Project report: Vision-based UAV landing on a ship deck using deep learning

Ritwik Shankar*

I. Introduction

Landing a rotorcraft on the deck of a moving ship represents a significant challenge due to the unique and demanding conditions of the maritime environment. This challenge is increasingly important as maritime operations expand the use of unmanned aerial systems (UAS) for essential tasks such as surveillance, supply delivery, reconnaissance, and search-and-rescue missions. A safe and precise landing on a ship deck is essential to complete these missions effectively, especially in situations where immediate or frequent landings and takeoffs are necessary. However, ocean swells, unpredictable winds, and limited deck space create an inherently unstable and dynamic landing platform.

The movement of the ship—pitching, rolling, and heaving in response to sea conditions—compounds the complexity of achieving a successful landing. In human-piloted operations, the pilot must time their descent to match the ship's motion closely, an approach that relies on extensive experience, precise judgment, and rapid adaptability. The margin for error is extremely small, as the confined space on the deck allows little room for adjusting in case of misalignment or turbulence.

For autonomous systems, the challenge is even greater, as they lack the immediate intuition and adaptability of human pilots. To safely and efficiently achieve ship deck landings, UAS must be equipped with advanced control systems capable of anticipating and adjusting to the ship's movements in real-time. This requires not only sophisticated control algorithms but also robust perception systems that provide accurate, up-to-date information on the ship's position and orientation. Autonomous, vision-based solutions offer promising potential, enabling precise estimation of deck motion and real-time adjustments to improve landing accuracy, even in adverse conditions.

A. Problem Statement

The continuous pitching, rolling, and heaving of a ship's deck, driven by ocean conditions and environmental factors, introduces significant instability and unpredictability. These movements reduce the margin for error, requiring any landing approach to be highly precise. The turbulent airflow around the ship, coupled with deck movement, creates further complications for maintaining stable flight. The problem is particularly complex for autonomous systems, which must calculate descent, adjust for deck motion, and compensate for instability in real-time. To address this challenge, our approach utilizes vision-based platform estimation and an algorithm designed for precision landing on a dynamic surface. Testing of the algorithm leverages a custom-built Stewart platform capable of simulating the ship deck's 3-DOF motion, including roll, pitch, and heave, to replicate real-world conditions for controlled trials.

B. Literature Review

The autonomous landing of Unmanned Aerial Vehicles (UAV), capable of Vertically Takeoff and Landing (VTOL), is a widely researched topic with many control algorithms using different types of sensors such as IR-LEDs, LIDARs, RTK-GPS, and vision.

In vision-based methods for autonomous landing, the research started with identifying static geometrical patterns like T, H, circular, rectangular or a combination of them like in fiducial markers. In [1],"T" shape was detected by using adaptive threshold, whereas [2, 3] detected "H" shape. Using concentric circles, pose estimation is done by measuring the distances between the circle in [4, 5]. Fiducial markers provide more robust and accurate pose estimation along with high detection speed and hence are widely used for autonomous landing algorithms. ARTag and ARToolkit Plus was studied in [6]. [7, 8] explored the landing task by using AprilTag and AprilTag 2 was used by [9].

My main reference was the work done in Penn State University [10]. Their experiments were done in Maneuvering and Seakeeping (MASK) Basin at the U.S. Naval SurfaceWarfare Center Carderock Division (NSWCCD). In their work, they have used a MoCap (Motion capture system) for estimation of their work, and their predictions of the heave were made by an AutoRegressive (AR) model, very similar to ARIMA (autoregressive Integrated Moving Average), and have

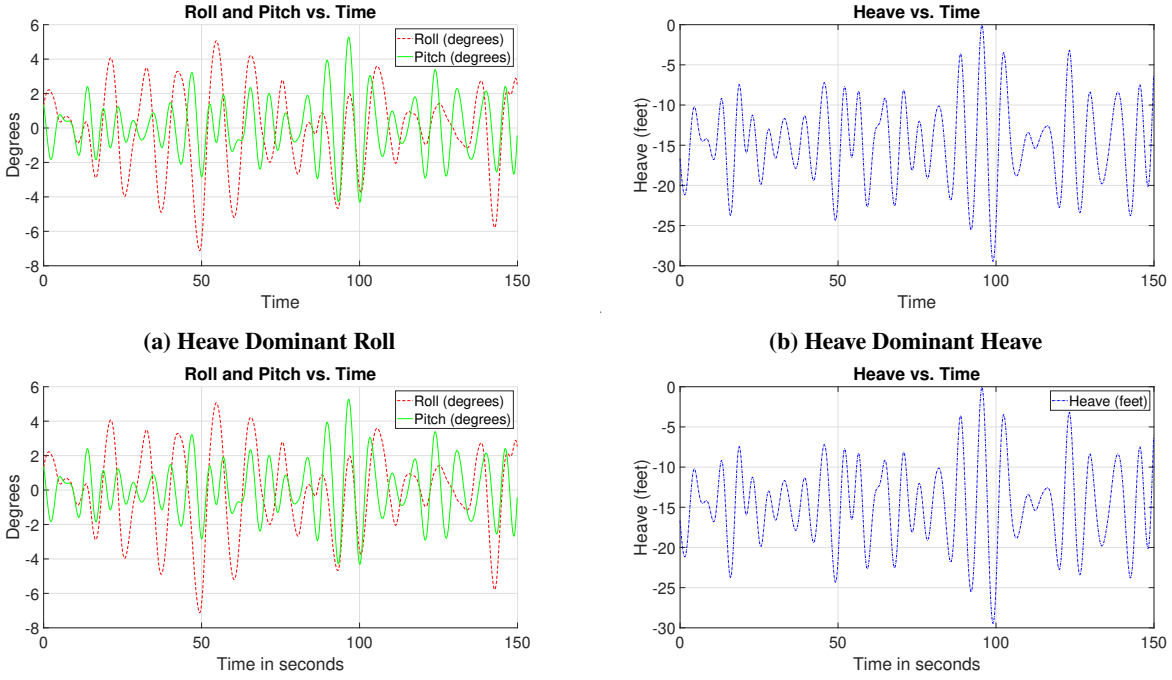
*Roll number: 210866, Under-Graduate Student, Dept. of Aerospace Engineering, IIT KANPUR

used a QP solver for optimal trajectory generation, moreover they have used the heave and Z velocity of the ship to make their predictions, while their work demonstrates a good proof of concept, it's not practical to get accurate measurements similar to a MoCap system, and the accuracy of the AR model decreases with increase in prediction time length, where predictions which were more than 2 seconds, got very bad. The ship dataset is from a US NAVAL frigate and is available to us. To make it more practical, I will be using position estimation of the ship using an onboard camera, stabilised by a gimbal, which would detect the ArUco marker kept on the ship deck. Further, I will be using Deep Neural Networks to get the prediction of the heave waves and finally integrate them with my MPC to generate optimal trajectory.

II. Ship Deck Simulation Platform

The Stewart platform was used for real-life emulation(Fig. 2) of the ship waves to test the algorithm. The Stewart platform is my undergraduate project, which I completed this semester, the report for which is available on the clickable link **"UGP report"**.

Experimentally taken ship data [11] was inputted into the platform to simulate ship conditions at sea state 4 [12]. The aim was to simulate roll(ϕ), pitch(θ), sway(Y) and heave(Z) motions to test the landing algorithm, as these are the most significant motions captured in the scone dataset; The SCONE dataset [11], acquired through experimentation, was slated for simulation. This dataset comprises two distinct types of data: one predominantly influenced by roll and the other by heave. It is structured as a MATLAB variable containing 36,000 data points, with a time interval of 0.05 seconds between consecutive points. Our research focuses on testing our algorithm using both the roll-dominated and Heave-dominated subset of this dataset ('SCONE_D3R_1' and 'SCONE_D3H_1'), where 'Rot_X' represents roll, 'Rot_Y' represents pitch, and Z represents heave. The deck is situated 15 feet (Z = -15 feet) above sea level (Z = 0). The data which is to be simulated is shown in Fig. 1.



(a) Heave Dominant Roll

(b) Heave Dominant Heave

(c) Roll Dominant Roll

(d) Roll Dominant Heave

Fig. 1 Comparison of Heave and Roll Dominant Parameters

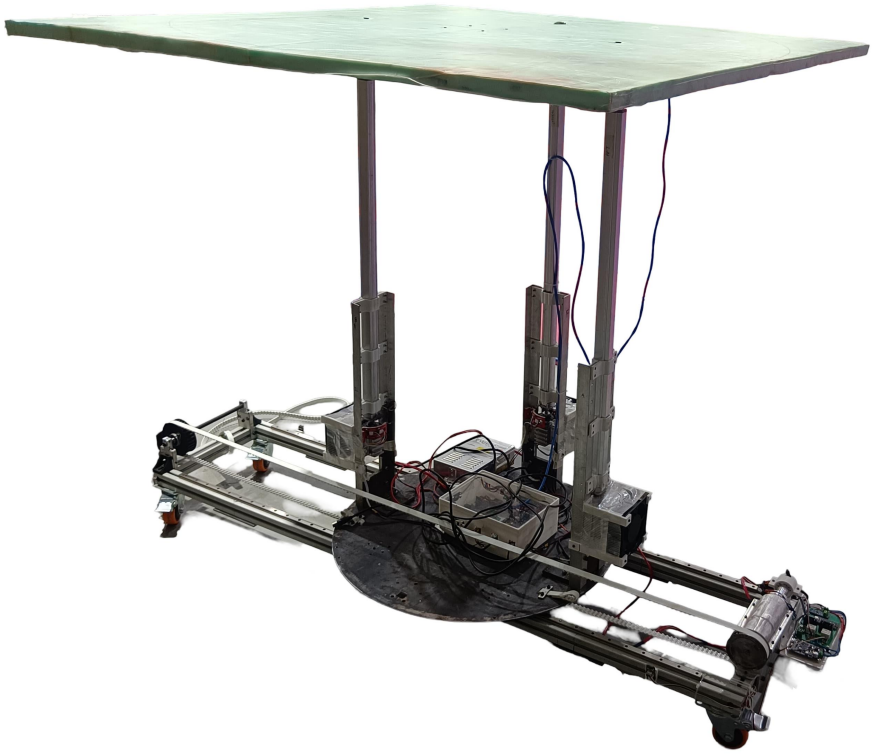


Fig. 2 Final Stewart Platform

III. Heave Prediction Model

I will be only using Z data history as the velocity is generally not available with good accuracy in real time. Please note that all my work is in the following OneDrive.

A. Selecting the best model

We had 5 datasets with 36000-time steps(at 20hz); four were used for training the ML model, and one was used for testing.

The following conditions were to be met to choose the best model for predicting heave motion:

- The predictions should take less than 0.04 seconds for the loop to run at 20Hz.
- The predictions should have negligible error for the first 3 seconds.
- The predictions should do reasonably well at mild outliers in the dataset.

Keeping these points in mind, the following models were tested:

1. ARIMA: AutoRegressive Integrated Moving Average

I started with a statistical model to understand the depth of how much a Deep Learning network would do better. The model was generated using Statsmodel's ARIMA model. A nonseasonal ARIMA model is classified as an "ARIMA(p,d,q)" model, where:

- **p** is the number of autoregressive terms,
- **d** is the number of nonseasonal differences needed for stationarity
- **q** is the number of lagged forecast errors in the prediction equation.

A model with order (3,1,1) was able to run at 20hz but had very poor results for prediction being 2-3 seconds in future,

shown In Fig 3.

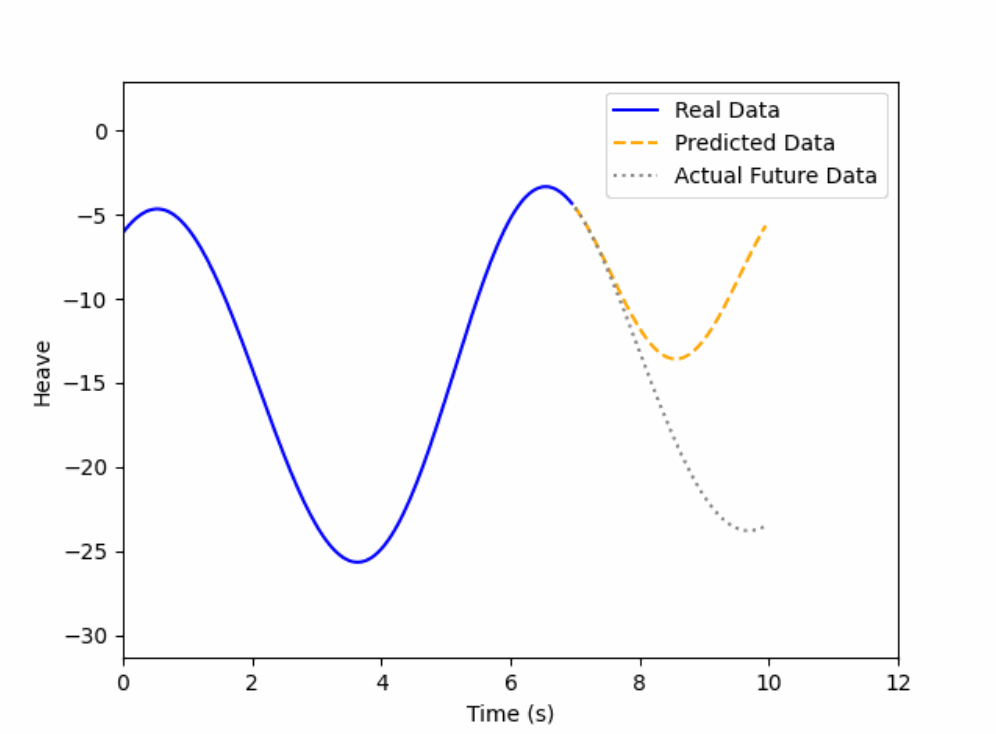


Fig. 3 Testing results for ARIMA

2. TCN: Temporal Convolution Network

A TCN uses CNN architecture to make temporal predictions; although it was a significant upgrade to ARIMA, doing convolution was computationally heavy, and the models which were able to run at 20hz did not do well.

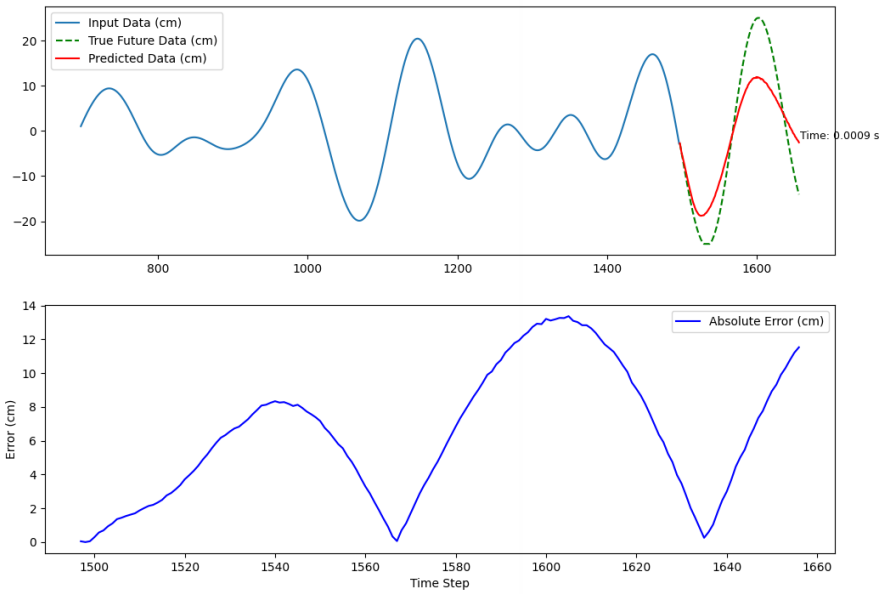


Fig. 4 TCN prediction

The prediction in 4 used a 40(800 data points @20Hz) seconds time window to give a prediction of 8 seconds(160 data points), and the model had a Number of Channels being 2, which defines the architecture of the TCN layers. The need for better models was obvious, and nothing was more obvious than LSTMS, which is famous for noting the long-term patterns of a model.

3. LSTM: Long Short-Term Memory

LSTM has by far the best performance out of all the models; TensorFlow was used to create a 3-layer network to see if good predictions could be made; the results are shown in Fig. 5

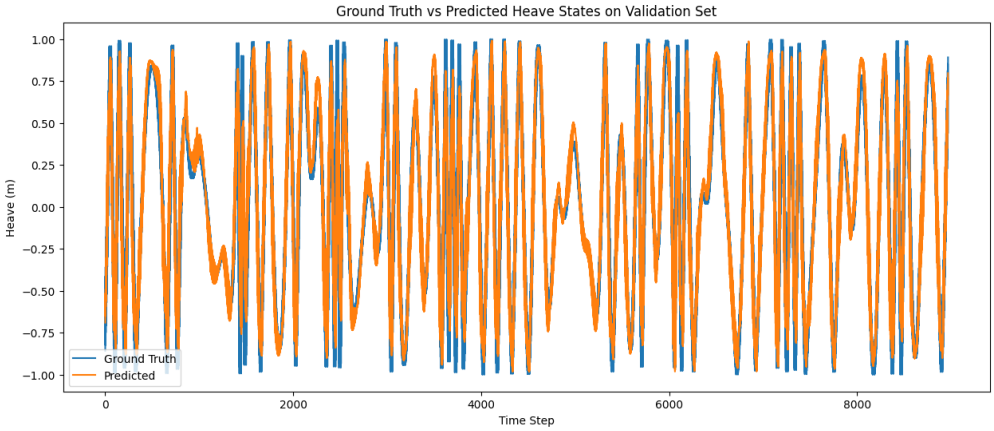


Fig. 5 LSTM results

The only problem was, LSTM are computationally heavy, and the tradeoff between accuracy and computation was just not working out, So a network with a similar principle like LSTM had to be used, with its complexity relaxed, the obvious choice after this finding was to turn to Gated Recurrent Unit (GRU) networks, which gave very promising results.

IV. GRU MODEL

A normalised (-1 to 1) version of the ship data was taken for predictions, as models tend to predict better for this range, while this will be multiplied by 25, as our algorithm will be tested on a platform moving ±25 cms. Now, the game was to optimally reduce the dimension of the data to properly encompass it into the GRU network; with much literature review, I could not find the most optimal way of doing it. A brute force method was devised, which will automatically train multiple models on similar configurations, test them out and give the test accuracy of them. Two NVIDIA RTX A6000 Graphics Cards, which had 48 Gbs of individual memory, were used for this. The hyperparameters for my model were

- Input sequence length
- Output sequence length
- Hidden state layer size and unit size
- learning rate
- Weight of loss given to initial seconds of prediction

A. Model Parameters

Two types of GRU models were made, The first received all the "input_sequence" length of data at every timestep and gave predictions of the future; this was computationally expensive as everything is calculated at every step, but gave good short-term accuracy, the second method was a normal sequential GRU, which took one data point at every time step and forwarded the cell states and gave the prediction, results for this were very bad for the initial time, but got good after "input_sequence" amount of time elapsed. A custom loss term was used to force continuity of the predictions and

focusing accuracy on initial prediction window.

$$\text{Total Loss} = w \times \text{MSE}_{\text{first } x\_seconds} + \text{MSE}_{\text{remaining}} + \lambda \times \text{Continuity Loss}$$

B. Model Architecture: Custom GRU Model with Dropout

The model is implemented as a multi-layer GRU network with the following components:

- **Input Layer:** Accepts sequences of length `sequence_length`, each consisting of univariate time series data.
- **GRU Layers:** A stack of GRU layers specified by the `hidden_sizes` parameter. Each GRU layer processes the output from the previous layer, enabling the model to capture complex temporal dependencies.
- **Dropout Layers:** Dropout with a probability of 20% is applied after each GRU layer to prevent overfitting.
- **Fully Connected Layer:** A linear layer maps the output of the last GRU layer to the desired `output_size`, representing the prediction horizon.
- **Activation Function:** A hyperbolic tangent (`tanh`) activation function constrains the output values within $[-1, 1]$.

C. Optimization and Training Loop

- **Optimizer:** Adam optimizer with learning rate scheduling.
- **Learning Rate Scheduler:** Exponential decay with a factor of 0.98 per epoch.
- **Gradient Clipping:** Gradients are clipped to a maximum norm (`max_norm=1000.0`) to prevent exploding gradients.
- **Epoch Loop:**
  - For each epoch:
    - Iterate over batches of data.
    - Compute the total loss.
    - Perform backpropagation and update the model parameters.
    - Accumulate and monitor the loss, and save the model if the loss decreases.

D. Results

With preliminary testing, it was found the models with layers [512, 512], [512,256,64], and [1024, 128] performed well on the data. More than 40 models were trained for over 300 epochs each. Approximately 9-10 days in aggregate were used to train these models(big thanks to EE's A6000 GPUs). The best models from the training are given in Fig 6. Here "Weight" refers to the weight multiplied by the error for the first "x_seconds". The time is given in time-steps (20hz), divide by 20 to get seconds.

Hidden States	Output Size	Input Size	Weight	X_seconds	Avg Error (3s)
[256, 128, 64]	120	1000	40	5	0.8209
[2048, 2048] - Seq	120	800	40	5	0.7401
[256, 128, 64]	120	1000	100	4	0.9148
[512, 512]	120	600	15	4	1.1159
[256, 128, 64]	120	1200	100	4	1.0743
[256, 128, 64]	120	1000	40	5	1.2755
[256, 128, 64]	120	1000	100	5	1.2402
[256, 128, 64]	120	1100	100	4	1.3412
[512, 512]	120	1200	100	5	1.2013
[256, 128, 64]	160	1000	20	5	0.8745

Fig. 6 Top models trained: error in cms

A snapshot of testing can be seen in Fig 7

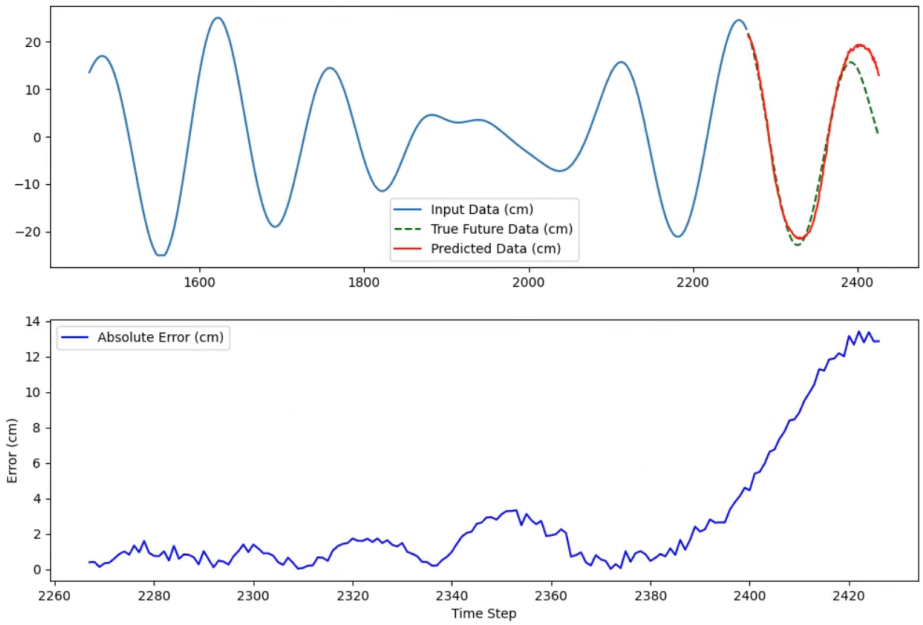
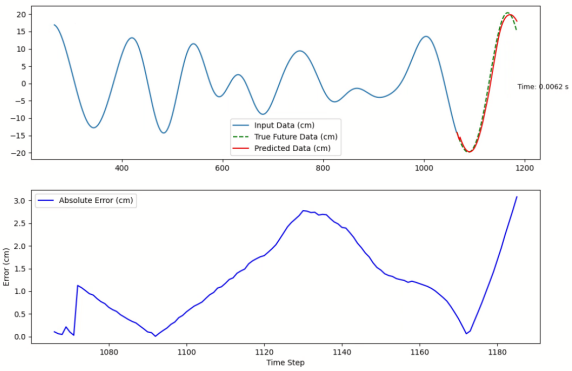
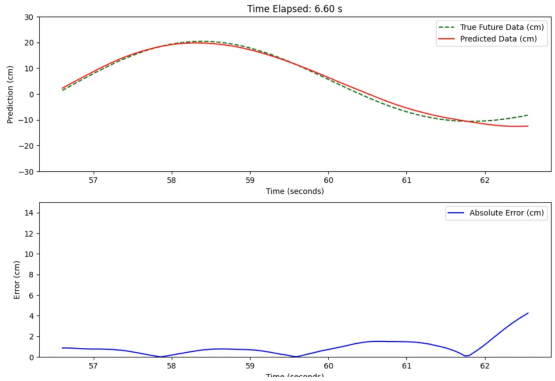


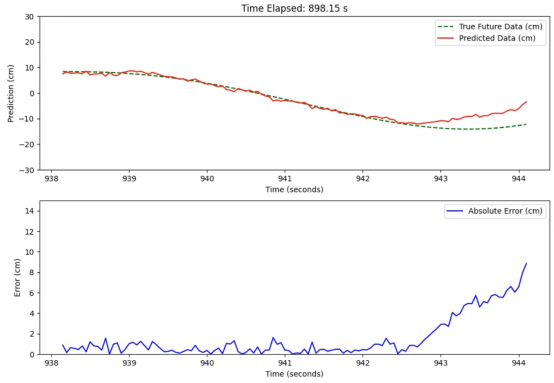
Fig. 7 Screenshot of prediction with [2048, 2048] sequential model



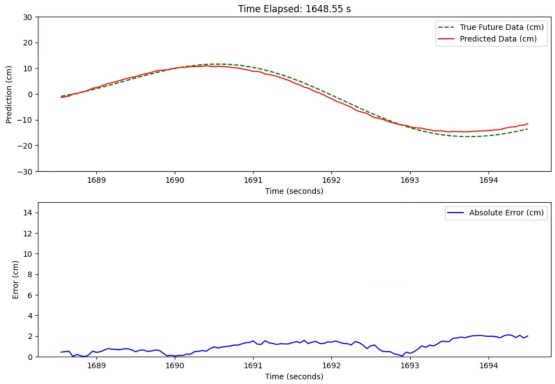
(a) GRU [256, 128, 64] with 800 input sequence



(b) GRU [512, 512] with 800 input sequence



(c) GRU [1024, 1024] with 1000 input sequence



(d) GRU [2048, 2048] with 1000 input sequence

Fig. 8 Different GRU Models Comparison

V. Vision and Landing algorithm

The overall system consists of two components: the Unmanned Aerial vehicle in the air and the landing platform on the ground. The UAV consists of all the electronics, actuators and sensors, where as the ground system consists of Deck simulation platform with the visual markers pasted on the top of it. The UAV used for the experiments is a self-assembled quad-rotor in the standard X frame and running PX4 autopilot, see Fig.11.

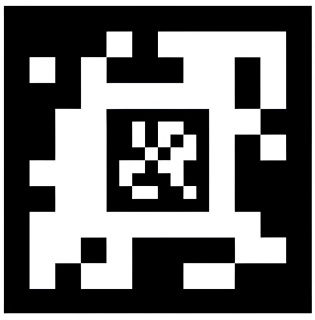


Fig. 9 Fractal marker composed of two internal markers [13]

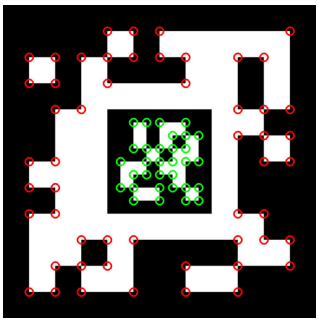


Fig. 10 Detected all visible internal markers of the fractal marker [13]

Fig. 11 UAV used for the experiments

A. Landing Pipeline

Two algorithms are operating in parallel within this module. One is the Landing algorithm, and the other is a Horizontal (X, Y) motion controller. This horizontal controller handles the trajectory planner to guide the UAV to hover above the X and Y locations of the landing zone. This is handled by the PX4 position controller. Currently, I am giving relative pose as position error for the PX4 position controller. The landing algorithm takes care of the descent phase of the UAV. It handles the generation of the optimal trajectory generation for the drone using a constrained optimization using the cvxpy library.The UAV will descend if roll and pitch are both less than 5 degrees, otherwise it will hover to get to that position. The autonomous landing protocol is described in V.B.

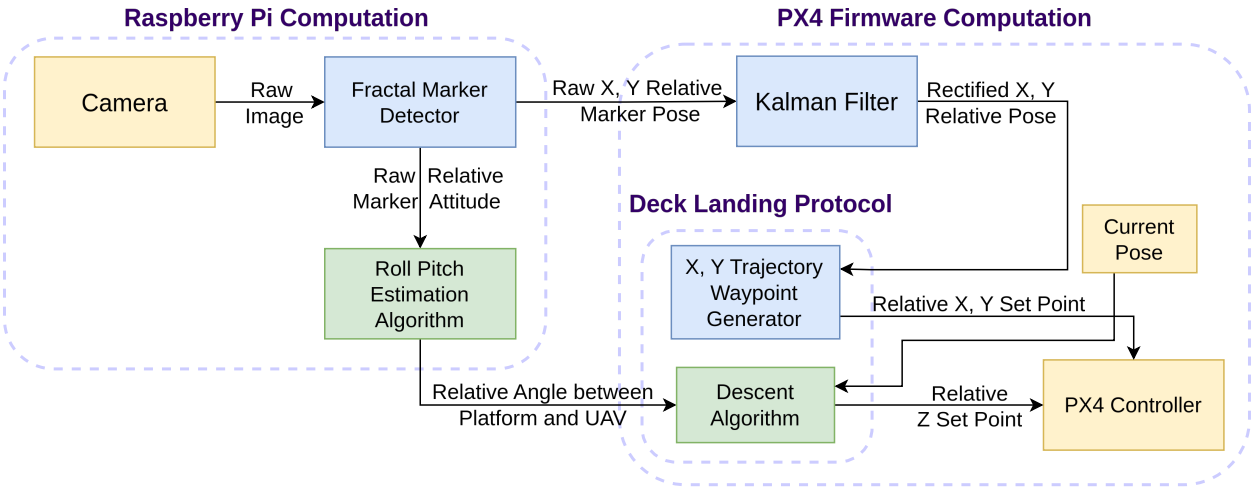


Fig. 12 Autonomous vision-based landing pipeline

B. Model Predictive Control (MPC) for Altitude Regulation

The implemented MPC controller is designed to regulate the altitude of a UAV by predicting its future states over a defined prediction horizon and optimizing its control inputs. This controller operates in two distinct phases: *tracking* and *landing*. The detailed description of the control logic is as follows:

1. Tracking Phase

The goal of the tracking phase is to follow a reference trajectory, which is generated using the predicted movement of the landing platform. The reference trajectory (z_{ref}) is updated in real-time, incorporating a fixed offset to maintain a safe tracking altitude.

Optimization Problem At each time step, the optimization problem is defined as:

$$\min_{z,v,u} \quad 10\sum_{k=1}^{N}(z_k - z_{\text{ref},k})^2 + 0.1\sum_{k=1}^{N-1}u_k^2 + 10(z_2 - z_{\text{ref},2})^2 + 10(z_N - z_{\text{ref},N})^2 \tag{1}$$

$$\text{subject to} \quad z_{k+1} = z_k + v_k \cdot \Delta t, \quad \forall k \in [1, N-1] \tag{2}$$

$$v_{k+1} = v_k + u_k \cdot \Delta t, \quad \forall k \in [1, N-1] \tag{3}$$

$$z_1 = z_{\text{current}}, \quad v_1 = v_{\text{current}} \tag{4}$$

$$v_{\min} \le v_k \le v_{\max}, \quad \forall k \in [1, N-1] \tag{5}$$

where:

- z_k, v_k, u_k are the altitude, vertical velocity, and vertical acceleration (control input) at step k.
- N is the prediction horizon.
- Δt is the control loop time step (0.05 seconds in this implementation).

- z_{ref} is the reference trajectory for altitude.
- $v_{\min}$, $v_{\max}$ are the velocity constraints.

Control Loop The controller solves this optimization problem at 20 Hz, publishing the optimal velocity (v_2) as the control setpoint for the UAV.

2. Landing Phase

The landing phase ensures the UAV descends smoothly to the landing platform while satisfying safety constraints.

Optimization Problem The landing phase is initiated when the tracking phase is complete. A linear descent trajectory is generated from the current altitude to a target altitude slightly above the landing platform (z_{landing}). The optimization problem is defined as:

$$\min_{z,v,u} \quad 10\sum_{k=1}^{N}(z_k - z_{\text{ref},k})^2 + 0.1\sum_{k=1}^{N-1} u_k^2 + 10(z_{\text{landing},k} - z_{landing})^2 + 20v_{landing}^2 \tag{6}$$

$$\text{subject to} \quad z_{k+1} = z_k + v_k \cdot \Delta t, \quad \forall k \in [1, N-1] \tag{7}$$

$$v_{k+1} = v_k + u_k \cdot \Delta t, \quad \forall k \in [1, N-1] \tag{8}$$

$$z_1 = z_{\text{current}}, \quad v_1 = v_{\text{current}} \tag{9}$$

$$z_k \geq z_{\min}, \quad v_{\min} \leq v_k \leq v_{\max}, \quad \forall k \in [1, N-1] \tag{10}$$

$$z_N = z_{\text{landing}}, \quad v_N = 0 \tag{11}$$

where:
- z_{landing} is the target altitude for landing.
- $z_{\min}$ is the minimum safe altitude.

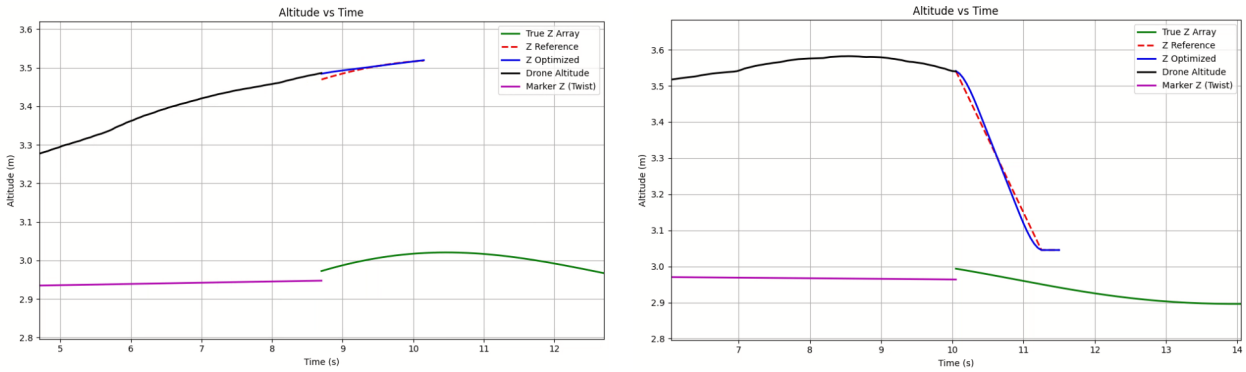
Landing Conditions The UAV checks if the absolute difference between the current altitude and the platform altitude is within a predefined tolerance. Once the conditions are met, the UAV disarms and the landing process are completed.

3. Visualization and Debugging

The controller includes a real-time visualization tool that plots:
- **True Z Array:** The predicted altitude of the platform.
- **Z Reference:** The reference trajectory.
- **Z Optimized:** The optimized altitude trajectory from the MPC solution.
- **Drone Altitude:** The current altitude of the UAV.
- **Marker Z:** The altitude of the platform's marker (for debugging).

C. Results

A gazebo simulation was set up with ROS1 and GAZEBO-SITL(software in the loop) using PX4's MAVROSE PROTOCOL. Iris drone was set up as the test bed for my algorithm, and the necessary changes were done in both the modules of PX4, and a ROS1 package was created, which would give vision the prediction and computer vision data by using a Raspberry Pi. I added a 6DOF platform in my simulation from GITHUB, which was my simulating platform in Gazebo. The platform was initially RL-based, but I converted the IK from RL to the general IK of the platform, which was able to provide 1.2m of heave and 20 degrees of roll and pitch. Right now, the algorithm is programmed to land after tracking for 10 seconds. The horizon window for the QP-solver is 30 time steps (or 1.5 seconds).

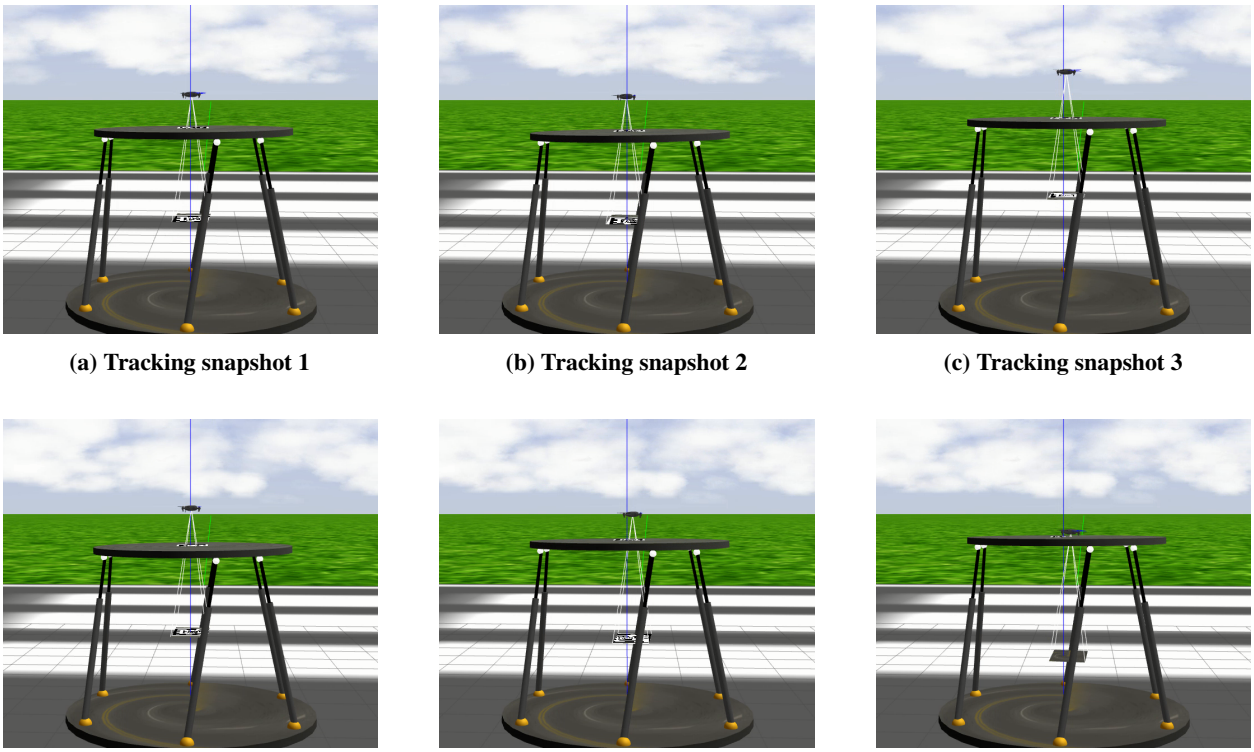


(a) Tracking stage **(b) Landing stage**

Fig. 13 Comparison of optimized trajectory in tracking and landing phases

Here, "True Z Array" is the prediction of the platform, "Z reference" is the reference trajectory which the drone should be following, from this the optimiser calculates an Optimised trajectory to take the drone to follow the reference trajectory, Marker Z (twist) is the history of the platform. PS: sorry for this naming

The snapshots are shown in Fig 14. Video of this landing is available at Heave_Landing.mp4



(a) Tracking snapshot 1 **(b) Tracking snapshot 2** **(c) Tracking snapshot 3**

(d) Landing initiated snapshot 4 **(e) Landing proceeding snapshot 5** **(f) Drone disarmed snapshot 6**

Fig. 14 Landing sequence snapshots

The history of the plots of the platform and the drone are given in Fig. 15. Landing is triggered at time step: 23 seconds.

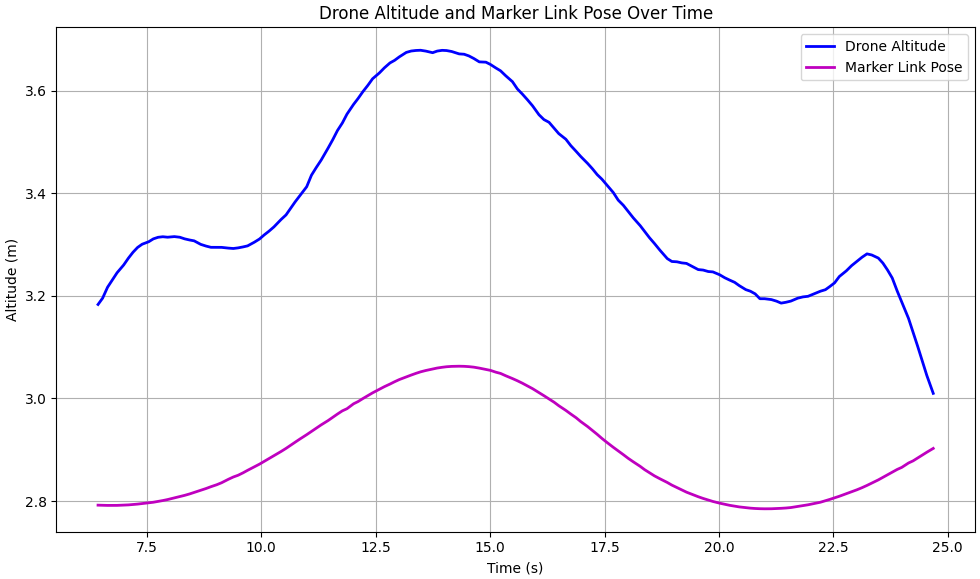


Fig. 15 Altitude plot of landing

VI. Conclusion

This report presents a simulated validation experiment for my algorithm to land on a heaving platform. The real-life testing of this algorithm couldn't be done on time, but that will be completed in under 2 weeks, as it took me more time than I expected to set all this up on a Jetson Nano and my drone. The test-bed system for this algorithm is already set at the Helicopter building (Fig .16), IIT Kanpur, where I completed this project. Real-life algorithm validation on a Quadcopter has already been done on a stable platform, and the work to land a quadcopter using this algorithm by getting true prediction values is almost completed. This will be followed by improving the GRU model by incorporating emulator data, training it to get better accuracy, and finally landing on predictions made on board.

VII. Acknowledgments

I would sincerely like to thank my main guide, Dr Abhishek (Professor, Dept. of Aerospace Engineering), for giving me the resources to work on this problem, Mr Chiranjeev Pranchand (PHD, EE), this experiment is a part of his thesis. I would also like to thank the people of the Helicopter lab for constantly providing 3D-printed parts and logistical support.

References

[1] Xu, G., Zhang, Y., Ji, S., Cheng, Y., and Tian, Y., "Research on computer vision-based for UAV autonomous landing on a ship," *Pattern Recognition Letters*, Vol. 30, No. 6, 2009, p. 600–605. https://doi.org/10.1016/j.patrec.2008.12.011, URL http://dx.doi.org/10.1016/j.patrec.2008.12.011.

[2] Fucen, Z., Haiqing, S., and Hong, W., "The object recognition and adaptive threshold selection in the vision system for landing an Unmanned Aerial Vehicle," *2009 International Conference on Information and Automation*, IEEE, 2009. https://doi.org/10.1109/icinfa.2009.5204904, URL http://dx.doi.org/10.1109/icinfa.2009.5204904.

[3] Fan, Y., Haiqing, S., and Hong, W., "A vision-based algorithm for landing unmanned aerial vehicles," *2008 International Conference on Computer Science and Software Engineering*, Vol. 1, IEEE, 2008, pp. 993–996.

[4] Benini, A., Rutherford, M. J., and Valavanis, K. P., "Real-time, GPU-based pose estimation of a UAV for autonomous takeoff and landing," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 3463–3470.

[5] Lange, S., Sunderhauf, N., and Protzel, P., "A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments," *2009 international conference on advanced robotics*, IEEE, 2009, pp. 1–6.

Fig. 16 Real-life test bed for algorithm validation

[6] Fiala, M., "Comparing ARTag and ARToolkit plus fiducial marker systems," *IREE International Worksho on Haptic Audio Visual Environments and their Applications, 2005.*, IEEE, ???? https://doi.org/10.1109/have.2005.1545669, URL http://dx.doi.org/10.1109/have.2005.1545669.

[7] Li, Z., Chen, Y., Lu, H., Wu, H., and Cheng, L., "UAV Autonomous Landing Technology Based on AprilTags Vision Positioning Algorithm," *2019 Chinese Control Conference (CCC)*, IEEE, 2019. https://doi.org/10.23919/chicc.2019.8865757, URL http://dx.doi.org/10.23919/chicc.2019.8865757.

[8] Wang, Z., She, H., and Si, W., "Autonomous landing of multi-rotors UAV with monocular gimbaled camera on moving vehicle," *2017 13th IEEE International Conference on Control amp; Automation (ICCA)*, IEEE, 2017. https://doi.org/10.1109/icca.2017.8003095, URL http://dx.doi.org/10.1109/icca.2017.8003095.

[9] Wang, J., and Olson, E., "AprilTag 2: Efficient and robust fiducial detection," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016. https://doi.org/10.1109/iros.2016.7759617, URL http://dx.doi.org/10.1109/iros.2016.7759617.

[10] Hendrick, C., Nicholson, D., Jaques, E., Horn, J., Langelaan, J., and Sydney, A., "Scaled Experiments in Flight Control Design for Autonomous Landing in High Sea States," 2022. https://doi.org/10.2514/6.2022-3280.

[11] Schwartz, A., "Systematic Characterization of the Naval Environment (SCONE) - Standard Deck Motion Data for a Generic Surface Combatant," [Online]. Available: http://www.navalengineers.org/Symposia/PastSymposia/Launch-and-Recovery-2016/Program/Schwartz, 2015.

[12] Owens, E. H., *SEA CONDITIONSDouglas scale; Peterson scale; Sea stateSea conditions*, Springer US, New York, NY, 1984, pp. 722–722. https://doi.org/10.1007/0-387-30843-1_397, URL https://doi.org/10.1007/0-387-30843-1_397.

[13] Romero-Ramire, F. J., Muñoz-Salinas, R., and Medina-Carnicer, R., "Fractal Markers: A New Approach for Long-Range Marker Pose Estimation Under Occlusion," *IEEE Access*, Vol. 7, 2019, pp. 169908–169919. https://doi.org/10.1109/ACCESS.2019.2951204.