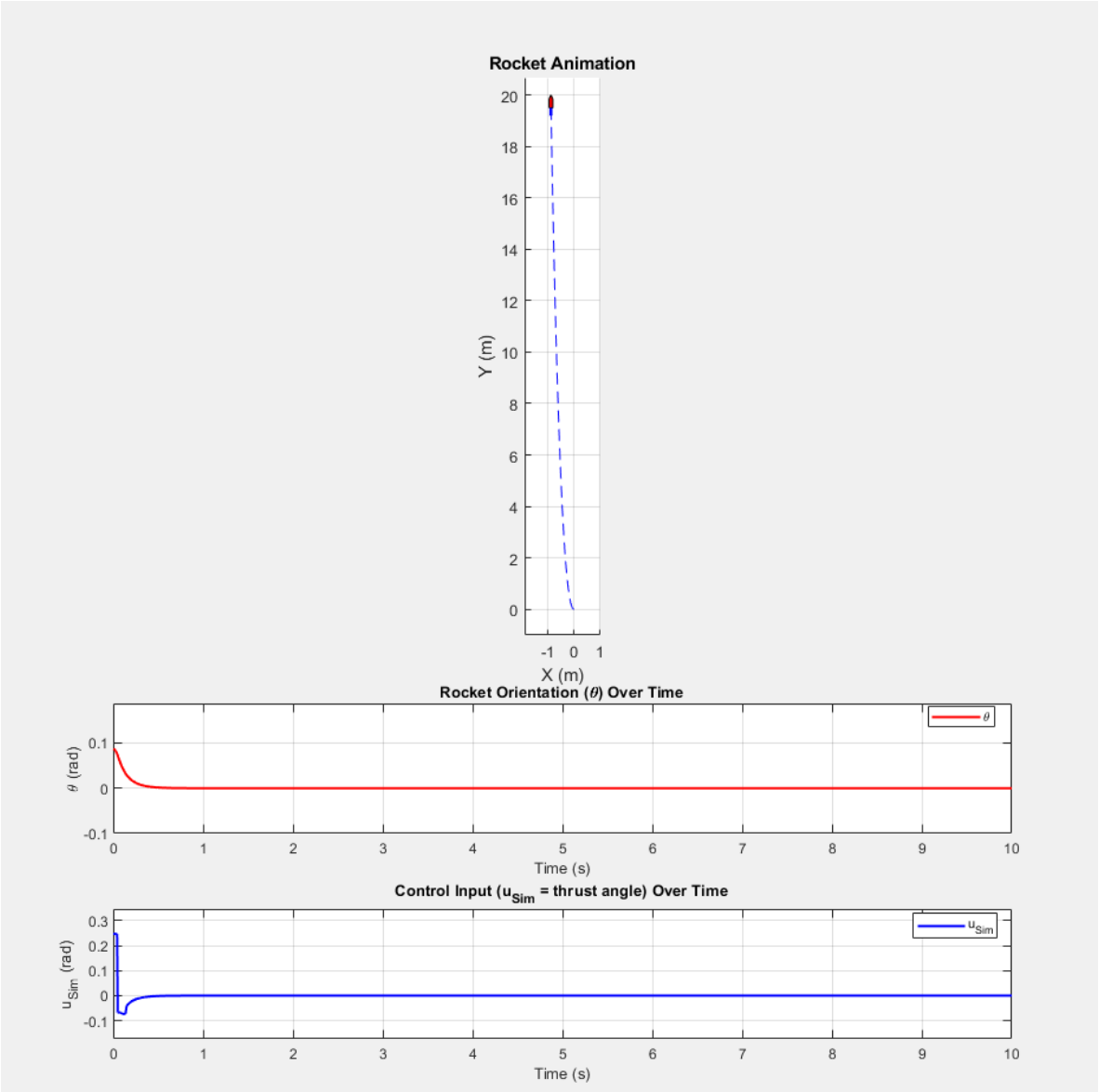


Figure 8: Animation of the MPC control. See the video

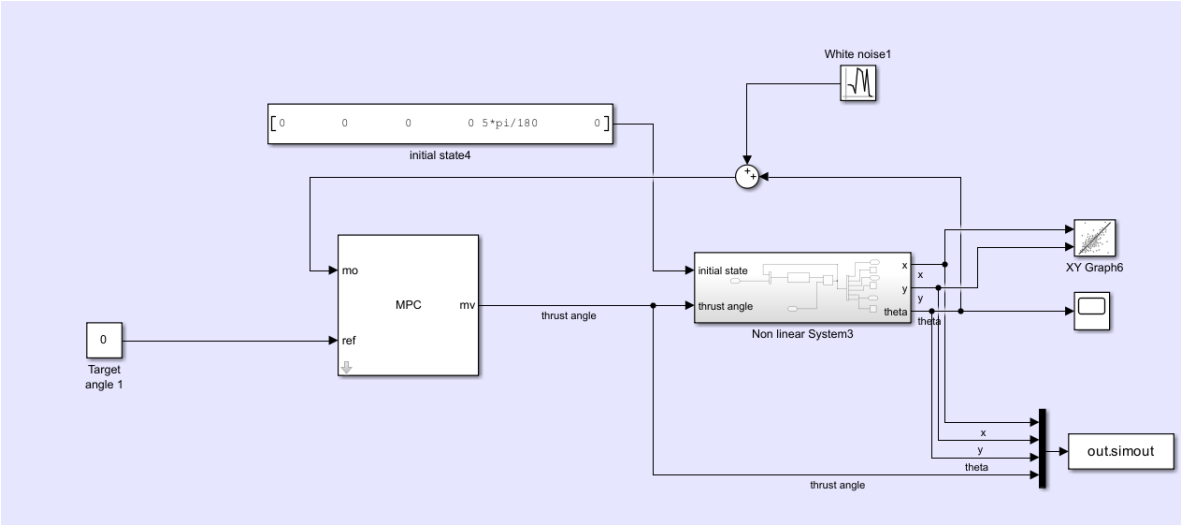


Figure 9: Simulink loop for MPC — with disturbances.

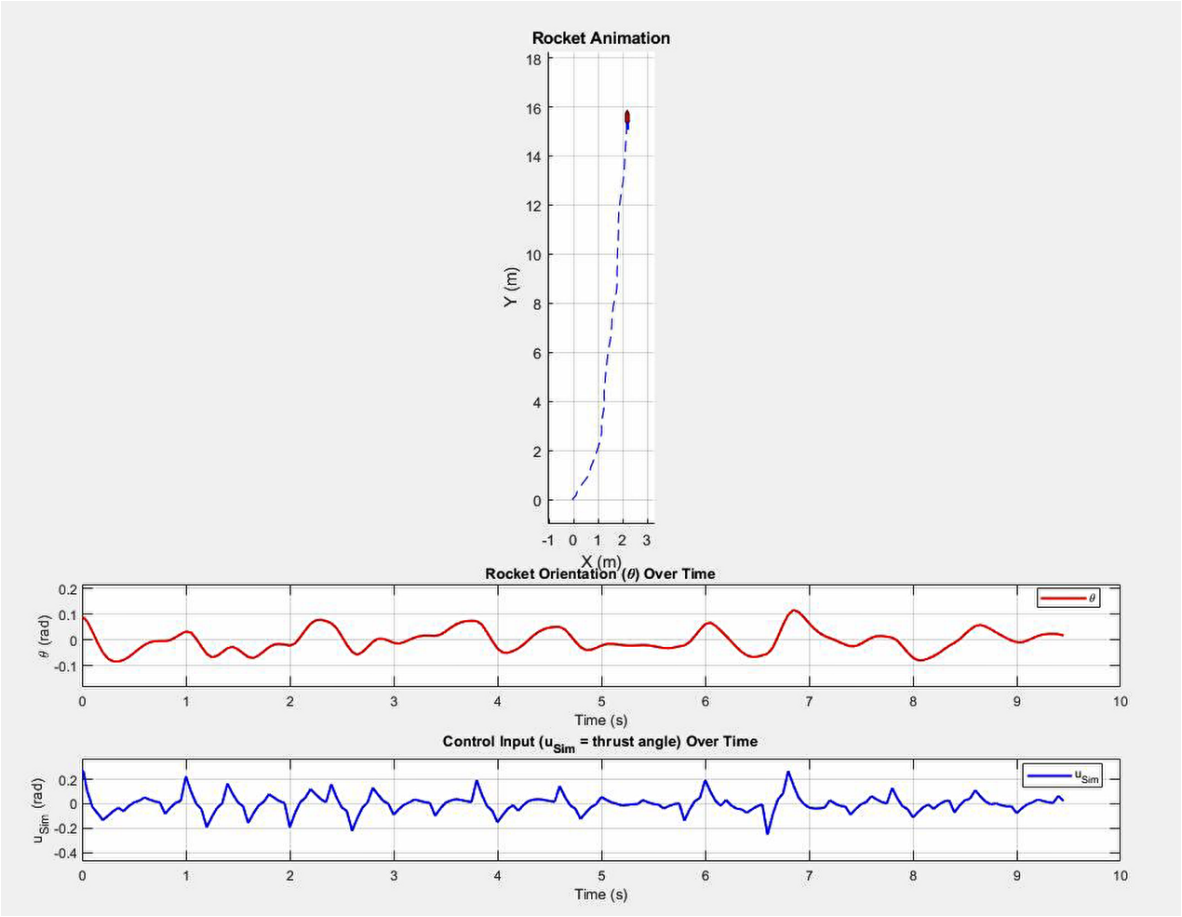


Figure 10: Animation of the MPC control, in the presence of **noise**. See the full video

4 Hovering Phase

The second control application for this rocket is **hovering**. In this mode, the rocket behaves similarly to a quadcopter drone, enabling precise and controlled movement between various points in space. This type of control is particularly useful for performing advanced maneuvers, such as landing or reaching specific target locations.
During the hovering phase, the thrust is no longer fixed at maximum intensity and can instead be modulated to control the rocket's altitude. As shown in the system analysis, introducing the throttle as an additional input makes the system fully reachable.
In the following sections, we will explore several control strategies designed to achieve the objective of guiding the rocket to a desired target in space.

4.1 Optimal Control

This technique has already been covered in paragraph 3.4. Let's now apply it to the new, more complex system.

We set the starting state to $x_0 = [0; 0; 0; 0; 5°; 0]$ and the final to $x_f = [0; 0; 2; 0; 0; 0]$. Figure 11 shows that the rocket successfully converged to the desired target state. Figure 12 shows the trajectory followed by the rocket during the transition from x_0 to x_T. An animation of the transition can be found here.

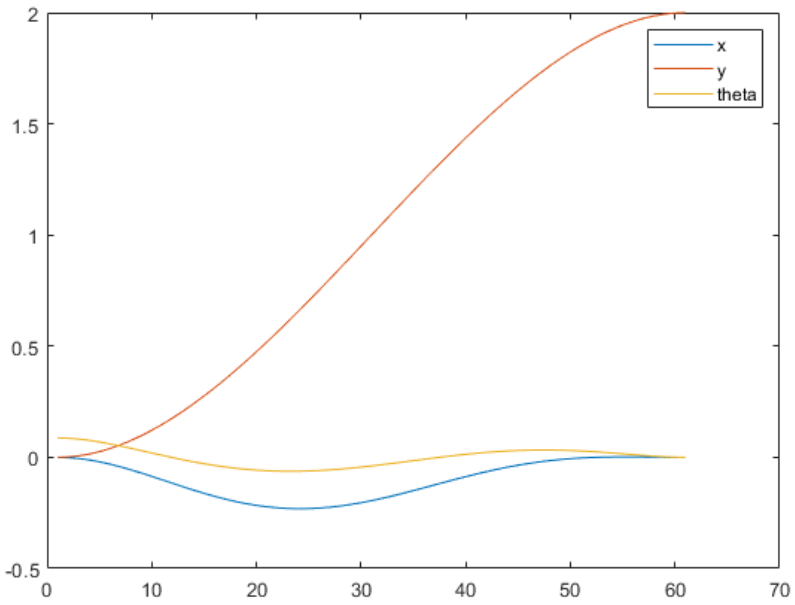


Figure 11: Result of the optimal control

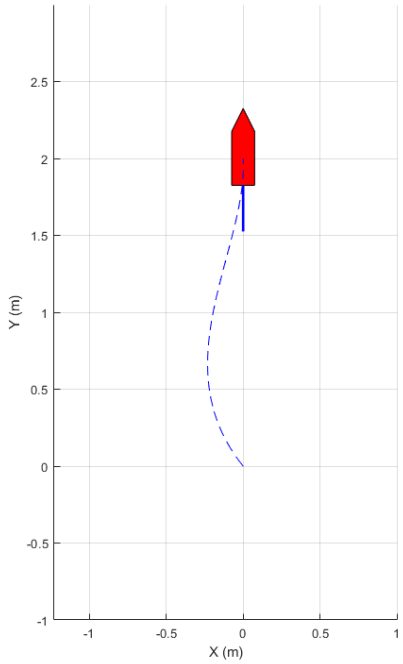


Figure 12: Result of the optimal control - trajectory. See animation here

4.2 Optimal control with condition on the input

When implementing a physical system, we always have some limits on the input variables. Therefore, we have to add some constraints to our problem. As a result, the problem becomes:

$$\begin{cases} \min\limits_{\mathbf{U}_d} ||\mathbf{U}_d|| \\ \text{subject to:} \\ \quad \mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k \\ \quad x_0 = \overline{x0} \\ \quad x_N = \overline{x_T} \\ \quad u_{min} \leq u_k \leq u_{max} \quad \forall k = 1, 2, ..., N \end{cases}$$

To solve this problem, we use the MATLAB function ***lsqlin***.
However, in order to apply it, we have to be careful about the fact that the function will work with a 1D vector $\mathbf{u}$ with $2N$ elements, in the form:

$$\mathbf{u} = [u_1(t=1); u_2(1); u_1(2); u_2(2), ...]$$

So, after the computation of the solution, we will have to reshape it in order to get the shape that we prefer:

$$\mathbf{u} = \begin{bmatrix} u_1(t=1) & u_2(t=1) \\ u_1(t=2) & u_2(t=2) \\ \vdots & \vdots \end{bmatrix}$$

Below is the MATLAB implementation.

```
N = 60;
C_lsqlin = eye(2*N);
d_lsqlin = zeros(2*N,1);
Rn = Bd;
for i = 2:N
    Rn = [Bd Ad*Rn];
end
Aeq = Rn;
beq = xf - Ad^p*x0;
% lower and upper bounds
bound_thrust = M*g;
bound_angle = 5;
lb =-bound_angle*ones(2*N,1);
lb(1:2:end) = -bound_thrust;
ub = bound_angle*ones(2*N,1);
ub(1:2:end) = bound_thrust;

[u_lsqlin,~,~,exitflag1] = lsqlin(C_lsqlin,d_lsqlin,[],[],Aeq,beq,lb,ub);

% Reshape the vector u
u_lsqlin = reshape(u_lsqlin, 2, []).';
```

By comparing this result on MATLAB with the previous one, we saw that the two methods provide roughly the same vector (with a little difference due to finite-precision variable imprecisions: indeed the norm of the error is around 1e-13).

4.3 Optimal control, minimizing the time

If we now want to check the minimum time to get the rocket to the target position while respecting the constraints on $\mathbf{u}$, we can make a cycle that tries to find a solution to the minimization function using *lsqlin* starting from $N = 0$ and stopping as soon as a solution is feasible.

An implementation of this code is shown here:

```
exitflag = 0;
N = 1;
Rn = Bd;
while (exitflag <= 0)
    N = N+1;
    fprintf('processing N = %d\n', N);
    Rn = [Bd Ad*Rn];
    C_lsqlin = eye(2*N);
    d_lsqlin = zeros(2*N,1);
    Aeq = Rn;
    beq = xf - Ad^p*x0;
```

```
    % lower and upper bounds
    bound_thrust = M*g;
    bound_angle = 5;
    lb =-bound_angle*ones(2*N,1);
    lb(1:2:end) = -bound_thrust;
    ub = bound_angle*ones(2*N,1);
    ub(1:2:end) = bound_thrust;
    % solution with lsqlin
    [u_lsqlin,~,~,exitflag] = lsqlin(C_lsqlin,d_lsqlin,[],[],Aeq,beq,lb,ub);
    fprintf('exitflag = %d\n', exitflag);
end

u_lsqlin = reshape(u_lsqlin, 2, []).';
```

This code found that the minimum N is **19**. The solution is depicted in Figure 13 and the trajectory of the rocket in Figure 14. An animation of the trajectory can be found here.

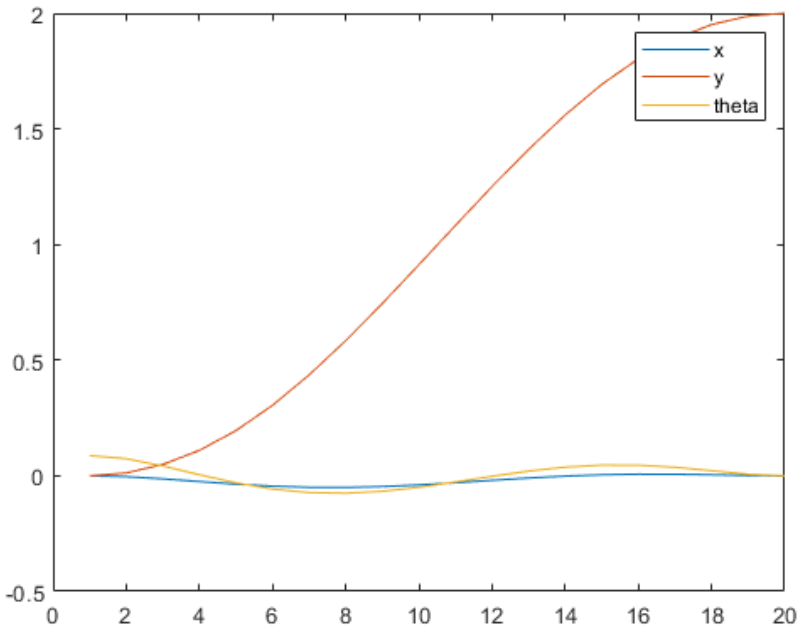


Figure 13: Result of the optimal control, minimum time (19 steps)

4.4 Decoupling

A PID can be applied only when a system is MIMO. If the system is not MIMO, the only way to control it with some PIDs is to decouple it. In our system, we can do it partially by rearranging the state variables.

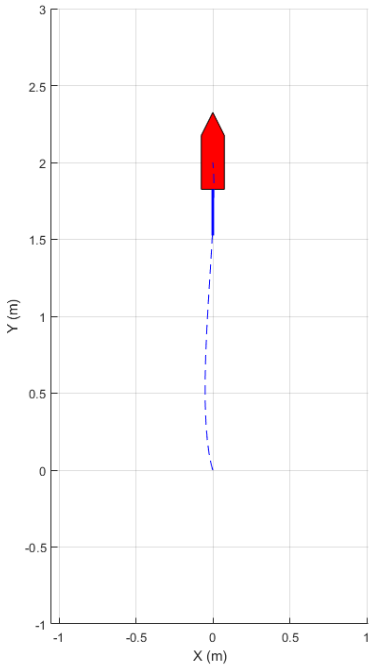


Figure 14: Result of the optimal control, minimum time (19 steps). See animation here.

4.4.1 Rearranging State Variables

To change the state variable order from:

$$x = \begin{bmatrix} x & x' & y & y' & \theta & \theta' \end{bmatrix}^\top$$

to:

$$x_{\text{new}} = \begin{bmatrix} y & y' & x & x' & \theta & \theta' \end{bmatrix}^\top,$$

we use a **permutation matrix** P, defined as:

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

This matrix reorders the state variables by swapping their positions. To rearrange the state variables, we compute the new system matrices as follows:

1. **State matrix** A_{new}:

$$A_{\text{new}} = PAP^\top = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -9.81 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. **Input matrix** B_{new}:

$$B_{\text{new}} = PB = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & -1 \end{bmatrix}$$

3. **Output matrix** C_{new}:

$$C_{\text{new}} = CP^\top = C_{\text{new}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

4. **Feedthrough matrix** D_{new} remains unchanged:

$$D_{\text{new}} = D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Rearranging the state variables results in a new state-space representation where the structure of $\tilde{A}$ and $\tilde{B}$ is as follows:

$$\tilde{A} = \begin{bmatrix} A_y & 0 \\ 0 & A_{x\theta} \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} B_y \\ B_{x\theta} \end{bmatrix}$$

where:

- A_y corresponds to the decoupled y, y' dynamics, which can be controlled with the input T.
- $A_{x\theta}$ corresponds to the coupled x, x', θ, θ' dynamics, which can be controlled with the input ϕ.

This block-diagonal structure shows the partial decoupling of the system: the y variable is decoupled from the x and θ variables. So, we can control y with T using, for example, a PID. However, since we have only one input to control the block $A_{x\theta}$, it is not possible to control them both with a PID. For this section, it is necessary to use an other controller.

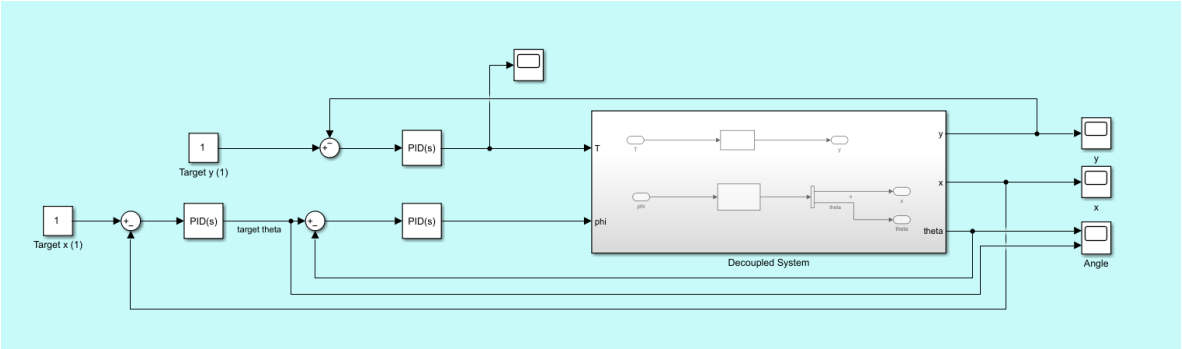


Figure 15: Decoupled system with double feedback loop.

4.4.2 Controlling the decoupled system

It is now possible to control the two subsystems using two controllers that operate independently from each other.
First subsystem (A_y, B_y)**:** This system is SISO. Therefore, a normal PID is enough to control it.
Second subsystem $(A_{x\theta}, B_{x\theta})$: this subsystem is trickier, as we have only one input to control two outputs. A good strategy is therefore to apply a cascade control.

Cascade control is a hierarchical control strategy where two controllers are used: a fast inner loop that stabilizes a critical variable and a slower outer loop that controls the overall system behavior. The output of the outer loop acts as the setpoint for the inner loop. This approach improves system performance by handling fast dynamics separately from slower ones.

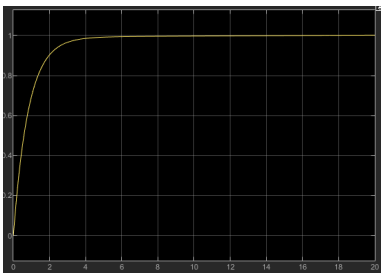
In this project, a **cascade control strategy** is applied to stabilize a rocket with one input (ϕ, thrust deflection angle) and two outputs (θ, rocket rotation angle, and x, horizontal position):

- **Inner Loop (Fast Control):** A PID controller regulates the rocket's orientation θ by adjusting the thrust deflection angle ϕ. This loop operates at a higher speed to ensure rapid stabilization of the rocket's tilt.
- **Outer Loop (Slow Control):** A second, slower PID controller regulates the horizontal position x. This controller generates a setpoint for the inner loop (i.e., θ_{target}) based on the position error, ensuring the rocket moves towards the desired location.

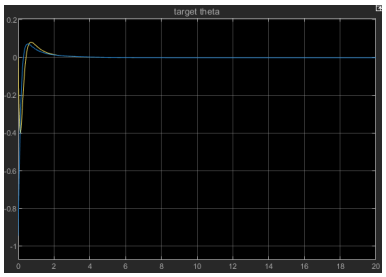
Figure 15 shows the two feedback loops to control the system, while Figure 16 presents the results of the implemented feedback loops:

- The first graph illustrates the feedback loop controlling the output y.
- The second graph depicts the feedback loop for the dynamics of θ; it can be seen that the response is very fast, with θ closely tracking its target.
- The third graph shows the slower feedback loop governing the dynamics of x, demonstrating the gradual correction of the horizontal position.

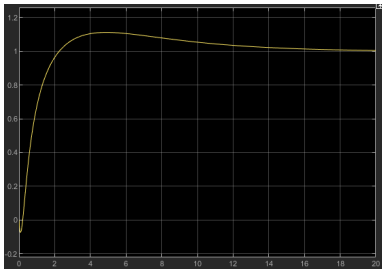
For what concerns the first feedback loop (the one for the variable y)



(a) Feedback loop for y (decoupled from the other 2 variables)



(b) Feedback loop for θ (fast dynamic)



(c) Feedback loop for x (slow dynamic)

Figure 16: Results of the feedback loops: (a) y, (b) θ, and (c) x.

5 Physical Implementation

After designing the controller, a physical rocket prototype was built to serve as a proof of concept. The goal was to develop a 2-DoF thrust vectoring rocket capable of stabilizing itself when attached to a test bench. Several design choices had to be made, the most critical being the selection of the propulsion system.

5.1 Propulsion

While a combustion-based propulsion system would have been the most exciting option, it was not practical due to several factors, including safety concerns, high costs, and regulatory restrictions. For these reasons, an electric propulsion system was chosen. The thrust system consists of a single brushless motor equipped with an 8x3.7 three-blade propeller. This motor is mounted on a 2-DoF Cardan joint, actuated by two servomotors, as shown in Figure 17.

One major drawback of this design is that it cannot operate independently from the test bench. The motor generates a rotational torque along the rocket's main axis, causing it to spin uncontrollably. While this effect is cancelled by the test bench, it would lead to uncontrolled spinning if the rocket were to be flown freely.

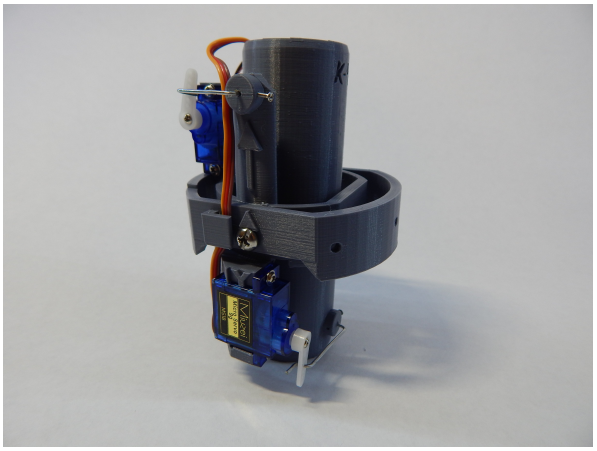


Figure 17: Actuated gimbal for thrust vectoring.

5.2 ESC and Battery

The ESC had to be chosen to match the motor, ensuring it could supply sufficient current without overheating. Since the motor had a maximum current draw of 35A, a 50A ESC was selected.

Regarding the power source, a 4S LiPo battery was chosen to supply the necessary power. This battery powers the ESC, which, in turn, drives the brushless motor through three-phase cables. The ESC also includes a 5V 3A BEC, which provides power to the servomotors.

5.3 Control Board

An Arduino Nano was selected as the main controller. To obtain the rocket's orientation, an IMU was required, and the MPU6050 was chosen. Additionally, a few extra features were integrated:

- An RC receiver to control the rocket's thrust.
- A safety switch to enhance operational safety: to activate the propeller, the user must press the button and wait at least three seconds.

The board is powered by a 9V alkaline battery, which is connected to the V_{in} pin of the Arduino. The ground of this battery is shorted to the ground of the main 4S LiPo battery that powers the motors.

The 5V output from the ESC's BEC was not used to power the Arduino because, under high current draw from the servomotors, the voltage would occasionally drop for a fraction of a second. This caused the Arduino to shut down and restart.

A possible solution would have been to place a capacitor in parallel with the BEC. However, I preferred a separate power supply for the control board, allowing it to operate independently from the motor. This setup was particularly useful for safety reasons, as it enabled software debugging without the risk of the propeller unexpectedly starting.

All components were soldered onto a perfboard. Figure 18 shows an image of the control board.

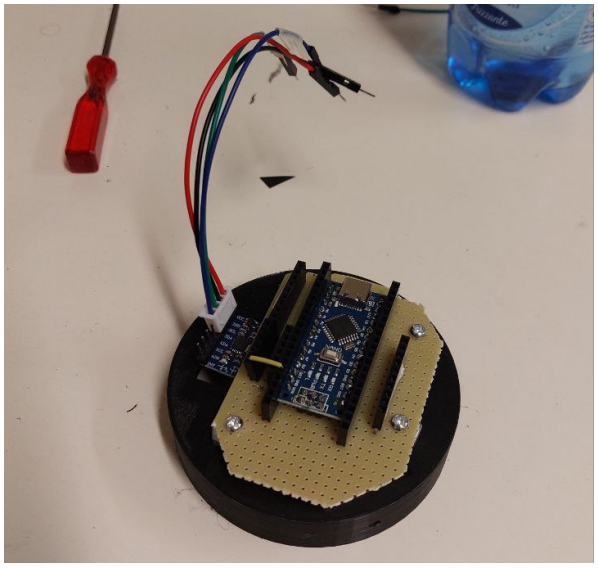


Figure 18: Control Board of the rocket

5.4 Parameter Evaluation

Once the rocket was assembled, it was necessary to determine its key parameters for integration into MATLAB© and PID tuning.

Tuning the PID in MATLAB© enabled a faster iterative process while allowing simulations of various scenarios (e.g., disturbance rejection, stability regions, etc.). Specifically, the following variables had to be measured:

1. Mass
2. Maximum thrust
3. Position of the center of mass
4. Distance from the center of mass to the thrust application point
5. Moment of inertia

5.4.1 Mass

The mass was measured using a scale, as shown in Figure 19. The measured mass was $993g$.

Figure 19: Measurement of the mass

5.4.2 Maximum Thrust

The maximum thrust generated by the propulsion system was measured using a scale, as shown in Figure 20. The recorded maximum thrust was $2.2kgf$.

5.4.3 Center of Mass

The center of mass was determined by balancing the rocket on a straight edge. The point at which the rocket remained stable was recorded as the center of mass.

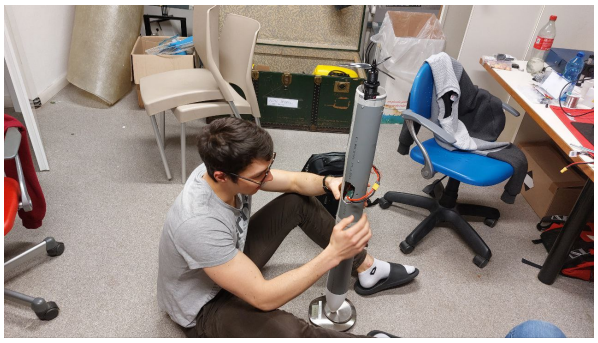


Figure 20: Maximum thrust measurement

5.4.4 Moment of Inertia

After researching different methods for computing the moment of inertia, I decided to perform a bi-filar pendulum test, following the approach demonstrated in this YouTube video by BPS.Space.

Two strings were attached 20 cm ahead and behind the center of mass, suspending the rocket from a table. The strings were positioned vertically, ensuring that the rocket remained parallel to the ground.

The distance $\boldsymbol{d}$ from the table to the center of mass—calculated as the length of the strings plus the rocket's radius—was measured to be 0.65 m. The rocket was then set into small oscillations about its center of mass, and the oscillation period $\boldsymbol{T}$ was measured to be 1.6 seconds.

The moment of inertia was then computed using the following formula:

$$I = \frac{MgT^2l^2}{4\pi^2 d} = 0.088kg \cdot m^2 \tag{3}$$

6 Results

After tuning the PID, I tested the rocket using a bench test. The procedure involved calibrating the rocket in a vertical position and then applying an external disturbance to alter its angle. The goal was to evaluate whether the rocket could autonomously return to its initial vertical position.

However, at the time of writing, the PID controller has not yet been fully optimized, as the rocket still exhibits divergent behavior. Further tuning and refinements are necessary to achieve stable control.

7 Future Work

The primary objective is to achieve stability during the bench test.

Also, a model in Processing has already been implemented to visualize the rocket's orientation. In future developments, this script will be integrated with the controller to receive real-time data from the IMU sensor. This will allow for direct observation of the orientation perceived by the rocket, helping to identify potential sensor-related issues, such as drift errors over time—an issue commonly associated with gyroscopes.

Once stability is achieved, the next step will be to transition to a combustion-based propulsion system and attempt the first (hopefully successful) untethered flight.