Stabilizing a Landing Rocket

***Abstract*—In this project, we develop a simulation framework for a landing rocket in a two-dimensional environment, inspired by SpaceX's reusable rocket technology. The problem is formulated using Newtonian mechanics to model the dynamics of the rocket. To ensure a safe and controlled landing, Model Predictive Control (MPC) is employed as the core control strategy. The MPC framework incorporates numerical optimization, using CasADi [1] to solve the optimization problem and generate optimal control inputs over the horizon. The proposed approach demonstrates the effectiveness of numerical optimization in guiding the rocket to land safely on the ground, showcasing its potential for real-world applications.**

***Index Terms*—MPC (Model Predictive Control), CasADi, Numerical Optimization, Reusable Rocket, RK4 (Runge-Kutta 4), Interior Point Method (IPOPT).**

I. Introduction

Reusable rocket technology has transformed the aerospace industry, with SpaceX pioneering cost-efficient space missions. By reusing Falcon 9 rockets, SpaceX achieves savings of approximately $18 million per launch, significantly reducing operational costs [2]. This advancement underscores the economic and environmental benefits of reusable launch systems.

This project aims to ensure a safe rocket landing by minimizing descent velocity and correcting orientation. The system utilizes the Model Predictive Control (MPC) algorithm to calculate the most effective control inputs, which serve to *stabilize* the rocket by counteracting the increasing downward speed and deviations from a vertical orientation, ultimately achieving a controlled and stable touchdown.

In this paper, we first present the Newtonian formulation of the system dynamics, followed by the definition of constraints. We then introduce the optimization problem, and finally we analyze the simulation results and discuss the system's behavior.

II. Modeling and System Identification

Accurate modeling and system identification are essential for designing an effective control strategy [3] for the rocket landing. This section outlines the dynamics governing the rocket's motion, formulates the equations of motion based on Newtonian mechanics, and describes the process of representing the system in state space. The simplifications and assumptions made to focus on the core problem are also discussed.

As illustrated in Fig.1, the reference coordinate system is defined as shown. The counterclockwise (CCW) direction is considered positive, while the clockwise (CW) direction is negative for both the rocket and the nozzle. Additionally, the reference angle is measured from the north, which differs from

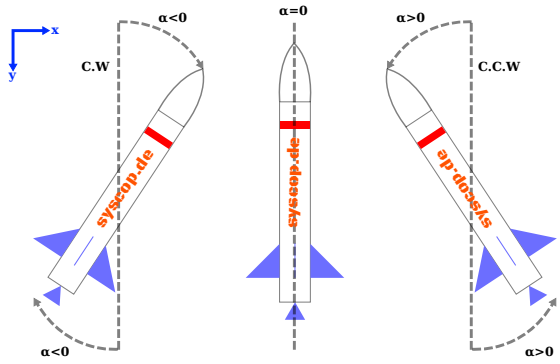


Fig. 1: States the rocket body orientation w.r.t the world coordinate system.

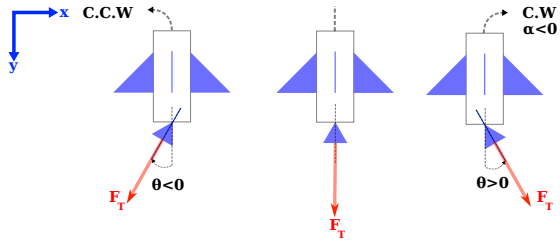


Fig. 2: States the nozzle angle and the respective force w.r.t the rocket's body coordinate system.

the standard mathematical convention, where the angles are typically measured from the positive x-axis.

It can be observed that when the nozzle angle rotates in the positive direction, it generates a torque in the negative direction in the rocket body, and vice versa. This inverse relationship between nozzle rotation and the resulting torque influences the rocket's orientation control as shown in Fig.2.

A. Assumptions

To simplify the problem while preserving its core objectives, the following assumptions were made:

1) Neglecting Aerodynamic Drag: The effects of air resistance were disregarded to focus on the primary forces governing the rocket's motion.
2) Rigid Body Approximation: The rocket is assumed to remain rigid throughout its descent, with no deformation affecting its dynamics.
3) Constant Mass: The deduction of fuel weight as the engine consumes fuel was not accounted for, assuming the rocket's mass remains constant during the simulation.
4) Exclusion of Environmental Forces: External environmental factors such as wind and turbulence were ne-

glected, assuming the rocket descends in a stable and controlled environment.

These assumptions provide a tractable framework for the problem and help maintain the simplicity of the mathematical model.

B. System Dynamics

To model the body of a rocket, we consider a simplified cylindrical structure while ignoring aerodynamic effects as mentioned in (Section II-A). The rocket is equipped with an engine capable of producing thrust, subject to the following constraint:

$$0 < F_T < F_{T,\max} \tag{1}$$

Here, $F_{T,\max}$ represents the maximum thrust the engine can deliver (In the simulation $F_{T,\max} = 600\ N$). Thus, the thrust cannot exceed this upper limit.

In addition, the rocket nozzle, which is responsible for directing the thrust, can rotate within a restricted range.

$$-0.4 < \theta < 0.4 \tag{2}$$

Where θ is the angular displacement of the nozzle w.r.t rocket's body (in Radians). These constraints form part of the system's physical limitations. Later on we will use the above-mentioned constraints to form the optimization problem.

The forces acting on the rocket are defined below:

1) **Gravitational Force (F_g)** The downward force due to gravity is given by:

$$F_g = m \cdot g \tag{3}$$

where m is the rocket mass (Kg), and g is the gravitational acceleration, assumed to be $9.81\ \frac{\mathrm{m}}{\mathrm{s}^2}$.

2) **Thrust Force (F_t)** The force generated by the engine is directed by the angle of the nozzle (θ), and its magnitude can be controlled to influence the rocket's motion.

Below the state-space vector (eq.4) is defined:

$$\mathbf{s} = [x, y, \alpha, \dot{x}, \dot{y}, \dot{\alpha}]^T \tag{4}$$

Where x is the rocket position in the x direction (as depicted in Fig.1), y is the rocket y position, α is the rocket orientation w.r.t. the world coordinate system, $\dot{x}$ shows the velocity in the x direction, $\dot{y}$ velocity in the y direction, and $\dot{\alpha}$ is the angular velocity.

The equations of motion for the rocket are given by:

$$\ddot{x} = \frac{1}{m} F_T \sin(\alpha + \theta) \tag{5}$$

$$\ddot{y} = \frac{1}{m} F_T \cos(\alpha + \theta) - g \tag{6}$$

$$\ddot{\alpha} = \frac{r}{I} F_T \sin(\theta) \tag{7}$$

Where θ is the nozzle angle relative to the rocket's body frame, I is moment of inertia of the rocket body (cylindrical shape), calculated as $I = \frac{1}{12} m (L)^2$, where m is the mass and L is the length of the rocket body, and r is the distance from the center of the rocket's body to the nozzle.

In Eq.5, $\ddot{x}$ represents the acceleration in the x (horizontal) direction, $\ddot{y}$ in Eq.6 denotes the falling acceleration along the y-axis, and $\ddot{\alpha}$ in Eq.7 signifies the angular acceleration (reference coordinate has been defined in Fig.1).

The state space dynamics can be expressed as:

$$\dot{\mathbf{s}} = f(s, u) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \\ \ddot{x} \\ \ddot{y} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \\ \frac{1}{m} F_T \sin(\alpha + \theta) \\ \frac{1}{m} F_T \cos(\alpha + \theta) - g \\ \frac{r}{I} F_T \sin(\theta) \end{bmatrix}$$

This formulation provides the foundation for implementing a control strategy to regulate the rocket's motion during its descent.

The control inputs are therefore defined as follows:

$$u = [F_T, \theta]^\top$$

III. Formulation of the Minimization Problem

The minimization problem is formulated using the CasADi optimization framework. The following is a detailed explanation of the formulation process.

The Control Variables (F_t and θ):

$$\mathbf{u^T} = \begin{bmatrix} \mathbf{F}_k \\ \boldsymbol{\theta}_k \end{bmatrix} = \begin{bmatrix} F_0 & F_1 & \dots & F_{N-1} \\ \theta_0 & \theta_1 & \dots & \theta_{N-1} \end{bmatrix}$$

where:

- F_k: Thrust force at time step k.
- θ_k: Nozzle angle at time step k.

Both F_k and θ_k are defined as optimization variables for N discrete time steps.

The control input is $u_k \in \mathbb{R}^m;\ m = 2$,($F_k$ in N and $\theta_k$in Radians) subject to the lower and upper bounds $\underline{\mathrm{U}} \leq u_k \leq \overline{U}$.

$$\underline{\mathrm{U}} = \begin{bmatrix} 0\ (N) \\ -0.4\ (rad) \end{bmatrix}, \overline{U} = \begin{bmatrix} 600\ (N) \\ 0.4\ (rad) \end{bmatrix}$$

In the state space $s_k \in \mathbb{R}^n;\ n = 6$ from an initial condition $s_0 = s_{\text{init}}$ towards a desired or target state $s_f \in \mathbb{R}^n;\ n = 6$ across a specified time frame, which corresponds to the N discrete steps $k \in \{0, \dots, N\}$.

$$\dot{s} = f(s, u); \quad s_0 = s_{\text{init}} = \begin{bmatrix} 100\ (m) \\ 600\ (m) \\ 1.04\ (rad) \\ 0\ (\frac{m}{s}) \\ 0\ (\frac{m}{s}) \\ 0\ (\frac{rad}{s}) \end{bmatrix}, \quad u_0 = \begin{bmatrix} 0\ (N) \\ 0\ (rad) \end{bmatrix}.$$

$$s_{k+1} = F(s_k, u_k, \Delta t), \quad \forall k = 0, \dots, N-1 \tag{8}$$

To transition from the continuous-time system s_t = f(s_t, u_t) to the discrete-time form s_{k+1} = F(s_k, u_k, Δt), numerical discretization methods RK4 is applied. By approximating the

state evolution over small time steps Δt=$\frac{1}{50}$ (s).
RK4 integration method is implemented as follows:

$$k_1 = f(s_k, u_k), \tag{9}$$

$$k_2 = f(s_k + \frac{\Delta t}{2} k_1, u_k), \tag{10}$$

$$k_3 = f(s_k + \frac{\Delta t}{2} k_2, u_k), \tag{11}$$

$$k_4 = f(s_k + \Delta t k_3, u_k), \tag{12}$$

$$s_{k+1} = s_k + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{13}$$

The optimization in discrete form can be written as:

$$\begin{aligned} \min_{\{u_k\}_{k=0}^{N-1}} \quad & \sum_{k=0}^{N-1} \Big((s_k - s_f)^T Q (s_k - s_f) + u_k^T R u_k \Big) + FT \\ \text{subject to} \quad & \underline{\mathrm{U}} \leq u_k \leq \overline{U}, \quad \forall k = 0, \ldots, N-1. \end{aligned} \tag{14}$$

The optimization is initialized from an initial state s_{init}. In the end, the system is fed with the computed optimal control inputs one after another. A function F, which describes the dynamics of the system, defines how the state evolves based on the control inputs. In Particular, the state update is given by:

$$s_k = F(s_{k-1}, uk - 1, \Delta t),\ k = 1, \ldots, N. \tag{15}$$

This recursive relation can also be expressed in a nested form as Eq.16

$$s_k = F(F(F(F(s_0, u_0), u_1), \ldots, u_{k-1})) \tag{16}$$

Where $s_0 = s_{\text{init}}$. In this manner, the state s_k at each time step is updated by feeding the previous state back into the function F along with the corresponding control input.

In stabilization problems (for our case, landing a rocket), adding a terminal cost helps prevent the system from ending in an undesired state. It ensures that the system converges to s_f, reducing oscillations or overshooting in practical implementations where FT that defined in Eq.14 is given by:

$$FT = \Big((s_N - s_f)^T Q_f (s_N - s_f) \Big) \tag{17}$$

Here, s_N is the final state in the horizon.

$$s_N = \begin{bmatrix} x_N \\ y_N \\ \alpha_N \\ \dot{x}_N \\ \dot{y}_N \\ \dot{\alpha}_N \end{bmatrix}$$

s_f is the target state.

$$s_f = \begin{bmatrix} x_f \\ y_f \\ \alpha_f \\ \dot{x}_f \\ \dot{y}_f \\ \dot{\alpha}_f \end{bmatrix} = \begin{bmatrix} 400\ (m) \\ 0\ (m) \\ 0\ (rad) \\ 0\ (\frac{m}{s}) \\ 0\ (\frac{m}{s}) \\ 0\ (\frac{rad}{s}) \end{bmatrix}$$

u_N is the final control in the horizon.

$$u_N = \begin{bmatrix} F_{T,N}\ (N) \\ \theta_N\ (rad) \end{bmatrix}$$

Q is the symmetric positive semi-definite matrix penalizing deviations from s_f.

$$Q = \mathrm{diag}(q_x, q_y, q_\alpha, q_{\dot{x}}, q_{\dot{y}}, q_{\dot{\alpha}}) = \mathrm{diag}(10^3, 10^3, 10^3, 10^3, 10^3, 10^3)$$

R is the final weight matrix penalizing large control inputs.

$$R = \mathrm{diag}(r_{F_T}, r_\theta) = \mathrm{diag}(0.85, 1)$$

Q_f is the final symmetric positive semi-definite matrix penalizing deviations from s_f.

$$Q_f = \mathrm{diag}(q_{f,x}, q_{f,y}, q_{f,\alpha}, q_{f,\dot{x}}, q_{f,\dot{y}}, q_{f,\dot{\alpha}}) = \mathrm{diag}(10^3, 10^3, 4 \times 10^7, 10^3, 5 \times 10^3, 5 \times 10^3)$$

A. IPOPT Solver Configuration

The configuration limits the maximum number of iterations to 50, meaning that the solver will terminate within a predictable number of steps. This is important for real-time control applications where the computation time available is limited.

B. Solution and Output

The optimal control sequence is obtained by solving the optimization problem using the Casadi function `optim.solve()`.

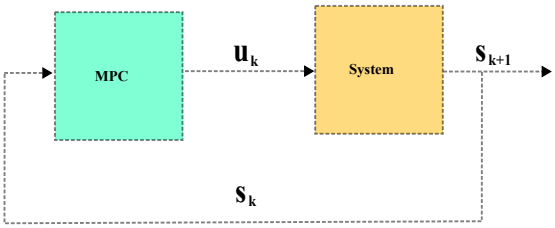


Fig. 3: MPC block diagram.

MPC block solves a finite horizon constrained optimization problem using the current state feedback to compute an optimal control sequence, applies the first control input, and then updates the system state recursively to drive the system toward the desired target.

IV. Results and Discussion

In this section, we present the results of the optimization-based control strategy implemented that has been discussed so far. The simulation was performed in a custom environment using PyMunk as the physics engine which models our environment while considering all the assumption stated above. Although the overall system follows the theoretical

formulation described in this paper, the coordinate system and rotation conventions in PyMunk differ from the standard mathematical conventions as depicted in Fig.2, and Fig.4. This discrepancy leads to changes in the signs, and angle phase shifts in the implementation.

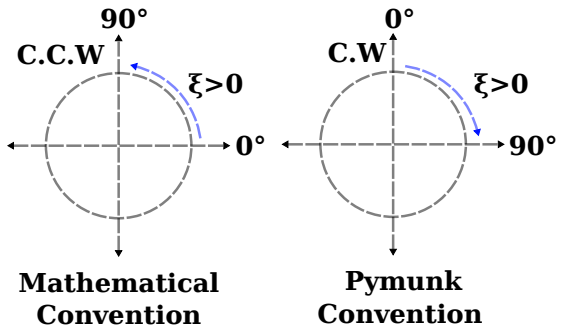


Fig. 4: Conventions.

Due to this difference, the signs of rotational terms in the state transition and control matrices were adjusted in the implementation to align with PyMunk's coordinate system. This adaptation ensures that the simulation behavior correctly reflects the theoretical dynamics described in this paper.

A. System Behavior and Performance

As depicted in Fig.7, the optimization successfully computes control inputs that:

- Minimize the orientation angle α (pitch) and its rate of change $\dot{\alpha}$, bringing both values close to zero over time.
- Adjust the system's position to converge towards the target landing location.
- Respect the physical constraints of the system, including the limit on thrust and nozzle angle.

The table I summarizes the computational performance of the CasADi solver for different function evaluations. The Objective Function (nlp_f) measures the cost function evaluation time, while the Constraint Function (nlp_g) accounts for constraint verification. The Gradient of the Objective (nlp_grad_f) and Jacobian of the Constraints (nlp_jac_g) compute the first-order derivatives necessary for optimization, whereas the Hessian of the Lagrangian (nlp_hess_l) represents second-order derivatives, making it the most computationally expensive operation. The Total Solver Time aggregates all operations, reflecting the overall efficiency of the solver. Notably, the Hessian computation exhibits the highest mean execution time and standard deviation, indicating a major contribution to computational cost. The total solution time varies across runs, suggesting that problem complexity influences solver convergence speed.

B. Concluding Remarks on the Results

The simulation results confirm the effectiveness of the proposed MPC-based control strategy [1]. Despite the differences in the coordinate system in PyMunk, the adaptation in the implementation ensures consistency with the theoretical model. The optimization algorithm successfully balances the competing objectives of stabilizing the system and achieving the target state.

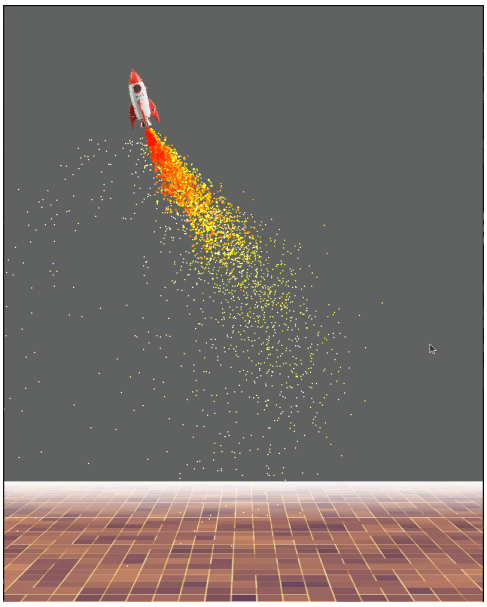
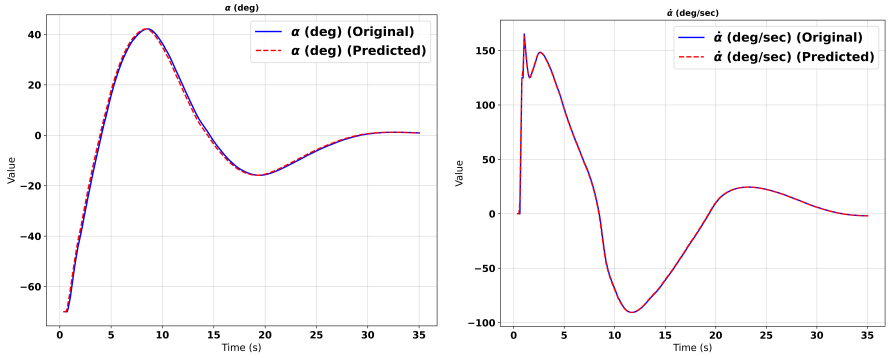


Fig. 5: Visualization of the simulation. The system trajectory is controlled using MPC to reach the desired target state. This figure shows the first 35 seconds of the simulation.

[1]Source code is available at: GitHub Repository (Earth Return/2d-mpc).

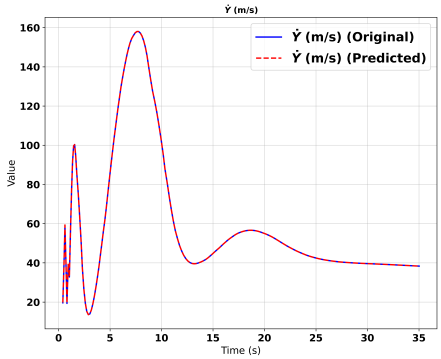
TABLE I: CasADi Solver Performance Statistics

Metric	**Min (μs)**	**Max (μs)**	**Mean (μs)**	**Std Dev (μs)**
nlp_f (Objective Function)	85.00	142.00	109.88	12.31
nlp_g (Constraints)	111.00	188.00	129.45	15.47
$nlp_{\text{grad } f}$ (Gradient of f)	179.00	258.00	213.82	19.71
$nlp_{\text{hess } L}$ (Hessian)	513.00	805.00	611.36	77.24
$nlp_{\text{jac } g}$ (Jacobian)	106.00	164.00	128.74	12.92
Total Time (ms)	74.95	102.22	90.54	6.71

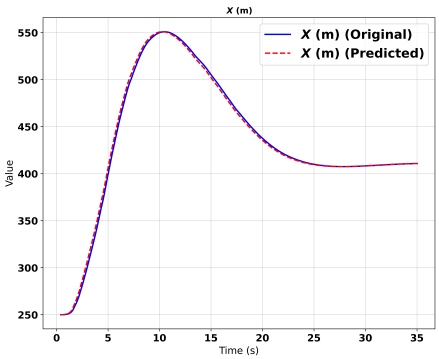


(a) Showing α converges to zero.

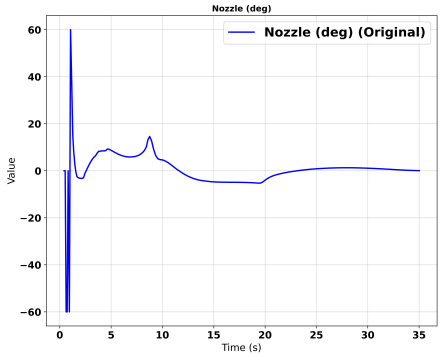
(b) Showing $\dot{\alpha}$ converges to zero.



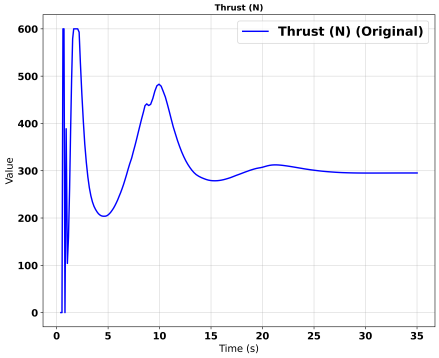
(c) $\dot{y}$ decreasing and approaching zero.



(d) x gradually approaches x_f

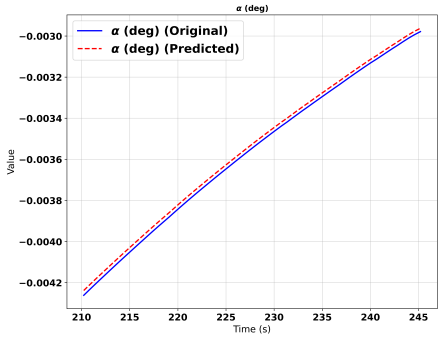


(e) θ stabilizing around zero.

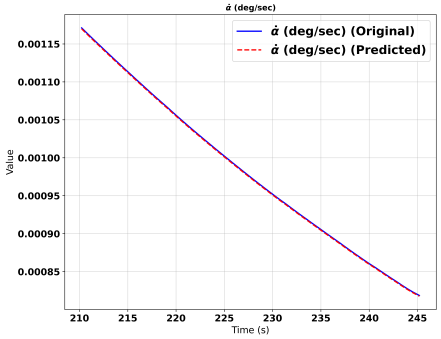


(f) F_t Thrust force around 300 (N).

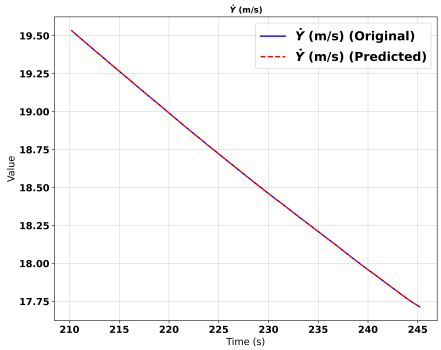
Fig. 6: The first 35 seconds of the simulation. As evident, the rocket's body angle and nozzle angle converge near zero after compensating for the initial angular deviation of -60 degrees from the vertical line. Additionally, the MPC controller adjusts the position x to approach the target value while minimizing the falling velocity to ensure a safe landing. The dotted red line represents the predicted states from the system dynamics based on the optimal control, demonstrating that it closely follows the real states obtained from the PyMunk physics library.



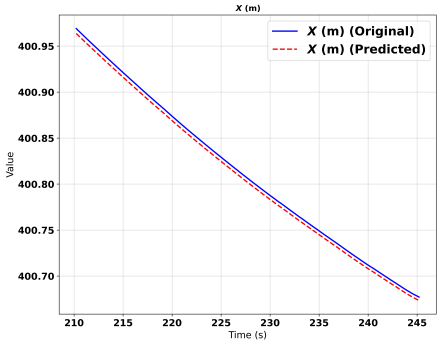
(a) Showing α converges to zero.



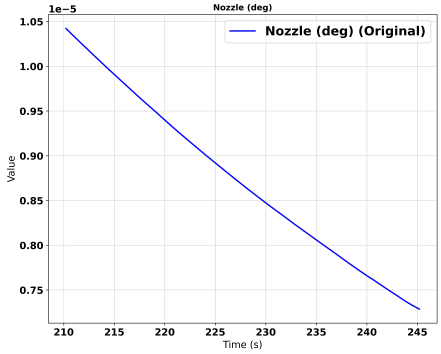
(b) Showing $\dot{\alpha}$ converges to zero.



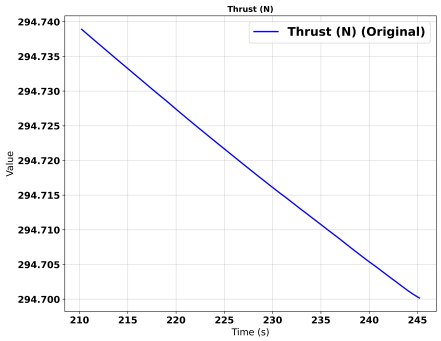
(c) $\dot{y}$ decreasing and approaching zero.



(d) x gradually approaches x_f



(e) θ stabilizing around zero.



(f) F_t Thrust force around 300 (N).

Fig. 7: The last 35 seconds of the simulation. As evident, the rocket's body angle and nozzle angle converge to zero and the MPC controller adjusts the position x to approach the target value 400, the falling velocity is less than 20 $\frac{m}{s}$.

V. Acknowledgment

We are deeply grateful for the exceptional support provided by Professor. The chair of Dr. Moritz Diehl and his affiliated institution. In particular, we extend our heartfelt thanks to Dr. Jakob Harzer for his invaluable guidance and unwavering support throughout this endeavor. His insightful advice and encouragement were instrumental in the successful completion of this work. Without their assistance and support, this achievement would not have been possible.

References

[1] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[2] V. S. Reddy. The spacex effect. *New Space*, 6(2):125–134, Jan. 2018.

[3] Johan Schoukens. System identification, 2013. Lecture notes.