LEARNING TO GENERATE INITIAL GUESS FOR TRAJECTORY OPTIMIZATION PROBLEM OF ROCKET LANDING

Group 2
Prayag Sharma, Youssef Nsouli, Yuxuan Zhu

ABSTRACT

The powered descent guidance of a three-degree-of-freedom (3-DOF) rocket vehicle requires solving a complex optimal control problem in real-time to ensure precise and fuel-efficient landings. Existing methods heavily rely on convex optimization techniques that necessitate a high-quality initial guess for rapid convergence, a challenging requirement given the real-time nature and computational constraints onboard rockets. This paper proposes leveraging Neural Networks (NNs) to generate accurate initial guesses for the optimizer, significantly reducing convergence time and enhancing the reliability of powered descent guidance algorithms. By analyzing and implementing various NN architectures, the study demonstrates the feasibility and advantages of this learning-based approach over traditional straight-line initializations. Comparative results highlight the potential improvements in computational efficiency and trajectory accuracy, paving the way for more robust and effective autonomous landing systems.

1 INTRODUCTION

The powered descent guidance of a three-degree-of-freedom (3-DOF) rocket vehicle involves calculating an optimal thrust sequence and a 3D trajectory to bring a rocket/booster from its initial state in air to rest on the ground in a vertical alignment. Trajectory optimization when performed online is called guidance, which is a part of the sub-group in rocket science referred to as Guidance, Navigation and Control (GNC). This is a challenging optimal control problem requiring precise navigation to ensure safe and accurate landing. Past two decades have seen a huge progress both within industry eg. Spacex of Engineering (2017), Blue Origin Fritz et al., and research agencies like JPL, NASA Blackmore et al. (2010) towards solving this problem. The fuel optimal landing of a rocket involves generating control inputs that minimize fuel consumption while adhering to dynamic and operational constraints, such as bounds on control authority such as thrust upper and lower bounds being one of them eg Acikmese & Ploen (2007), Reynolds et al. (2020), Kamath et al. (2023). The landing problem that solves the rocket landing problem by considering the rocket as a 3-DOF point mass model is considered in this study Acikmese & Ploen (2007). A complete 6-DOF model can also be used to accurate the rocket behavior more accurately, but is avoided here for simplicity. The 3-DOF rocket landing problem has a high degree of non-linearity and was previously solvable only by non-linear programming techniques in an offline setting. This prevented rockets from performing pinpoint landings due to the need for real-time solution generation by the rocket's computer, with response times of less than 1 second.

In 2009, NASA's JPL developed a lossless convexification technique that allowed this problem to be formulated purely as a convex optimization problem solvable in real-time on board the rocket Acikmese & Ploen (2007), Scharf et al. (2017), Carson et al.. This enabled pinpoint landings, unlike earlier guidance methods that lacked this capability Lu (2018). Current methods for solving this problem rely heavily on convex optimization, requiring a high-quality initial guess to achieve convergence quickly. Given that the rocket's true state is only known mid-air, it is difficult to pre-store an initial guess. The current approach to deal with this issue is to either initialize the solver with a straight line initial guess joining the current state and the final state, thus relaying heavily on the optimization solver to come up with the true optimal trajectory.

Or interpolate an initial guess from a large number of initial guesses stored on-board based on potential mid-air states, but due to limited memory on the rocket computer, this method is unreliable.

In this project, we aim to find a new time-saving mechanism to generate optimal trajectories. The objective function is in the form of minimizing the amount of fuel consumed by the rocket. We propose designing Neural Networks (NNs) to generate an initial guess that closely approximates the optimal solution therefore, helping the optimizer to reach the optimal solution in a fewer iterations. This can significantly help in reducing the optimizers convergence time, thereby enhancing the efficiency and reliability of the powered descent guidance algorithm. This is very recent idea and only a couple of studies have come to light to best of authors knowledge. Shen et al. (2022) introduces a Deep Neural Network based architecture which trains a separate DNN between any two nodes of the trajectory. The role of these DNN's is to predict the state of the vehicle at the next node/time instant when provided the current state/ state at the previous node as the input. if the trajectory is divided into 20 nodes, then 19 DNN's are used in this architecture. On the other hand Briden et al. (2024) proposes a transformer based trajectory generator for the powered descent guidance problem and introduces various flags to ensure the direct use of the transformer based guidance algorithm. This project studies various architectures that were successfully capable of capturing and generating initial guesses given the current state of the rocket. Furthermore efforts are also made to to compare the objective gain in performance by passing the initial guess proposed by the learning based Neural network for a given initial condition to a straight line initial guess.
This paper is divided majorly into three sections, "How does the proposed solution work?" explains in detail how and where exactly does the designed NN fit on-board the rocket to calculate the intial guess. "Powered Descent Guidance 3-DOF trajectory Optimization" details the optimal control problem and formulates as an optimization problem to be solved, followed by section on Machine learning that provides details of the Neural Network Design. "Using Machine Learning Models for Rocket Landing Optimization" tackles the framework discussed in this study. "Experimental Results" showcases the results of the NN study. "Warm-Starting the Solver" showcases and highlights the findings and important takeaways of this project.

2 How does the proposed solution work?

Figure 1 shows the general optimization procedure carried out on-board the rocket. For the given current state of the rocket, a straight-line initial guess joining the current state and the final desired state is selected as the initial guess. This guess clearly is not an optimal choice, since it is generally a zero vector. This solution methodology heavily relies on an optimization algorithm to come up with a feasible and optimal solution in real-time. The optimization solver converts the original non-convex optimization problem into an equivalent convex optimization problem and solves it by initializing the problem around the current initial guess x_0. After every single solve of the convex optimization problem a feasibility and convergence check is done, if the solution does not meet the desired accuracy, then another iteration of sequential convex programming is initiated with the current optimal solution as the initial guess. The optimal solution x^* contains all the sequence of f_i's from the current time t_0 to the final time t_f.

We propose a new architecture in which we replace the straight line initial guess generation with a neural network based guess generator. This new architecture is shown in Figure 2. The sequential convex optimization problem is now initialized with a guess generated by a custom designed neural network that takes, as input, the current state of the rocket and predicts what the optimal trajectory of the rocket may look like based on the training data. This neural network, once trained, can be placed on-board the rocket to compute the guess trajectory. If the neural network is able to accurately predict the trajectory, it would reduce the time needed for the sequential programming architecture to find the optimal solution. The idea is, any improvement from the current straight line initial guess generator, such as the Neural Network design will only improve the computational time of arriving at the optimal solution.

To train the neural network appropriately, we start generating optimal trajectories for a variety of possible initial conditions. From a uniform distribution we pick a possible initial condition and run the trajectory optimization code with a straight-line initial guess and record the final converged solution and the initial condition. We perform this for over 6000 trajectories and record the data. A model for this is also shown in Figure 3.

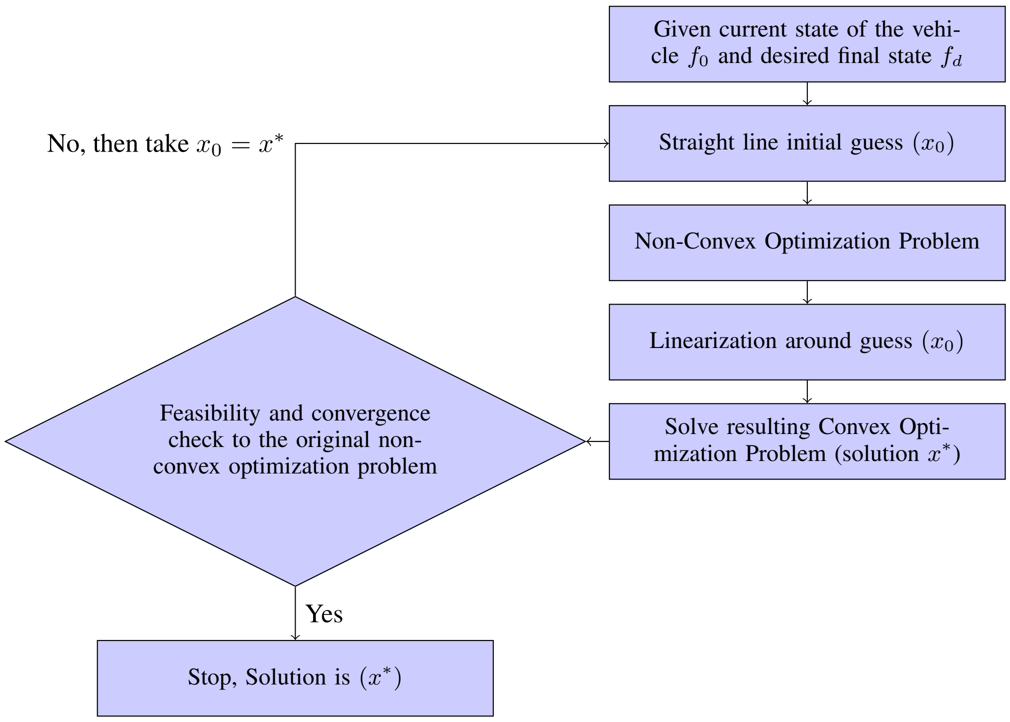

Given current state of the vehicle f_0 and desired final state f_d
Straight line initial guess (x_0)
Non-Convex Optimization Problem
Linearization around guess (x_0)
Solve resulting Convex Optimization Problem (solution x^*)
Feasibility and convergence check to the original non-convex optimization problem
No, then take $x_0 = x^*$
Yes
Stop, Solution is (x^*)

Figure 1: Standard model for computing trajectories on-board the rocket computer.

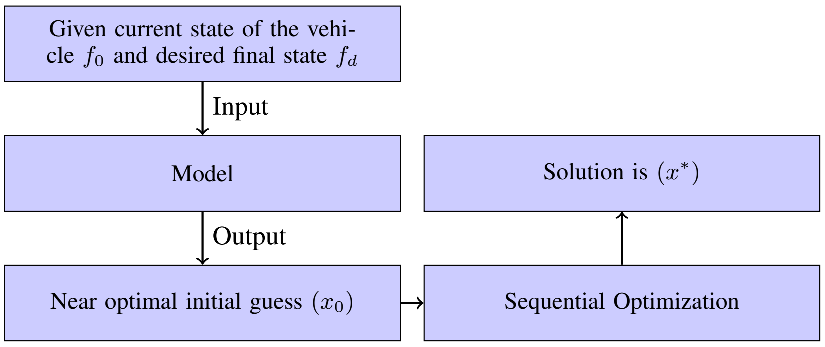

Given current state of the vehicle f_0 and desired final state f_d
Input
Model
Output
Near optimal initial guess (x_0)
Sequential Optimization
Solution is (x^*)

Figure 2: Proposed model for computing trajectories on-board the rocket computer.

3 Powered Descent Guidance 3-DOF Trajectory Optimization

This section provides details about the trajectory optimization problem formulated for the rocket landing problem. The table below provides a list of notation used in this section and elsewhere.

g	$=$ gravitational acceleration vector of Mars
g_e	$=$ magnitude of Earth's gravity
H	$=$ Hamiltonian of the system
I	$=$ identity matrix of appropriate dimensions
I_{sp}	$=$ specific impulse for thrusters
$\mathbb{R}$	$=$ the set of real numbers
$\mathbb{R}^n$	$=$ the space of n-dimensional real vectors
m	$=$ spacecraft mass
m_{wet}	$=$ spacecraft initial mass with the fuel
$\hat{n}_f$	final thrust direction

$\hat{n}_0$	initial thrust direction
p_k	$=$ vector coefficients for basis function ϕ_k
$Q = Q^T > 0$	$=$ denotes that the symmetric matrix Q is positive definite
R_0, R_1, R_2	$=$ functions defining the Hamiltonian
r	$=$ surface relative spacecraft position vector
r_0	$=$ spacecraft initial position vector
$\dot{r}$	$=$ surface relative spacecraft velocity vector
$\dot{r}_0$	$=$ spacecraft initial velocity vector
S_j, v_j, a_j	$=$ matrices, vectors, and scalars defining the convex state constraints
T_c or T	$=$ net thrust force vector acting on the spacecraft
T_1 and T_2	$=$ lower and upper bounds on the available thrust for the thrusters
t_f	$=$ time of flight for powered descent
t_f^*	$=$ optimal time of flight
u	$=$ net control acceleration induced by the thrusters
v	= unit vector describing a thrust pointing constraint
x	$=$ augmented spacecraft position and velocity vector
z	$=$ natural logarithm of mass
α	$=$ constant describing mass consumption rate
Γ	$=$ slack variable that bounds thrust magnitude
η	$=$ augmented vector of $p_k, k = 1, \ldots, M$
θ_{alt}	$=$ glide slope angle
$\tilde{\theta}_{\text{alt}}$	$=$ bound on the allowable glide slope angle
λ	$=$ costate vector
ρ_1	$=$ lower bound on thrust magnitude
ρ_2	$=$ upper bound on thrust magnitude
σ	$=$ mass normalized slack variable
$\Phi_k, \Lambda_k, \Psi_k, \Upsilon_k$	$=$ matrices describing the state transition in discrete time
ϕ	$=$ cant angle for the thrusters
ϕ_k	$=$ basis functions for numerical discretization
$\|x\|$	$=$ 2-norm of vector $x, \sqrt{x^\top x}$

3.1 Problem formulation

In this section we formulate the powered descent optimal control problem for a 3-DOF rocket landing on the surface of Mars. Please refer to the table below for the notations. An optimal control problem seeks to find the control sequence that minimizes a particular cost subject to the dynamics of a system and various other constraints enforced along the path and the boundary of this trajectory. The powered descent guidance problem for landing on the Martian surface has various constraints both in terms of the state variables and control variables.

3.1.1 State Dynamics

In control theory, the dynamics of a system can be represented as a function of the state vector, control vector and time as follows:

$$X : \text{State vector}, \; U : \text{Control Vector}, \; t : \text{time} \tag{1}$$

$$\dot{X}(t) = f(X(t), U(t), t) \tag{2}$$

For the current problem, the following state dynamics are to be considered:

Dynamics:

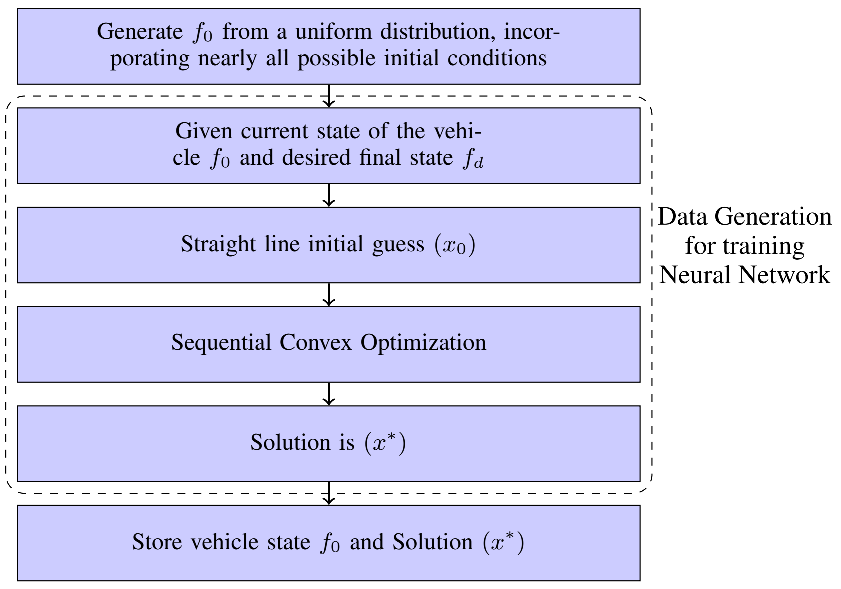

Generate f_0 from a uniform distribution, incorporating nearly all possible initial conditions
Given current state of the vehicle f_0 and desired final state f_d
Straight line initial guess (x_0)
Sequential Convex Optimization
Solution is (x^*)
Store vehicle state f_0 and Solution (x^*)
Data Generation for training Neural Network

Figure 3: Training the Neural Network.

$$\dot{r}\,(t) = V(t) \tag{3}$$
$$\dot{V}\,(t) = \ddot{r}(t) = g + \frac{T_c(t)}{m(t)} \tag{4}$$
$$\dot{m}(t) = -\alpha\,\|T_c(t)\| \tag{5}$$
$$r, V, T, g \in \mathbb{R}^3,\ m \in \mathbb{R} \tag{6}$$
$$X = [r, V, m]^\top, U = T_c \tag{7}$$

3.1.2 Path Constraints

Path constraints generally refer to the constraints that must not be violated by the optimal trajectory along the way. They are to be enforced at all times. Here, we have considered path constraints of multiple types, such as bounds on the thrust magnitude. This is primarily because the thruster onboard is limited in its capacity and may only generate a finite amount of thrust and it is in our best interest to have our control solution be realistic and close to the actual model. However, the following constraint is non-convex in nature and will be discussed in detail later on.

$$0 < \rho_1 \le \|T_c(t)\| \le \rho_2 \tag{8}$$

Other such constraints are like Glide slope constraints such as shown in the following image. A glide slope constraint ensures that the rocket does not go below a certain altitude level. The altitude level to be respected is a second-order cone function of the surface position.

All such constraints, such as glide slope constraints, bounds on the velocity vector of the rocket etc. can be written in the form of second order cones which are convex by nature as follows:

$$\|Sx(t) - v\| + c^\top x + a \le 0 \tag{9}$$

3.1.3 Boundary Conditions

Boundary conditions tell the optimizer what are the expected outcomes at the end of the trajectory for both state vector and control input. For the current problem the following boundary conditions on initial and final mass, as well as an initial starting point and ending location is specified. Along with this, we have are restrictions on velocity and thrust magnitude that ensure the rocket is at still position at the final time.

Conditions:

$$m(0) = m_{\text{wet}}\,, \quad r(0) = r_0, \quad \dot{r}(0) = \dot{r}_0 \tag{10}$$
$$r\left(t_f\right) = 0, \quad \dot{r}\left(t_f\right) = 0 \tag{11}$$
$$T_c(0) = \|T_c(0)\|\,\hat{n}_0, \quad T_c\left(t_f\right) = \left\|T_c\left(t_f\right)\right\|\hat{n}_f \tag{12}$$

3.1.4 Cost Function

The cost function in this problem is maximizing the final mass of the rocket or equivalently spending least amount of fuel possible. The final time can also be optimized if considered as a free parameter to be determined by the optimal control problem. This can be represented by the following equation:

$$\max_{t_f, T_c(\cdot)} \; m\left(t_f\right) = \min_{t_f, T_c(\cdot)} \int_0^{t_f} \|T_c(t)\|\,\mathrm{d}t \tag{13}$$

3.1.5 Complete Optimal control Problem

Combining all the sections above we can formulate a continuous time optimal control problem as follows:

Problem 1:

$$\left[\begin{array}{lll} & \max\limits_{t_f,T_c(\cdot)} m(t_f) = \min\limits_{t_f,T_c(\cdot)} \int_0^{t_f} \|T_c(t)\|\,\mathrm{d}t & \\ \text{State Dynamics:} & & \\ \dot{r}(t) = V(t) & \text{Path Constraints:} & \text{Boundary Conditions:} \\ \ddot{r}(t) = g + \frac{T_c(t)}{m(t)} & 0 < \rho_1 \le \|T_c(t)\| \le \rho_2 & m(0) = m_{\text{wet}},\ r(0) = r_0,\ \dot{r}(0) = \dot{r}_0 \\ \dot{m}(t) = -\alpha\|T_c(t)\| & \|Sx(t) - v\| + c^T x + a \le 0 & r(t_f) = 0,\ \dot{r}(t_f) = 0 \\ r, V, T, g \in \mathbb{R}^3,\ m \in \mathbb{R} & & T_c(0) = \|T_c(0)\|\hat{n}_0,\ T_c(t_f) = \|T_c(t_f)\|\hat{n}_f \end{array}\right] \tag{14}$$

To address the various non-convexities present in the problem formulation and Acikmese & Ploen (2007) reformulated or relaxed the above problem into an equivalent convex problem. A brief about the convexification has been given in the appendix. One can refer to Acikmese & Ploen (2007) and Scharf et al. (2017) for further details. After convexification the equivalent problem we try to address is of the following form:

Relaxed Convex Problem 2:

$$\left[\begin{array}{lll} & \min\limits_{t_f,\Gamma,T_c(\cdot)} \int_0^{t_f} \Gamma(t)\,\mathrm{d}t & \\ \text{State Dynamics:} & \text{Path Constraints:} & \text{Boundary Conditions:} \\ \dot{r}(t) = V(t) & \|T_c(t)\| \le \Gamma(t) & m(0) = m_{\text{wet}},\ r(0) = r_0,\ \dot{r}(0) = \dot{r}_0 \\ \ddot{r}(t) = g + \frac{T_c(t)}{m(t)} & 0 < \rho_1 \le \Gamma(t) \le \rho_2 & r(t_f) = 0,\ \dot{r}(t_f) = 0 \\ \dot{m}(t) = -\alpha\Gamma(t) & \|Sx(t) - v\| + c^T x + a \le 0 & T_c(0) = \|T_c(0)\|\hat{n}_0,\ T_c(t_f) = \|T_c(t_f)\|\hat{n}_f \\ r, V, T, g \in \mathbb{R}^3,\ \Gamma, m \in \mathbb{R} & & \end{array}\right] \tag{15}$$

The above problem is discretized and solved using a non-linear programming solver, `fmincon`, provided in `MATLAB`. The time grid is discretized evenly using a continuous spacing $t_k = k\Delta t, k = 0, ..., N$

3.1.6 Trajectory Optimization Results

This section provides a layout of the `MATLAB` code used to generate trajectories drawn from a uniform distribution of nominal initial conditions for training the neural networks. In total 6000, runs are generate and the data output of the optimal trajectories is stored in a `CSV` file.

3.1.7 Results

The Following figures show how the trajectory of the rocket looks like for one nominal initial condition. One can observe that the optimal solution attains its value at the boundary, which is similar to the analysis shown by the PMP in the previous section.

Optimal solutions generated with various initial conditions for training the neural network are also included follows:

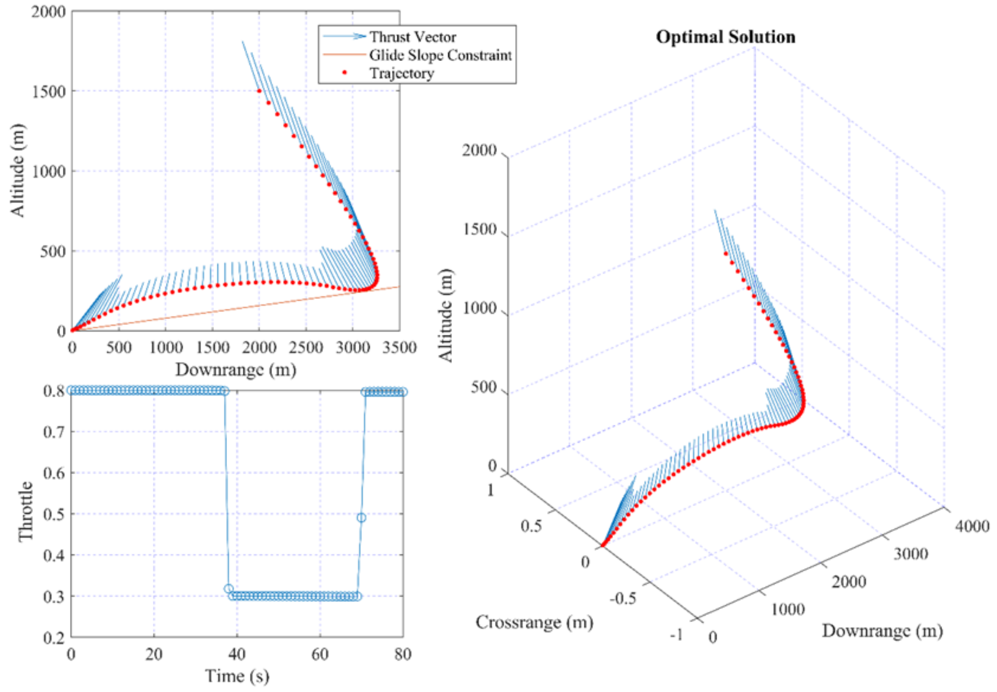
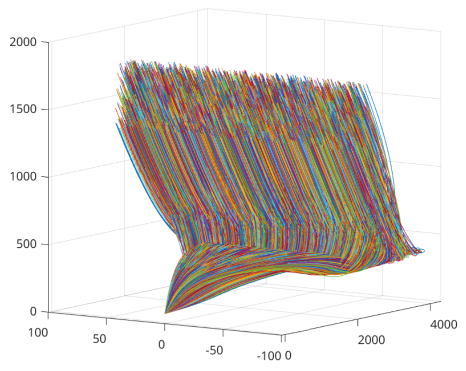
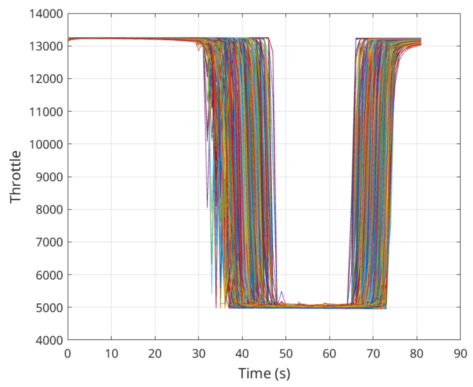


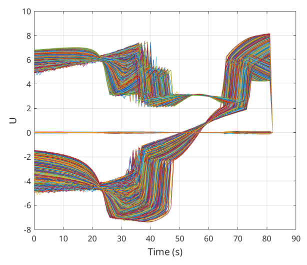
Figure 4: Nominal Optimal solution for one trajectory



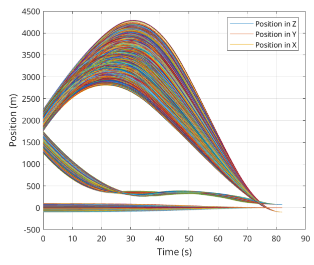
3D Trajectories for 6000 runs.



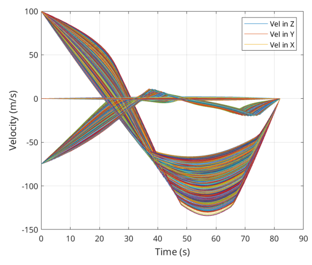
Norm of Thrust vector for 6000 runs.



Thrust Components plotted separately for 6000 runs.



Position plotted separately for 6000 runs.



Norm of Thrust vector for 6000 runs.

Figure 5: Graphs of the variation of the rocket vector state versus time.

4 Using Machine Learning Models for Rocket Landing Optimization

4.1 Goals and Incentives

The aim of this portion of the project is boost the project's time-sensitive performance. The means to do that will be in the form of modeling and training a Deep Neural Network (DNN hereinafter) and a Long Short Term Memory network (LSTM hereinafter) that will analyze and predict the optimal solution of the rocket's trajectory. The performance of each type of Neural Network (NN) will be measured and they will be evaluated against one another.

Optimally, the goal is to fully predict the rocket's path based on its current parameters; however, such results are likely to be unattainable. So, the goal to-be-achieved is an approximation of the optimal solution of the convexified optimization problem discussed above, which shall perform as a "good" estimator. The measure of "good" will be in how close an NN's output is to the optimal solution.

4.2 Input Data

The premises of generating realistic scenarios is a simple one. Random coordinate points with an altitude between $[1500, 2000]$ and horizontal distance between $[1100, 1900]$ are chosen, and the rest of the input features are chosen from pseudo-realistic scenarios that a rocket would be in just before trying to land. Then, a solver is used to generate a valid trajectory for the rocket, so that it would land within 100 meters of the origin point, $(0, 0, 0)$, which, for all intents and purposes, can be considered the landing zone.

The trajectories are then discretized to 83 points. The goal of discretization is so that the path points could be used as inputs to the NN in the `Python` environment. The data pre-processing phase is essential to prepare the dataset for effective training and testing. The dataset is loaded from the file, consisting of multiple traces containing 83 rows and 12 columns. The total number of traces, denoted as `n_trajectories`, can be derived by the total number of rows divided by 83.

The data takes on the following form:

T_x	T_y	T_z	$\|T\|$	v_x	v_y	v_z	x	y	z	$\log m$	t
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 2: Structure of input data

Where:

- T_x, T_y, T_z: Coordinate components of the thrust vector, T.
- $\|T\|$: The norm of the thrust vector.
- v_x, v_y, v_z: Coordinate components of the velocity vector, v.
- x, y, z: Coordinate components of the position vector.
- $\log m$: The logarithmic value of the rocket's mass, base 10.
- t: The "time-stamp" of the recorded trajectory point. This gives an idea about how much time is left before the rocket has to hit ground.

To create a training and testing set, $6,000$ trajectories were generated, 70% of the traces are allocated to the training set, while the remaining 30% are used for testing. This ensures that the model learns from a specific subset of traces and is evaluated on an entirely separate subset.

For the training set, input-output pairs are generated by taking all time-steps in each trace except the last one as inputs and all time-steps except the first one as targets, enabling the model to learn sequential relationships within each trace. In contrast, the testing set only uses the first time-step in each trace as input and the last time-step as the target. This setup is designed to test the model's ability to iteratively predict the sequence until reaching the final time-step, effectively measuring its ability to generalize over longer sequences.

4.3 Model Training

In the training phase, the NN iteratively processes the training data to minimize the error on the training set. The training loop can be described as follows:

Over each epoch and for each mini-batch of data in the training dataset, the following steps are performed:

- First, the input data, $\mathbf{X}^{t-1}$, is passed through the model in a forward pass, generating predictions, $\mathbf{Y}^{t}$, for each $t = 1, \dots T-1$. The input data is fed into the NN by the utility object, `DataLoader`, provided by `PyTorch`. An illustration showcasing a highly deviating prediction can be found in Figure 6.

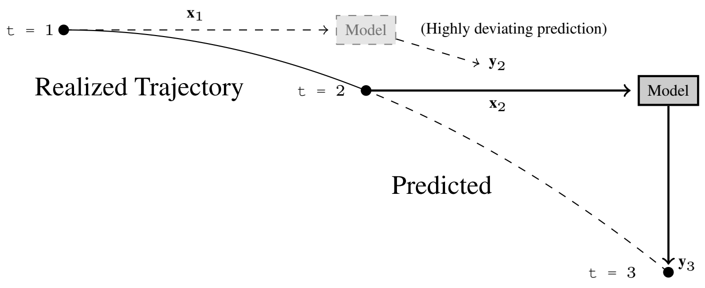


Figure 6: Illustration of the training environment of each model, depicting a highly deviating prediction. At t=1, the model is fed the true features, denoted as $\mathbf{x}_1$, then may produce a highly deviating output, $\mathbf{y}_2$. Then, since this is a training scenario, the actual realized features of t=2, denoted as $\mathbf{x}_2$, are plugged and the model predicts the next trajectory point is predicted as $\mathbf{y}_3$.

- Next, an appropriate loss function is used to calculate the different between the predictions $\mathbf{Y}^{t}$ and the true targets $\mathbf{X^t}$. This loss is then used in a backward pass to compute the gradients of the loss with respect to the model parameters through back-propagation.
- Finally, the model parameters are updated using the `ADAM` optimizer with a learning rate $\alpha = 0.001$.

The training loss for each epoch is averaged and printed to monitor the model's learning progress. Additionally, the training loop is tracked with `tqdm` to provide progress updates every 1000 batches.

4.4 Model Testing

In the testing phase, the model's performance is assessed on unseen data. For each test trajectory, the model starts from the first time-step, $t = 1$, and performs iterative predictions, sequentially predicting the next point in the trajectory until it lands. The model's final prediction for each test trajectory is then compared to the true last path point in that trajectory.

The performance of the model is measured using three evaluation metrics: mean squared error (MSE), mean absolute error (MAE), and root mean squared error (RMSE). These metrics are defined as follows:

$$\text{MSE} = \frac{1}{\text{n\_trajectories}} \sum_{i=1}^{\text{n\_trajectories}} (\mathbf{x}_i^\top - \mathbf{y}_i^\top)^2$$

$$\text{MAE} = \frac{1}{\text{n\_trajectories}} \sum_{i=1}^{\text{n\_trajectories}} |\mathbf{x}_i^\top - \mathbf{y}_i^\top|$$

$$\text{RMSE} = \sqrt{\text{MSE}}$$

After calculating these metrics, the final MSE, MAE, and RMSE values are printed to provide insight into the model's performance on the test set. Additionally, the predictions and targets for each test trajectory are saved as separate CSV files for further analysis, facilitating comparison and enabling visualization of the model's output accuracy.

4.5 Visualizing Output

Since the output features include three position coordinates, it is easy to visualize the output in a human-friendly manner.

By using Plotly, it is possible to draw trajectories on an interactive three-dimensional graph in Python. For example, the real trajectories from the input data are visualized in Figure 7.

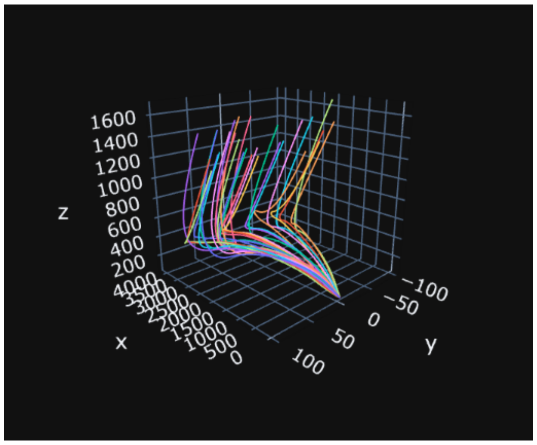


Figure 7: True input trajectories ($n = 200$) plotted in Python using Plotly.

5 Experimental Results

5.1 Experimental Settings

Model Architecture This study employs two distinct neural network architectures for time series prediction: a Deep Neural Network (DNN) and a Long Short-Term Memory (LSTM) network.

Deep Neural Network (DNN): The DNN model consists of multiple fully connected layers designed to capture nonlinear relationships within the input features. The specific architecture is as follows:

- **Input Layer**: Accepts input features at each time step with a dimensionality of `n_features = 12`.
- **Hidden Layers**: Comprises four hidden layers, each containing 128 neurons with ReLU activation functions. The operations for each layer are defined as:

$$\mathbf{h}^{(l)} = \mathrm{ReLU}(\mathbf{W}^{(l)}\mathbf{o}^{(l-1)} + \mathbf{b}^{(l)})$$

  where l denotes the layer number, $\mathbf{o}^{(l-1)}$ denotes the output of the previous layer, and $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ represent the weight matrix and bias vector, respectively.
- **Output Layer**: A fully connected layer that maps the final hidden state to the output dimension of 12.

An illustration of the DNN is depicted in Figure 8.

Long Short-Term Memory (LSTM): To capture temporal dependencies in the time series data, an LSTM-based recurrent neural network architecture is utilized. The detailed structure is as follows:

- **Input Layer**: Accepts input sequences of length 10, with each time step having 12 features. This means, to produce an output, $\mathbf{y}^t$, the LSTM needs $\mathbf{x}^{t-1}, \mathbf{x}^{t-2}, \ldots, \mathbf{x}^{t-9}$ as inputs.

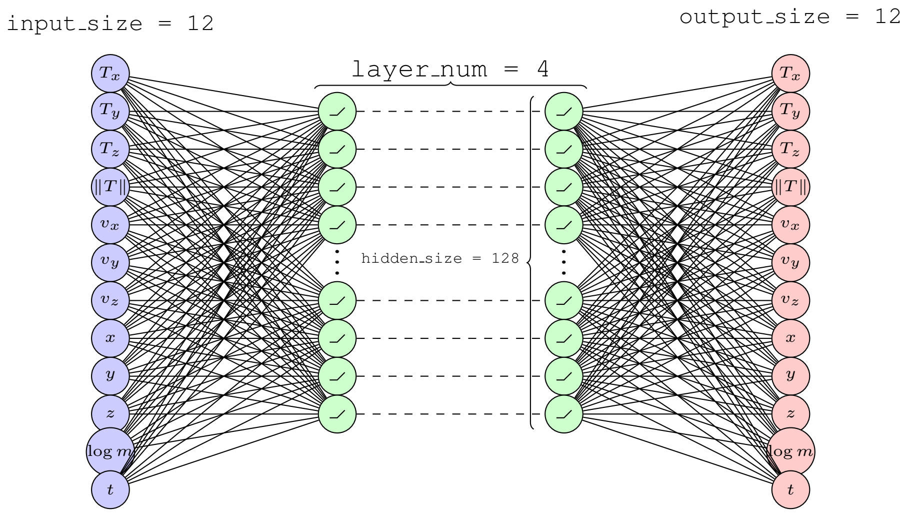


Figure 8: DNN architecture representation.

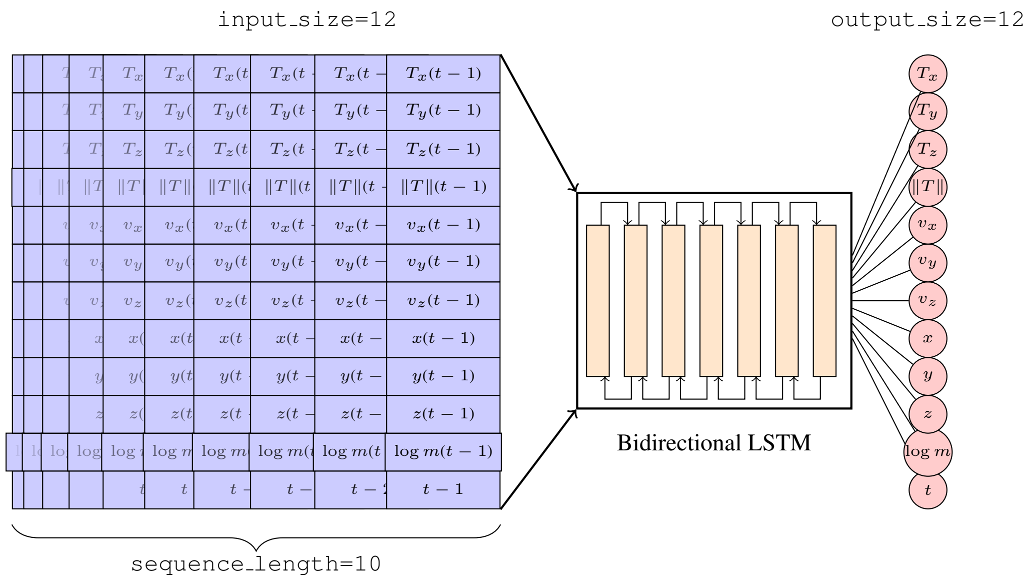


Figure 9: Summary schematic of the LSTM Model.

- **LSTM Layers**: Consists of two stacked LSTM layers, each with 128 hidden neurons. The `batch_first=True` parameter ensures that the input data shape is:

```
(batch_size, sequence_length, n_features)
```

- **Fully Connected Layer**: The hidden state from the last time step of the LSTM output is passed through a fully connected layer to produce a 12-dimensional output.

An illustration of the LSTM NN is demonstrated in 9.

Loss Function The accuracy of each output, denoted as $\mathbf{y}^t$, is measured against the input at the same timestamp, $\mathbf{x}^t$. The loss function used is the standard Mean Squared Error (MSE):

$$\mathcal{L}(\mathbf{X}^t, \mathbf{Y}^t) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i^t - \mathbf{y}_i^t\|_2^2$$

where

$$\mathbf{X}^t = \begin{bmatrix} \mathbf{x}_1^{t\top} \\ \mathbf{x}_2^{t\top} \\ \vdots \\ \mathbf{x}_n^{t\top} \end{bmatrix}$$

is the matrix of input trajectories, and

$$\mathbf{Y}^t = \begin{bmatrix} \mathbf{y}_1^{t\top} \\ \mathbf{y}_2^{t\top} \\ \vdots \\ \mathbf{y}_n^{t\top} \end{bmatrix}$$

is the output generated by the DNN or LSTM based on the input $\mathbf{X}^t$. Note that the time-step is chosen at random by the `DataLoader` object.

Evaluation Metrics As discussed before, the performance of the models against the test data is assessed using the following evaluation metrics:

- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**
- **Root Mean Squared Error (RMSE)**

It is worth noting that these metrics have been chosen since they are convex metrics. As mentioned before, the originally non-convex problem has been modeled as an equivalent convex problem with a convex feasible region, and the measure of "closeness" of a given point to the optimal solution can be estimated by convex functions that share an optimal solution.

5.2 DNN Results

For the training environment for the DNN, the number of epochs was set to `n_epochs = 40` and the batch size in the `DataLoader` object was set to `batch_size = 32`, so that the object can supply $10,760$ batches per epoch.

The accumulated evaluation metrics can be seen in Table 3:

As the results from Table 3 show many DNN's with similar performances, the traces generated by each should be examined to determine which architecture is most suitable for the test.

For each amount of hidden layers, the DNN with the best trajectory is depicted in Figure 10.

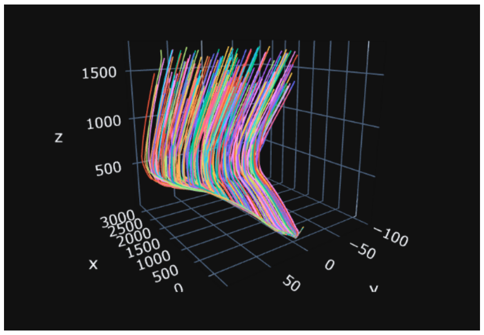
It is worth noting that at `num_layers=16` for hidden sizes of 32 and higher, the DNN's seemed to "condense" the trajectories to one single route, creating a tree-like figure. All three such DNN's do not create a logical trail, either. It is unclear why this is the case, but it may be due to inadequacy of training data, as the large amount of neurons need more data to converge onto a good set of parameters.

At `num_layers=4` and `num_layers=2` for `hidden_size=64` and lower, weird artifacts start to form. Nearing the tail of the trajectories, the paths seem to "wrap" around each other, similar to an ocean wave. Also, at the turning point of the trajectories, they seem to flatten, which would be likely cause crashes in a realistic scenario.

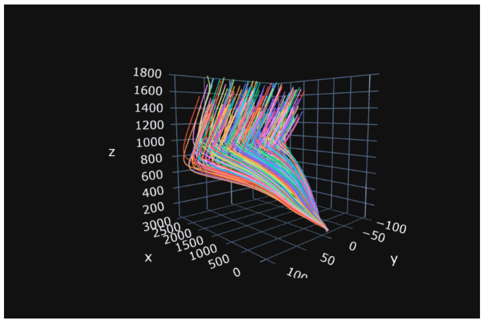
Based on the observations from Figure 10, `DNN_4_128` is the best performing DNN, and thus will be used. The rest of the trajectories can be seen in the appendix.

Layer Number	Hidden Size	MSE	MAE	RMSE
2	32	0.23	0.42	0.48
2	64	0.18	0.36	0.43
2	128	0.20	0.38	0.45
2	256	0.29	0.46	0.54
4	32	0.16	0.34	0.40
4	64	0.20	0.37	0.44
4	128	0.17	0.36	0.42
4	256	0.18	0.37	0.42
8	32	0.40	0.60	0.63
8	64	0.12	0.31	0.34
8	128	0.12	0.31	0.35
8	256	0.14	0.29	0.37
16	32	0.12	0.26	0.35
16	64	0.26	0.37	0.51
16	128	0.16	0.35	0.39
16	256	0.20	0.38	0.45

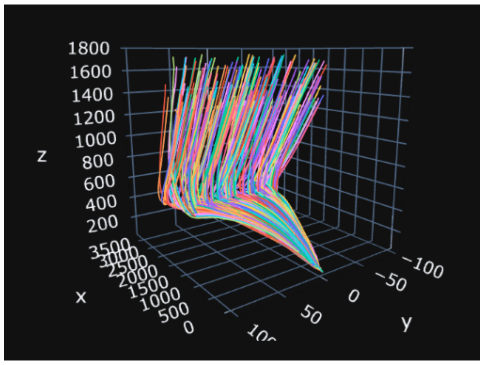
Table 3: Metric values from DNN's with a different number of hidden layers with different hidden sizes.



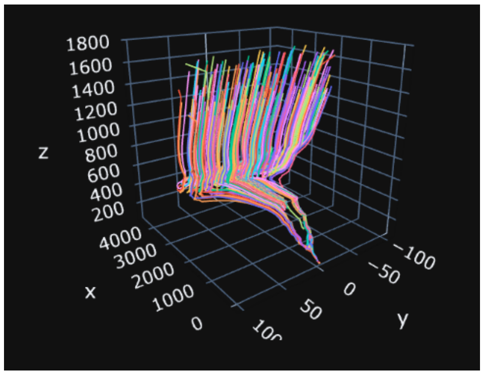
`num_layers=2, hidden_size=128`



`num_layers=4, hidden_size=128`



`num_layers=8, hidden_size=64`



`num_layers=16, hidden_size=32`

Figure 10: The best-performing DNN's for each `num_layers` value.

Model	Sequence Length	MSE	MAE	RMSE
RNN	1	0.48	0.61	0.69
RNN	3	0.30	0.40	0.55
RNN	10	0.21	0.32	0.46
RNN	25	0.41	0.53	0.64

Table 4: Performance Metrics for RNN with Different Sequence Lengths (Rounded to Two Decimal Places)

5.3 LSTM Results

The performance of the Long Short-Term Memory (LSTM) model was evaluated using varying sequence lengths: 1, 3, 10, and 25. The results indicate that the sequence length has a significant impact on the model's predictive performance. Among the tested sequence lengths, a sequence length of 10 achieved the best results. In contrast, shorter sequences, such as a length of 1, resulted in a higher MSE (0.4759), MAE (0.6071), and RMSE (0.6899), likely due to the limited temporal context provided to the model. Longer sequences, such as a length of 25, also degraded performance, with an MSE of 0.4128, MAE of 0.5293, and RMSE of 0.6425, potentially due to overfitting or the model's difficulty in handling excessive sequential dependencies.

Despite the ability of LSTMs to capture temporal dependencies, their overall performance was inferior to that of the Deep Neural Network (DNN) model across all metrics. The DNN, which operates without explicitly modeling temporal relationships, demonstrated superior accuracy and robustness compared to the LSTM. This is only demonstrated when looking at the trajectories generated by the LSTM Models, in Figure 11.

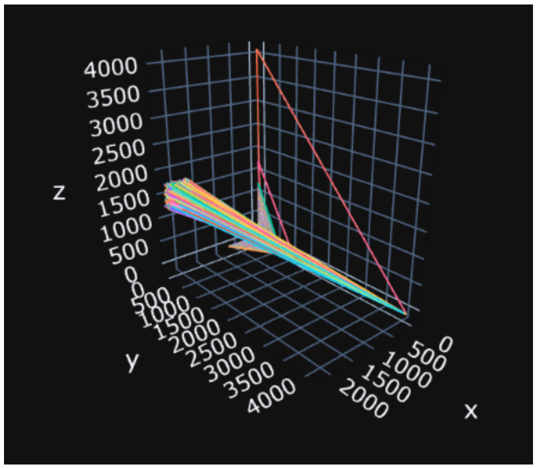


Figure 11: Trajectory Generated from the LSTM with `sequence_length=10`.

As it can be seen, the trajectories are erratic and do not seem to follow a certain pattern. For additional clarity, a single trajectory is shown in Figure 12.

The primary reasons for this performance discrepancy may be attributed to the following factors:

- **Data Characteristics:** The dataset might not exhibit complex temporal dependencies that LSTMs are designed to capture. In such cases, a simpler model like DNN may suffice and avoid the additional complexity introduced by recurrent layers.

- **Overfitting:** LSTMs with longer sequence lengths may be more prone to overfitting, particularly if the training data is insufficient or noisy. The increase in trainable parameters with longer sequences exacerbates this risk.

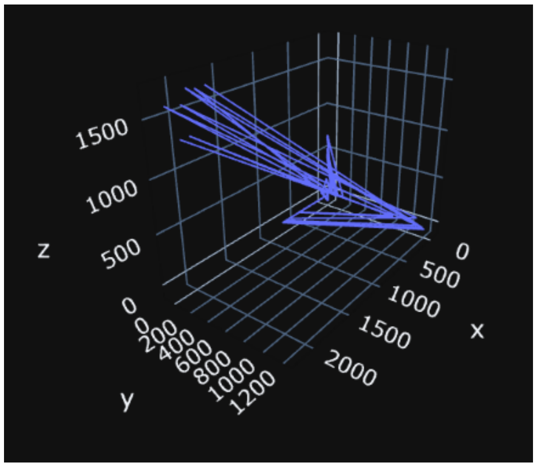


Figure 12: A single trajectory, as predicted by the LSTM of `sequence_length=10`.

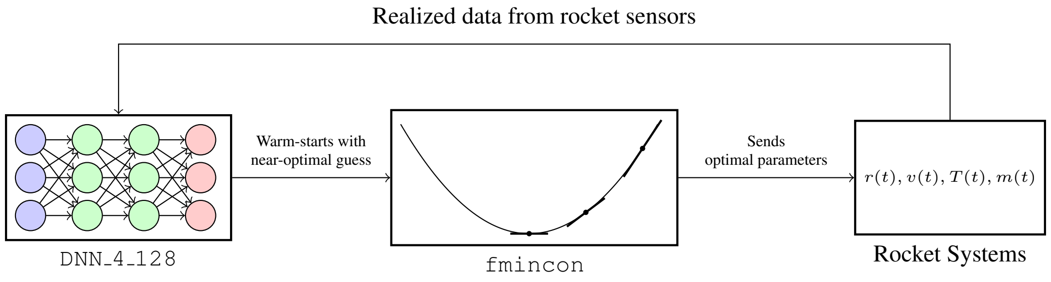


Figure 13: Illustration of the DNN and Solver working in tandem to navigate a rocket in a realistic scenario.

- **Model Complexity:** The added complexity of LSTMs requires more computational resources and may struggle to generalize as effectively as DNNs in scenarios where temporal dependencies are not dominant.

In conclusion, while LSTMs are powerful for capturing sequential patterns, their performance heavily depends on the data and sequence length configuration. In this experiment, the absence of pronounced temporal dependencies and the challenges associated with training recurrent architectures likely contributed to their underperformance relative to the simpler and more robust DNN model.

6 Warm-Starting the Solver

6.1 Mechanism

After choosing `DNN_4_128` as the best-performing NN, it will be tested in tandem with the nonlinear solver, `fmincon`, to see if the optimal solution is reached in less time.

The realistic scenario composes of first giving the DNN the initial true and realized parameters. Then, the DNN produces an initial guess for the second time-stamp, which will be fed to the convex solver to warm-start the solving algorithm. Then, the optimal solution is found by the solver and fed to the rocket systems. After a short amount of time, the true, realized parameters are fed back to the DNN and so on and so forth. A depiction of this model can be seen in Figure 13.

In the testing environment, we do not expect that the realized data will be different than the the output of the solver. So, the test will exclude the rocket systems components and be simplified to as can be seen in Figure 14.

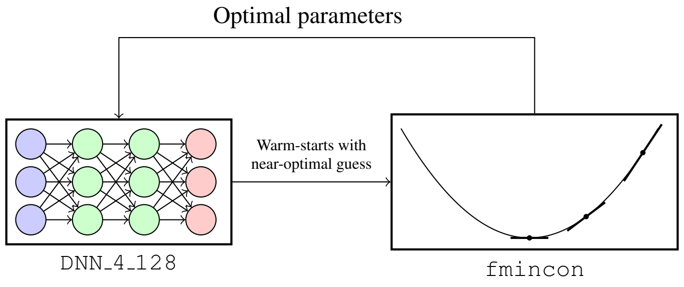


Figure 14: Illustration of the DNN and Solver working in tandem in the testing environment.

Since the solver and model were initially implemented in different programming languages, the DNN can be imported to `MATLAB` using the package `importNetworkFromPyTorch`, that takes in a pickled `PyTorch` model checkpoint. However, for the purposes of testing, the recorded time-stamped features will be considered. The forward-pass time of the model can be assumed to be negligible, considering that the solver to-be-used, `fmincon`, takes in the order of seconds to converge to the optimal solution.

The initial parameters from 10 distinct trajectories were plugged into the DNN and the resulting solution was iterated over each time-stamp until the rocket reaches the landing space. The average time taken until the optimal solution was reached in each time-stamp is available in the "DNN + Solver" column in Table 5. The control test composes of plugging the initial parameters into the solver directly and having it iterate until the rocket lands. The results are available in the same table under the "Solver only" column.

Test Case	**DNN + Solver**		**Solver Only**	
	Time (s)	**Iterations**	**Time (s)**	**Iterations**
1	5.2	17	20.0	74
2	12.5	43	21.2	74
3	19.0	70	21.4	74
4	5.1	16	20.9	74
5	17.3	61	21.0	74
6	5.2	17	21.0	74
7	5.8	19	21.0	74
8	5.6	18	21.0	74
9	5.7	19	21.0	74
10	16.16	56	21.0	74

Table 5: Time and iterations needed to converge to the optimal solution while using two different architectures.

As it can be seen, the use of DNN + Solver allows reaching for the optimal solution on average 10 seconds faster the solver, or in less than half the time. This also corroborates the previous assumption that the time needed for the DNN to compute the initial guess can be considered negligible, since it takes milliseconds to compute, much faster than the valuable seconds saved.

7 Conclusion

The problem of rocket guidance has, continues, and will continue to see studies as it is a difficult problem with a lot of sensitive areas. The problem addressed in this paper is a simplified problem, where a rocket is to land while attempting to minimize the amount of rocket fuel consumed. This is, generally, a non-convex problem. Although, it has a convex equivalent, which allows for optimal solutions to be found faster. Yet, the unpredictability of the rocket environment has forced researchers to implement solvers on-board the rocket to steer the rocket as desired.

As said solver may take seconds to compute new instructions, a DNN was proposed in this paper to generate a good initial guess for the solver. This initial guess is then plugged into the solver to warm-start the algorithm and achieve optimality faster.

The study performed in this paper has resulted in a trained DNN that can generate solutions that, when used to warm-start, led to optimal solutions found at almost twice the rate, on average.

REFERENCES

Behcet Acikmese and Scott R. Ploen. Convex programming approach to powered descent guidance for mars landing. *Journal of Guidance, Control, and Dynamics*, 30(5):1353–1366, 2007. doi: 10.2514/1.27553. URL `https://doi.org/10.2514/1.27553`.

Lars Blackmore, Behçet Açikmeşe, and Daniel P. Scharf. Minimum-landing-error powered-descent guidance for mars landing using convex optimization. *Journal of Guidance, Control, and Dynamics*, 33(4):1161–1171, 2010. doi: 10.2514/1.47202. URL `https://doi.org/10.2514/1.47202`.

Julia Briden, Trey Gurga, Breanna J. Johnson, Abhishek Cauligi, and Richard Linares. *Improving Computational Efficiency for Powered Descent Guidance via Transformer-based Tight Constraint Prediction*. 2024. doi: 10.2514/6.2024-1760. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2024-1760`.

John M. Carson, Michelle M. Munk, Ronald R. Sostaric, Jay N. Estes, Farzin Amzajerdian, James B. Blair, David K. Rutishauser, Carolina I. Restrepo, Alicia M. Dwyer-Cianciolo, George Chen, and Teming Tse. *The SPLICE Project: Continuing NASA Development of GN&amp;C Technologies for Safe and Precise Landing*. doi: 10.2514/6.2019-0660. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2019-0660`.

Matthew Fritz, Javier Doll, Kari C. Ward, Gavin Mendeck, Ronald R. Sostaric, Sam Pedrotty, Chris Kuhl, Behcet Acikmese, Stefan R. Bieniawski, Lloyd Strohl, and Andrew W. Berning. *Post-Flight Performance Analysis of Navigation and Advanced Guidance Algorithms on a Terrestrial Suborbital Rocket Flight*. doi: 10.2514/6.2022-0765. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2022-0765`.

Abhinav G. Kamath, Purnanand Elango, Yue Yu, Skye Mceowen, Govind M. Chari, John M. Carson III, and Behçet Açıkmeşe. Real-time sequential conic optimization for multi-phase rocket landing guidance. *IFAC-PapersOnLine*, 56(2):3118–3125, 2023. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2023.10.1444. URL `https://www.sciencedirect.com/science/article/pii/S2405896323018529`. 22nd IFAC World Congress.

Ping Lu. Propellant-optimal powered descent guidance. *Journal of Guidance, Control, and Dynamics*, 41(4):813–826, 2018. doi: 10.2514/1.G003243. URL `https://doi.org/10.2514/1.G003243`.

National Academy of Engineering. *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium*. National Academies Press, 2017. ISBN 9780309450393. URL `https://books.google.com/books?id=tCFCDgAAQBAJ`.

Taylor P. Reynolds, Michael Szmuk, Danylo Malyuta, Mehran Mesbahi, Behçet Açıkmeşe, and John M. Carson. Dual quaternion-based powered descent guidance with state-triggered constraints. *Journal of Guidance, Control, and Dynamics*, 43(9):1584–1599, 2020. doi: 10.2514/1.G004536. URL `https://doi.org/10.2514/1.G004536`.

Daniel P. Scharf, Behçet Açıkmeşe, Daniel Dueri, Joel Benito, and Jordi Casoliva. Implementation and experimental demonstration of onboard powered-descent guidance. *Journal of Guidance, Control, and Dynamics*, 40(2):213–229, 2017. doi: 10.2514/1.G000399. URL `https://doi.org/10.2514/1.G000399`.

Zhipeng Shen, Shiyu Zhou, and Jianglong Yu. Real-time computational powered landing guidance using convex optimization and neural networks. *arXiv preprint arXiv:2210.07480*, 2022.

A Convexification of the Non-convex Problem

A.1 Lossless Convexification and Problem reformulation to remove non-convexity

In this section, we try to address the various non-convexities present in the problem formulation and try to reformulate or relax them into convex ones. Then we try to prove that a solution to this relaxed problem attains the same solution as the original problem using Pontryagin's Maximum Principle.

A.1.1 Thrust upper bound

The thrust upper and lower bound represents a non-convex constraint.

$$0 < \rho_1 \leq \|T_c(t)\| \leq \rho_2 \tag{16}$$

Even though the thrust is a 3-dimensional vector, the 2-dimensional version can still clearly show the non-convex nature of thrust, and the graph can be seen below:

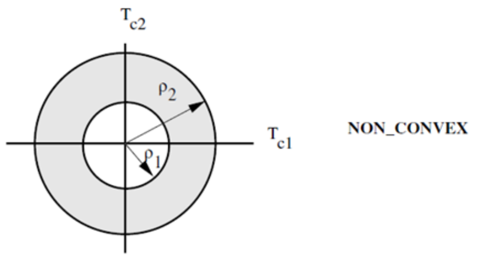


Figure 15: Non convex thrust constraint (a 2D representation)

The above non-convexity can be lifted by introducing a new auxiliary variable into the system. This leads to the following two constraints:

$$1.\ \|T_c(t)\| \leq \Gamma(t) \tag{17}$$

$$2.\ 0 < \rho_1 \leq \Gamma(t) \leq \rho_2 \tag{18}$$

The lifting of non-convexity can be visualized for a 2D case as follows:

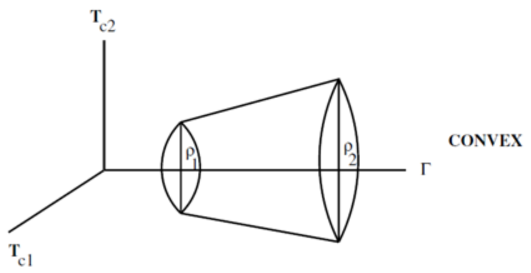


Figure 16: Relaxed convex thrust constraint (a 2D representation)

A.1.2 Non-Convex Cost

The non-convexity of the cost can be easily handled by an equivalent transformation as follows:

$$\min \int_{t_0}^{t_f} \|u\|\, dt \rightarrow \min \int_{t_0}^{t_f} \eta\, dt \tag{19}$$

$$\|u\| \leq \eta \tag{20}$$

A.1.3 Combining the relaxations

Therefore the original non-convex parameters in the original problem:

Problem 1:

$$\min_{t_f, T_c(\cdot)} \int_0^{t_f} \|T_c(t)\| \, \mathrm{d}t \tag{21}$$

$$\text{subject to: } 0 < \rho_1 \leq \|T_c(t)\| \leq \rho_2 \tag{22}$$

Can be removed by introducing the above relaxations, and a new problem can be formulated as follows:

Problem 2:

$$\min_{t_f, T_c(\cdot)} \int_0^{t_f} \Gamma \, dt \tag{23}$$

$$\text{subject to: } \|T_c(t)\| \leq \Gamma(t) \tag{24}$$

$$0 < \rho_1 \leq \Gamma(t) \leq \rho_2 \tag{25}$$

A.1.4 Proving the relaxations are tight / Lossless using Pontryagin's maximum principle, PMP

The proof here does not involve second order cone constraints, as it can be proven that these constraints do not affect the analysis shown below. It has been shown that the second order cone constraints of the current problem do not affect the Hamiltonian in a way that changes the nature of the optimal solution.

Lemma 1: Consider a solution of Problem 2 given by $\left[t_f^*, T_c^*(\cdot), \Gamma^*(\cdot)\right]$. Then, $\left[t_f^*, T_c^*(\cdot)\right]$ is also a solution of Problem 1 and $\|T_c^*(t)\| = \rho_1$ or $\|T_c^*(t)\| = \rho_2$ for $t \in \left[0, t_f^*\right]$. Consider The original problem below:

Problem 1:

$$\max_{t_f, T_c(\cdot)} \int_0^{t_f} \lambda_0 \|T_c(t)\| \, \mathrm{d}t \tag{26}$$

$$\text{subject to: } 0 < \rho_1 \leq \|T_c(t)\| \leq \rho_2, \lambda_0 \leq 0 \tag{27}$$

$$\dot{r}\,(t) = V(t) \tag{28}$$

$$\ddot{r}(t) = g + \frac{T_c(t)}{m(t)} \tag{29}$$

$$\dot{m}(t) = -\alpha \|T_c(t)\| \tag{30}$$

$$r, V, T, g \in \Re^3, \; m \in \Re \tag{31}$$

According to optimal control theory, we must construct the Hamiltonian and write the equations of optimality. PMP states that at the optimal control value, the Hamiltonian function attains its maximum.

Problem 1:

$$\max_{t_f, T_c(\cdot)} \int_0^{t_f} \lambda_0 \|T_c(t)\| \, \mathrm{d}t \tag{32}$$

$$\text{subject to: } 0 < \rho_1 \leq \|T_c(t)\| \leq \rho_2 \tag{33}$$

$$H = \lambda_0 \|T_c(t)\| + \lambda_r^T (V) + \lambda_V^T \left(g + \frac{T_c(t)}{m(t)}\right) + \lambda_m \left(-\alpha \|T_c(t)\|\right) \tag{34}$$

$$\text{Costate Equation: } \dot{\lambda} = -\frac{\partial H}{\partial X} \tag{35}$$

$$\text{State Equation: } \dot{X} = \frac{\partial H}{\partial \lambda} \tag{36}$$

$$\text{Boundry Conditions (Free final time): } H|_{t_f} = \frac{\partial \phi_f}{\partial X_f} = 0 \tag{37}$$

By solving the above equations we can find that the Hamiltonian attains its maximum if the switching function behaves as follows:

$$H = R_1(t)\,\|T_c(t)\| + \|R_2(t)\|\,\|T_c(t)\| + R_0(t) \tag{38}$$

$$H = \underbrace{(R_1(t) + \|R_2(t)\|)}_{\text{Switching Function}}\|T_c(t)\| + R_0(t) \tag{39}$$

$$H = (S_f)\,\|T_c(t)\| + R_0(t) \tag{40}$$

$$\|T_c(t)\| = \begin{Bmatrix} \rho_2, \text{ if } S_f > 0 \\ \rho_1, \text{ if } S_f < 0 \end{Bmatrix} \tag{41}$$

Similarly analysing the Optimal control solution for problem 2 as follows:

Problem 2:

$$\max_{t_f, T_c(\cdot)} \int_0^{t_f} \lambda_0 \Gamma \, \mathrm{d}t \tag{42}$$

$$\text{subject to: } 0 < \rho_1 \le \Gamma(t) \le \rho_2,\ \|T_c(t)\| \le \Gamma(t),\ \lambda_0 \le 0 \tag{43}$$

$$\dot{r}(t) = V(t) \tag{44}$$

$$\ddot{r}(t) = g + \frac{T_c(t)}{m(t)} \tag{45}$$

$$\dot{m}(t) = -\alpha\Gamma(t) \tag{46}$$

$$r, V, T, g \in \Re^3,\ m \in \Re \tag{47}$$

Writing the conditions of optimality as follows:

Problem 2:

$$\max_{t_f, T_c(\cdot)} \int_0^{t_f} \lambda_0 \Gamma \, \mathrm{d}t \tag{48}$$

$$\text{subject to: } 0 < \rho_1 \le \Gamma(t) \le \rho_2,\ \|T_c(t)\| \le \Gamma(t),\ \lambda_0 \le 0 \tag{49}$$

$$H = \lambda_0\Gamma + \lambda_r^T(V) + \lambda_V^T\left(g + \frac{T_c(t)}{m(t)}\right) + \lambda_m(-\alpha\Gamma(t)) \tag{50}$$

$$PMP:\ H(\lambda_0, \lambda, X, T_c^*, \Gamma^*(t)) \ge H(\lambda_0, \lambda, X, T_c, \Gamma(t)) \tag{51}$$

$$\tag{52}$$

By constructing the optimality equations and solving for the optimal control, it turns out the switching function that maximizes the Hamiltonian attains the same expression as it does in problem 1 as follows:

$$(0 < \rho_1 \le \Gamma(t) \le \rho_2) \tag{53}$$

$$H = R_1(t)\,\Gamma(t) + \|R_2(t)\|\,\Gamma(t) + R_0(t) \tag{54}$$

$$H = \underbrace{(R_1(t) + \|R_2(t)\|)}_{\text{Switching Function}}\Gamma(t) + R_0(t) \tag{55}$$

$$H = (S_f)\,\Gamma(t) + R_0(t) \tag{56}$$

$$\Gamma(t) = \|T_c(t)\| = \begin{Bmatrix} \rho_2, \text{ if } S_f > 0 \\ \rho_1, \text{ if } S_f < 0 \end{Bmatrix} \tag{57}$$

Which implies that the relaxation is lossless. Therefore, solving problem 2 is equivalent to solving problem 1. However, one can observe that since the new problem is convex in nature it can be solved in polynomial time as compared to the original non-convex problem. With a convex formulation we can find an upper bound on the number of iterations required to achieve a certain degree of optimality, and most importantly ensure that the solution is globally optimal.

A.2 Writing the equivalent Relax problem formulation

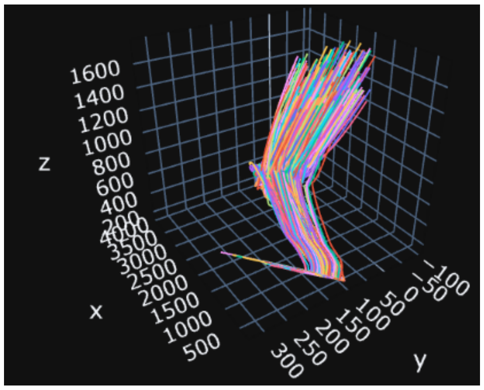
By combining the above results the equivalent relaxed problem can be written as follows:

Relaxed Convex Problem 2:

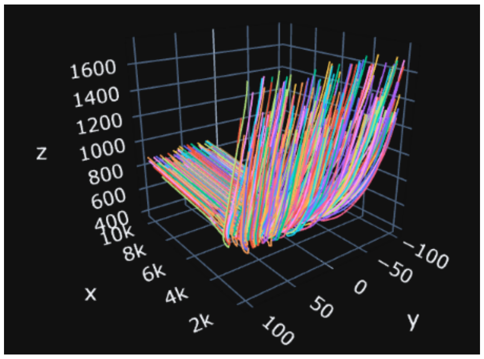
$$\left[\begin{array}{lll} & \min\limits_{t_f,\Gamma,T_c(\cdot)} \int_0^{t_f} \Gamma(t)\,\mathrm{d}t & \\ \text{State Dynamics:} & & \\ \dot{r}(t) = V(t) & \text{Path Constraints:} & \text{Boundary Conditions:} \\ \ddot{r}(t) = g + \frac{T_c(t)}{m(t)} & \|T_c(t)\| \leq \Gamma(t) & m(0) = m_{\text{wet}},\ r(0) = r_0,\ \dot{r}(0) = \dot{r}_0 \\ \dot{m}(t) = -\alpha\Gamma(t) & 0 < \rho_1 \leq \Gamma(t) \leq \rho_2 & r(t_f) = 0,\ \dot{r}(t_f) = 0 \\ r, V, T, g \in \mathbb{R}^3,\ \Gamma, m \in \mathbb{R} & \|Sx(t) - v\| + c^T x + a \leq 0 & T_c(0) = \|T_c(0)\|\hat{n}_0,\ T_c(t_f) = \|T_c(t_f)\|\hat{n}_f \end{array}\right] \tag{58}$$

The above problem is discretized and solved using a cone programming solver in `MATLAB`. The time grid is discretized evenly using a continuous spacing $t_k = k\Delta t, k = 0, ..., N$

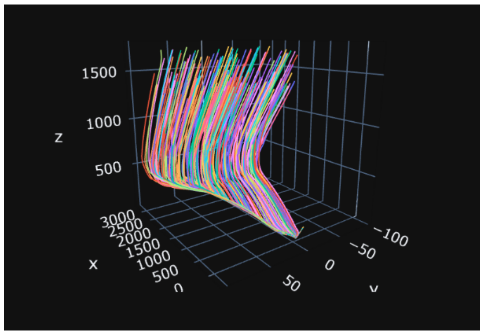
B All Trajectories



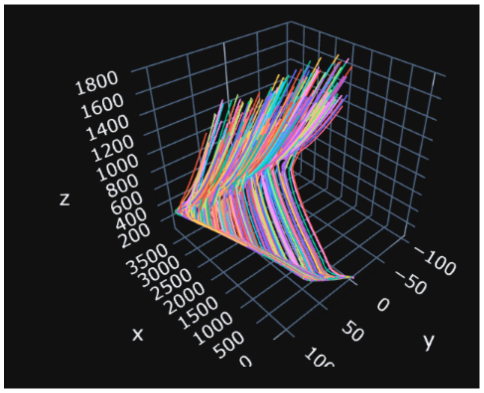
hidden_size=32



hidden_size=64

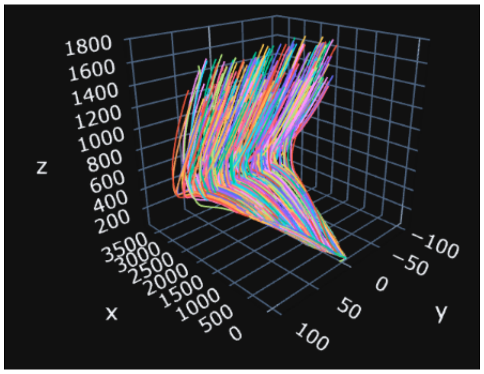


hidden_size=128

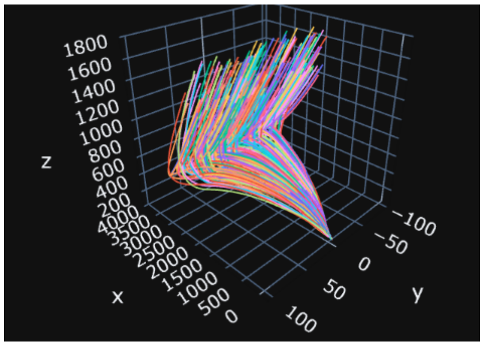


hidden_size=256

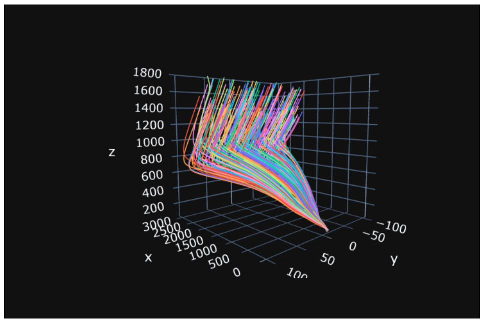
Figure 17: Trajectories generated by the DNN's for each where `num_layers=2`.



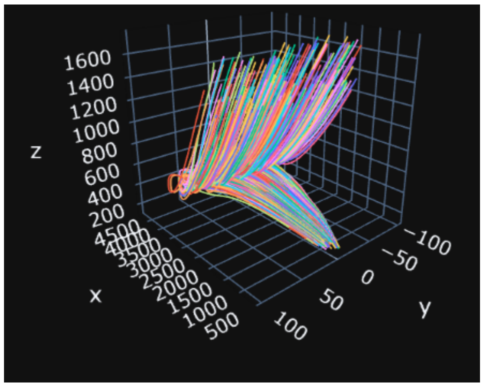
hidden_size=32



hidden_size=64

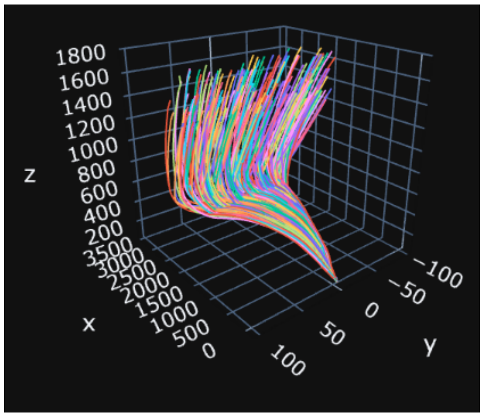


hidden_size=128

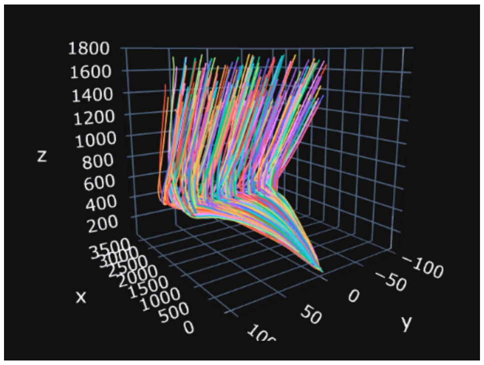


hidden_size=256

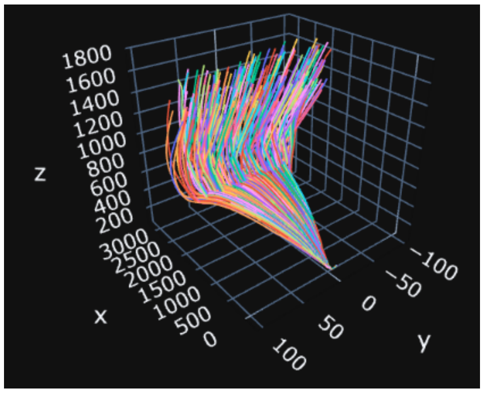
Figure 18: Trajectories generated by the DNN's for each where `num_layers=4`.



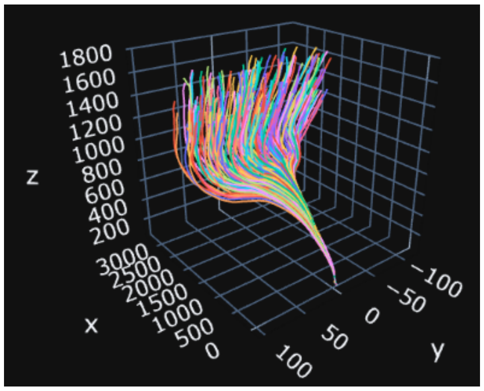
hidden_size=32



hidden_size=64

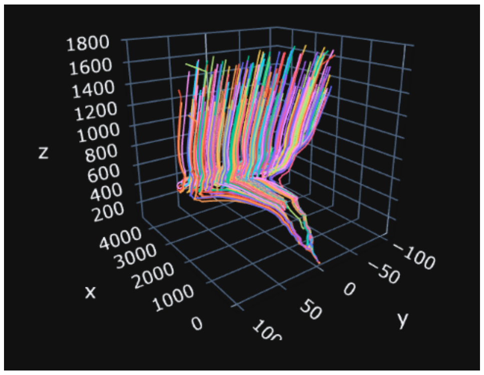


hidden_size=128

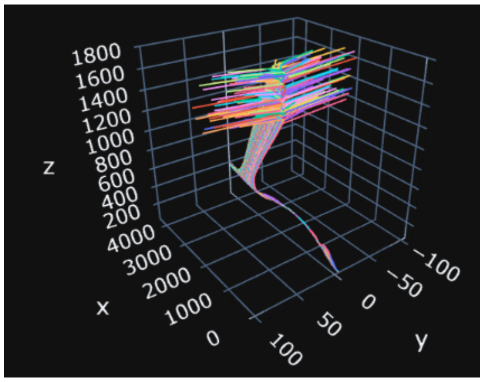


hidden_size=256

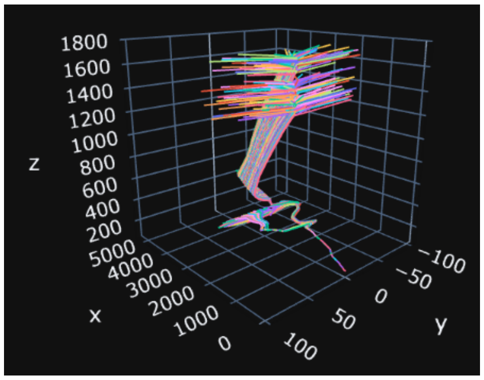
Figure 19: Trajectories generated by the DNN's for each where `num_layers=8`.



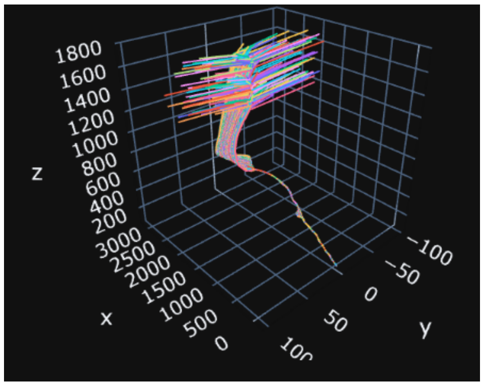
hidden_size=32



hidden_size=64

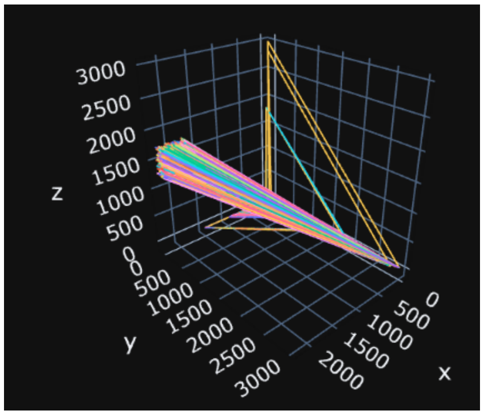


hidden_size=128

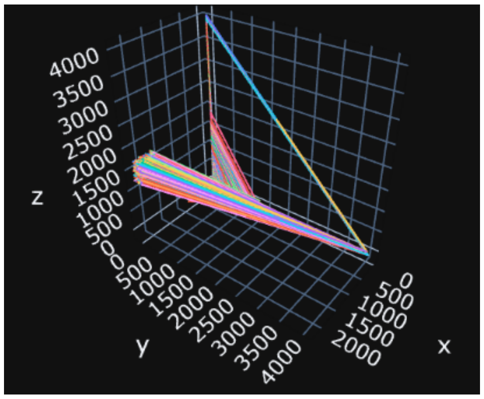


hidden_size=256

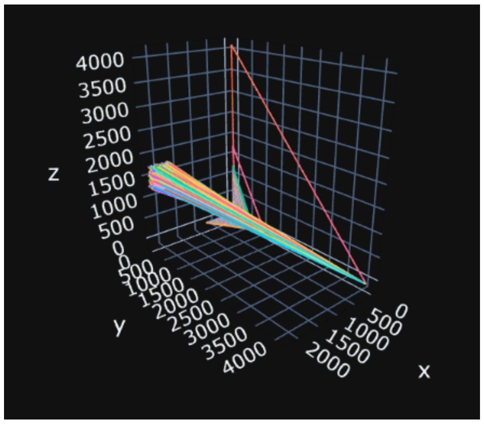
Figure 20: Trajectories generated by the DNN's for each where num_layers=16.



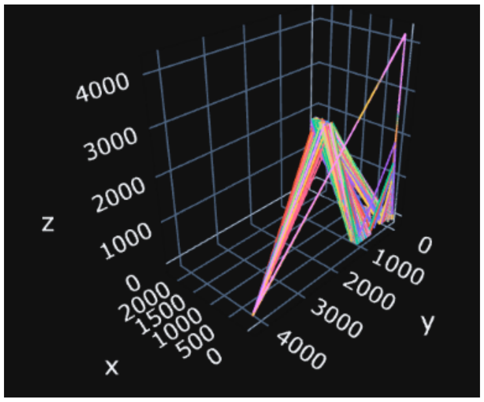
`sequence_length=1`



`sequence_length=3`



`sequence_length=10`



`sequence_length=25`

Figure 21: Trajectories generated by the LSTM's for each `sequence_length` value.