University of California Santa Cruz
Baskin School of Engineering
Electrical and Computer Engineering Department

Small Scale UAV with a Recurrent Neural Network

Ana Villa, Cole Schreiner, Austin Tchang & Marlon Brewer

ECE 263/210
Instructors: Gabriel Elkaim & Jason Eshraghian
8 March 2026

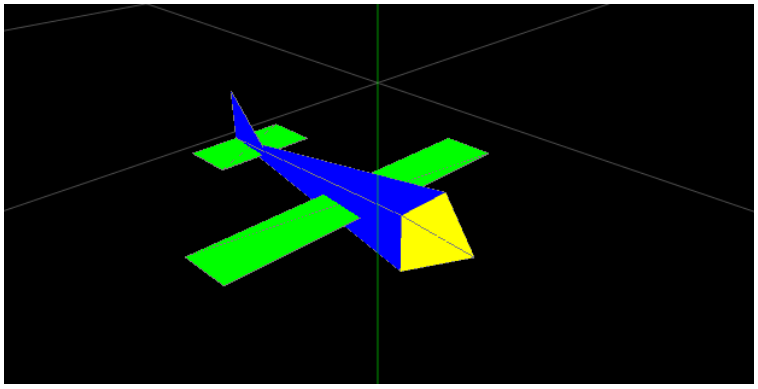

Table of Contents

1 Abstract

This project implements a neural network based autopilot for a small scale UAV using behavioral cloning to replicate classical control laws. While traditional flight systems rely on nested PID controllers for successive loop closure, this approach utilizes a Recurrent Neural Network (RNN) with a Long Short Term Memory (LSTM) architecture as a unified inference engine. Developed in PyTorch, the model maps a 15 feature input vector to four continuous control outputs. Input normalization and Tanh output bounding are applied to ensure stable gradient descent across disparate units. The training methodology follows a supervised imitation learning framework where the model learns by observing an expert PID teacher. This modular lifecycle consists of offline training on expert data, live inference integrated via an interface bridge, and online updates using a post flight data buffer to ensure environmental adaptation.

2 Background

The control of fixed-wing Unmanned Aerial Systems (UAS) has traditionally been rooted in linear control theory, specifically the architecture of successive loop closure. As established by Beard and McLain (2012), this framework decomposes the complex, high-degree-of-freedom states of an aircraft into nested PID (Proportional-Integral-Derivative) loops. Inner loops manage fast-acting attitude dynamics, such as roll and pitch, while outer loops govern slower navigational objectives like altitude and course tracking. While this remains the industry standard, it requires extensive manual tuning of gains and can struggle with the non-linear aerodynamic coupling and environmental disturbances, such as crosswinds, encountered in real-world flight.

Recent research has shifted toward replacing these classical controllers with Artificial Neural Networks (ANNs) acting as direct inference engines. Unlike traditional PID controllers

that require an explicit mathematical model of aerodynamic coefficients, ANNs function as universal approximators, capable of learning flight physics directly from data. This project utilizes an Imitation Learning (IL) framework, specifically Behavioral Cloning, where a neural network is trained to map sensory inputs to control surface deflections by observing an "expert" PID teacher. As noted by Shukla et al. (2020), while basic supervised learning can mimic stable flight, standard architectures often struggle to generalize across transitioning maneuvers (e.g., moving from a straight-line to a turn), which can lead to instability or divergence.

To address these limitations, this study adopts a Recurrent Neural Network (RNN) architecture using Long Short-Term Memory (LSTM) units. Standard Deep Neural Networks (DNNs) have shown high accuracy in trajectory tracking for holonomic vehicles (Kownacki et al., 2025), yet fixed-wing UAS present unique temporal challenges. The LSTM addresses these by maintaining an internal "hidden state" that serves as a memory buffer. This allows the controller to internalize the aircraft's momentum and the persistent nature of environmental disturbances. For instance, the LSTM can "sense" a constant lateral wind and learn to maintain a "crab angle", a steady-state heading offset, to keep the aircraft on its intended path, a task that traditionally requires complex integral logic.

The effectiveness of this learning-based approach is further enhanced by Feature Engineering. By moving beyond raw global coordinates and utilizing relative geometric features, such as wrapped heading error and normalized cross-track error (e_p), the model becomes a generalized path-following engine. This allows the system to remain robust across different waypoints and wind conditions. By combining the temporal strengths of LSTM architectures with the geometric precision of path-following laws, this project demonstrates a shift toward more adaptive, autonomous flight control systems capable of operating in unstructured and noisy environments.

3 Line Following

The line-following algorithm implemented in this project transitions the UAV from simple waypoint-heading logic to a robust geometric tracking system. The primary objective is to minimize the *Cross-Track Error (e_p)*, which is defined as the orthogonal distance between the aircraft's current position and the desired infinite straight-line path. The path itself is defined by a 3D origin point $\mathbf{r}$ and a unit direction vector $\mathbf{q}$. For lateral guidance, the system focuses on the North-East plane, where the course of the path, χ_q, is calculated as the *atan2* of the East and North components of the direction vector:

$$\chi_q = \text{atan2}(q_e, q_n)$$

This provides a constant reference heading that the aircraft would follow if it were perfectly centered on the line.

To determine how far the UAV has drifted, the cross-track error e_p is calculated using the relative position vector from the path origin to the aircraft. As implemented in the '*followStraightLine*' method, the formula:

$$e_p = -\sin(\chi_q)(p_n - r_n) + \cos(\chi_q)(p_e - r_e)$$

provides a signed distance: a positive value indicates the aircraft is to the right of the path, while a negative value indicates a drift to the left. This geometric derivation is essential because it allows the controller to understand its position relative to a trajectory rather than just a static coordinate.

Rather than simply turning directly toward the path, which could lead to aggressive overshooting or unstable oscillations, the system employs a nonlinear sliding-surface steering law to generate a commanded course χ_c. This law is governed by the equation:

$$\chi_c = \chi_q - \chi_\infty \frac{2}{\pi} \tan^{-1}(k_{\text{path}} e_p)$$

.

Here, χ_∞ represents the maximum approach angle which is set to 60° in the 'VCLC.py' configuration to ensure the UAV intercepts the path at a stable, predictable angle. The gain k_{path} controls the aggressiveness of this transition; as the error e_p approaches zero, the tangent function smoothly aligns the aircraft's commanded course with the path course χ_q.

In the Neural Network implementation, this geometric logic is integrated through the feature extraction process in 'interface_bridge.py'. Instead of requiring the LSTM to derive the concept of a "line" from raw GPS coordinates, the system provides the normalized cross-track error as a direct input feature. By providing e_p alongside body-frame rates and aero-angles, the network learns to associate specific displacement errors with the exact aileron and rudder deflections needed to maintain the sliding surface. This is particularly effective under wind disturbances; the LSTM's temporal memory allows it to internalize the steady-state "crab angle" required to counteract lateral wind drift, maintaining the path more fluidly than a standard PID loop.

4 RNN Learning Methods

4.1 Model Architecture

The neural network autopilot is implemented as a specialized type of recurrent neural network known as Long Short-Term Memory (LSTM). The neural network is trained by imitating an expert PID controller. Rather than separating flight control into nested PID loops described by Beard and McLain (2012), the model acts as a unified inference engine that maps the time associated flight states to flight control commands. The complete architecture of LSTM is implemented by PyTorch.

Input Feature Vector

The model receives a 15-dimensional input feature vector at each timestep, composed of three reference commands and twelve state variables. The reference commands consist of the course angle, altitude, and airspeed. The twelve state variables consist of the aircraft's attitude angles (roll, pitch, yaw), velocities (u, v, w), angular rates (p, q, r), and inertial positions (north, east, altitude). These input variables span heterogeneous physical units, so all inputs are normalized to zero mean and unit variance during preprocessing to ensure stable gradient flow during training.

Network Architecture

The core of the model is the LSTM architecture that operates on sequential temporal input observations. An LSTM architecture is chosen over vanilla RNNs due to its unique gating mechanism which are the forget, input, and output gates, these characterizes it's ability to selectively retain information across time-steps. Along with the gates, LSTMs have a memory cell where information is added or removed from the cell that holds long term memory. A general sequence the network will undergo is that the forget gate decides what to remove from the previous cell state, the input gate decides what new information to add to the memory cell, the cell state is updated based on the results of the forget and input gates, and the output gate determines the final output based on the current cell state.

The LSTM layers are followed by fully connected hidden layers with Rectified Linear Unit (ReLU) activations, introducing non-linearity while preserving gradient flow. In the final output layer, a Tanh activation function is applied to bound the output values to the interval [-1,1], reflecting flight control limits with the exception of the throttle output channel with it being bounded to the interval [0,1]. The four output channels correspond to throttle, aileron, elevator, and rudder flight controls, effectively replacing all four PID loops of the classical successive loop closure architecture.

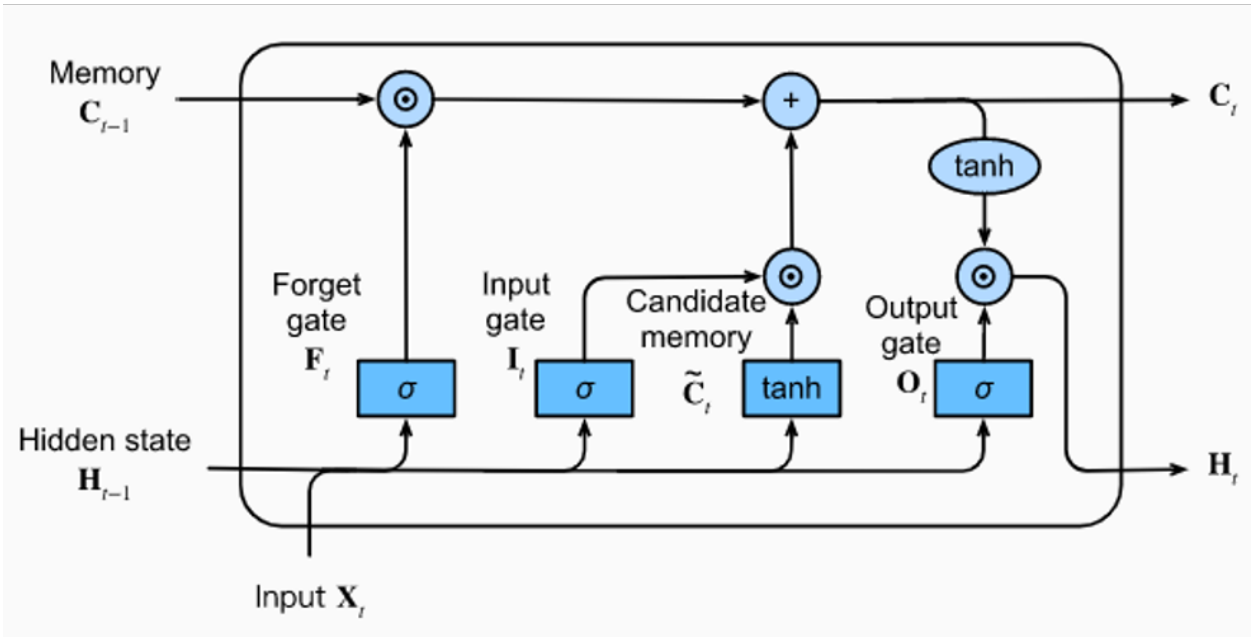

Figure 1: LSTM Architecture

4.2 Training Methodology

Dataset

The training dataset is generated by recording the full state-action trajectories of the expert PID controller operating within the ECE263 fixed-wing UAV simulation environment. For each simulation timestep, the 15-dimensional input feature vector are logged to a structured dataset (Dataset.csv). The dataset captures a range of flight conditions including straight-and-level cruise, altitude changes, course corrections, and wind-disturbed flight, ensuring that the training distribution spans the behavioral envelope the model is expected to reproduce.

Loss Function and Optimization

The model is training under a imitation learning objective by minimizing Mean Squared Error (MSE) between the network's predicted control outputs and the expert PID labels:

$$L = (1/N) \sum (y - \hat{y})^2$$

where y is the actual observed value given by the expert PID and $\hat{y}$ is the predicted value given by the model at each timestep i. MSE is an appropriate use here because the control

outputs are continuous-valued and the objective is to minimize the magnitude of deviation between the model and PID controller across all four output channels simultaneously. The first 80 percent of the collected dataset is used for training and the remaining 20 percent is reserved for validation.

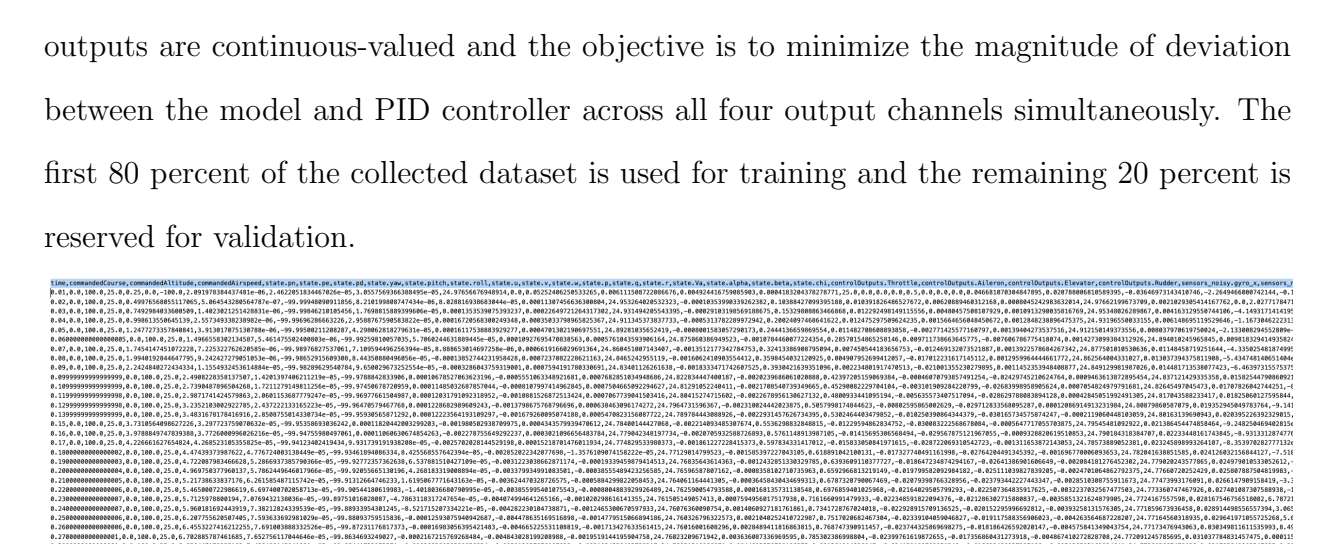

Figure 2: Data saved from UAV PID controlled simulation run

4.3 Training Lifecycle

The system follows a modular end-to-end lifecycle consisting of three distinct phases:

Offline Training: Initial weights are created by training the model on expert data exported from PID-driven flight simulations.

Live Inference: The trained model is integrated into the ECE263 framework via *interface_bridge.py*, which replaces the *UpdateControlCommands* logic of the classical autopilot with a forward pass through the LSTM. During inference, the model operates in real-time, inputting the current state vector and outputting four control commands.

Online Updates: The system features a "post-flight learning" or continual learning mechanism where imitation data is collected in a buffer during flight and used to update model weights upon every simulation reset.

The three phase design splits the initial supervised imitation phase from the adaptive online learning phase, allowing the system to be pre-trained on quality PID demonstrations and refined through self-generated flight experience. The result is a learning-based autopilot that improves with flight time while remaining stable due to its expert PID teacher during initial training.

5 Testing and Evaluation

The testing of the autopilot neural network follows a two-stage validation process. We perform and initial verification (Figure 3) to ensure the model can successfully imitate the teacher's control laws. Second, we conduct a closed-loop performance comparison to evaluate how the NN manages live flight dynamics compared to the original PID controller.

5.1 Neural Network Training Validation

The first stage of testing evaluates how well the Recurrent Neural Network is at imitating the PID controller. This is an offline learning test performed using an initial training method in *train.py* and the dataset recording in *dataset.py*. The dataset is built from exported simulator trajectories, where each sample contains a 15-feature input vector including heading error, cross-track error, and altitude error mapped to the 4 control commands: throttle, aileron, elevator, and rudder.

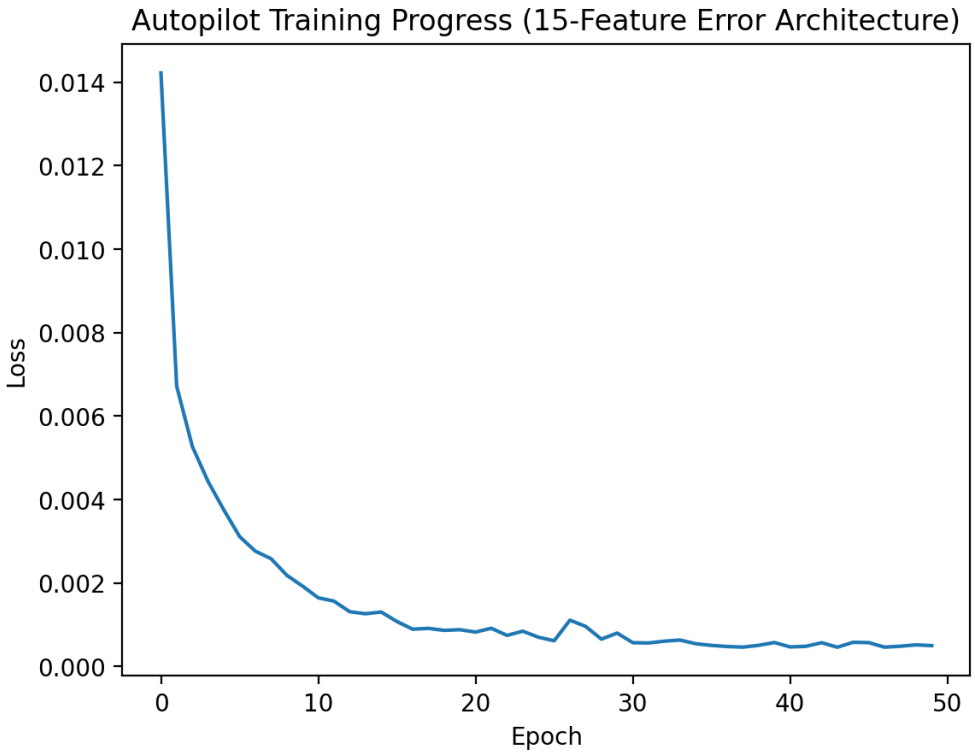

Figure 3: Training loss of the LSTM autopilot over 50 epochs converging as the model minimizes Mean Squared Error (MSE) against the PID teacher.

As shown in Figure 3, the training process demonstrates a reduction in Mean Squared Error (MSE), indicating that the model has converged to a stable mapping. A successful accuracy test proves that the LSTM memory window effectively captures the short-term temporal behavior required to clone the same corrections the expert controller would produce.

5.2 PID vs. RNN Path Following Error

The second stage is a closed-loop performance evaluation. The goal is to determine if the learned controller can guide the aircraft along a desired path with equal or improved performance compared to the PID baseline. For this experiment, two simulator runs were exported (*testPID.csv* and *testNN.csv*) and analyzed.

The comparison focuses on two critical behaviors: lateral path tracking and altitude regulation. Path tracking is measured using Cross-Track Error (CTE), defined as the

perpendicular distance from the aircraft to the desired straight-line centerline. Altitude performance is measured by the deviation from the 100-meter target altitude.

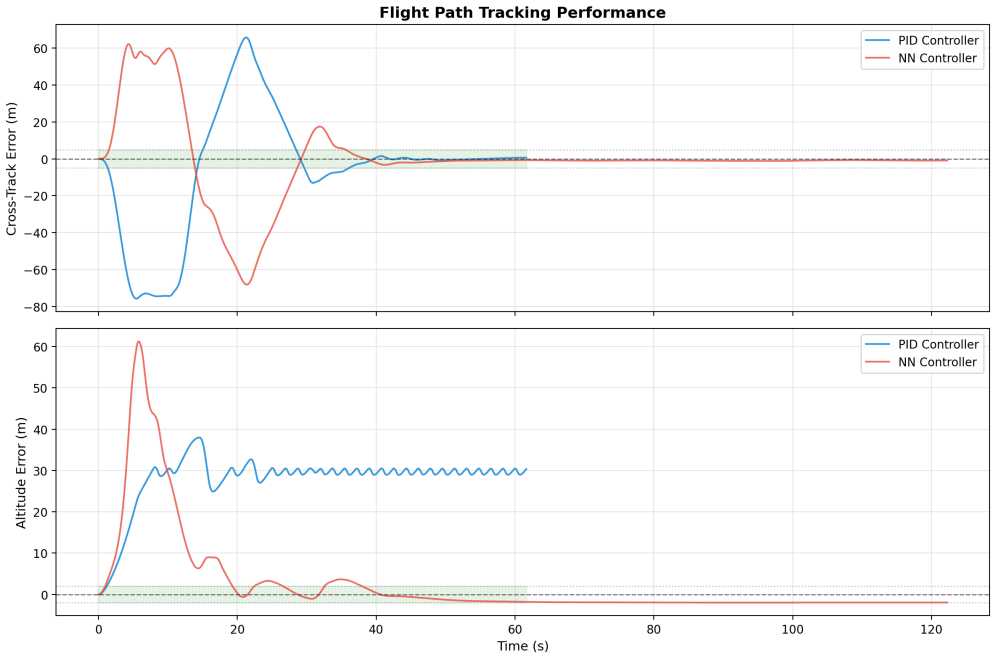

Figure 4: Cross-track and altitude error histories comparing the PID and NN controllers during live flight simulations.

The testing script computes Mean Absolute Error (MAE), Root Mean Square (RMS) error, and the percentage of time the aircraft remains within a ±5 meter cross-track tolerance band. As seen in Figure 4, the NN demonstrates a similar recovery and settling time when responding to the same path deviation.

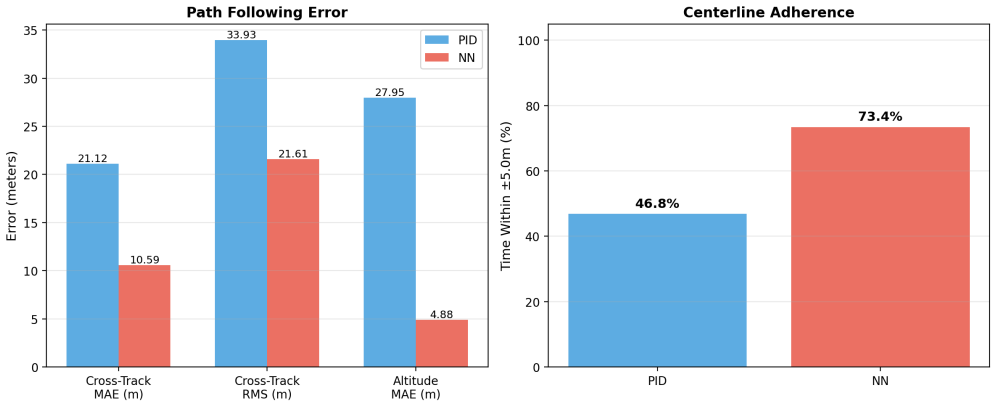

Figure 5: Path following performance: highlighting the NN's improvement in path adherence and altitude maintenance.

As illustrated in Figure 5, the NN typically outperforms the PID in several key areas. The benchmark results generated by the evaluation harness are summarized in Table 1.

Table 1: Performance comparison results from exported simulator trajectories.

Metric	**PID Controller**	**NN Controller**	**Improvement**
Cross-Track MAE (m)	21.12	10.59	**49.8%**
Cross-Track RMS (m)	33.93	21.61	**36.3%**
Altitude MAE (m)	27.95	4.88	**82.5%**
Time within ±5 m (%)	46.84%	73.44%	**+26.6%**

The results indicate that the NN does not merely copy the PID; it discovers nonlinear relationships between error signals and control inputs to refine its strategy. By optimizing the mapping from state to control, the NN achieves significantly lower tracking error and returns to the desired centerline much faster compared to the PID teacher.

5.3 Generalization Under Wind Perturbations

To evaluate the robustness of the neural network, we conducted a third trial for the NN autopilot. In this experiment, the expert PID teacher was recorded flying against a constant wind force originating from the left. The RNN was trained on this specific dataset and then attempted to fly the UAV against a wind force originating from the right without additional training.

5.3.1 Iterative Training

The model was tested at three distinct stages of training: 1 epoch, 20 epochs, and 225 epochs to observe how increased learning sessions to the teacher's data affected its ability to generalize when given unseen environmental conditions.

- **1 Epoch:** At the earliest stage of learning, the model exhibited high error but remained

somewhat stable as it had not encountered the expert's specific left wind correction patterns. It achieved a Cross-Track MAE of 24.271m and an Altitude MAE of 22.229m.

- **20 Epochs:** As training progressed, the model began to struggle significantly. The Cross-Track error tripled to 78.696m as the model became more confident in issuing "left-wind" corrections against a "right-wind" environment. Time within tolerance dropped to 11.2

- **225 Epochs:** At the final stage, performance collapsed. By over-specializing on the training data, the RNN issued extreme control commands that were fundamentally incorrect for the reversed wind direction, resulting in a Cross-Track MAE of 417.691m and an RMS of 493.060m.

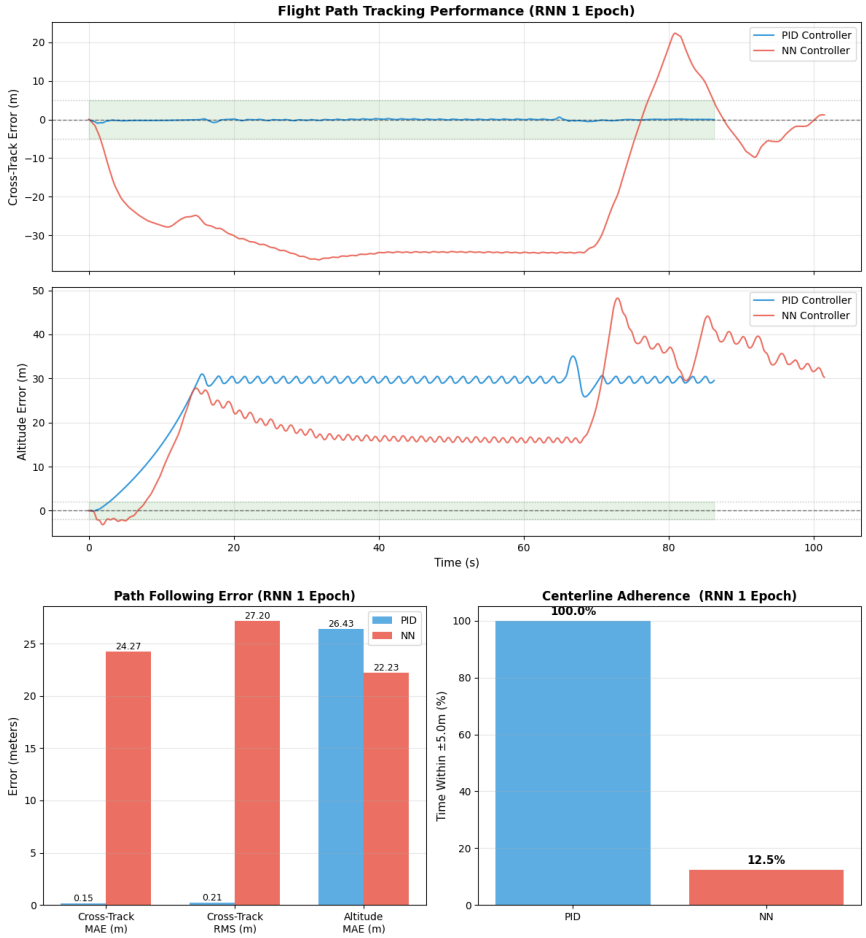

Figure 6: Wind trial results at 1 Epoch

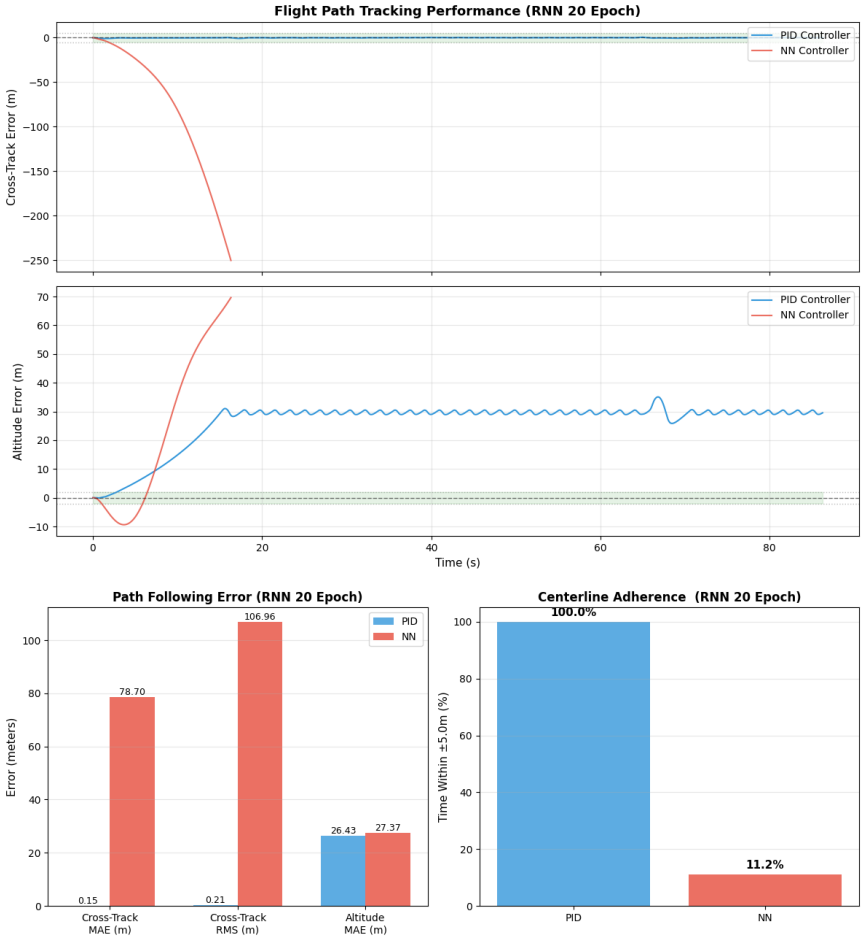

Figure 7: Wind trial results at 20 Epochs

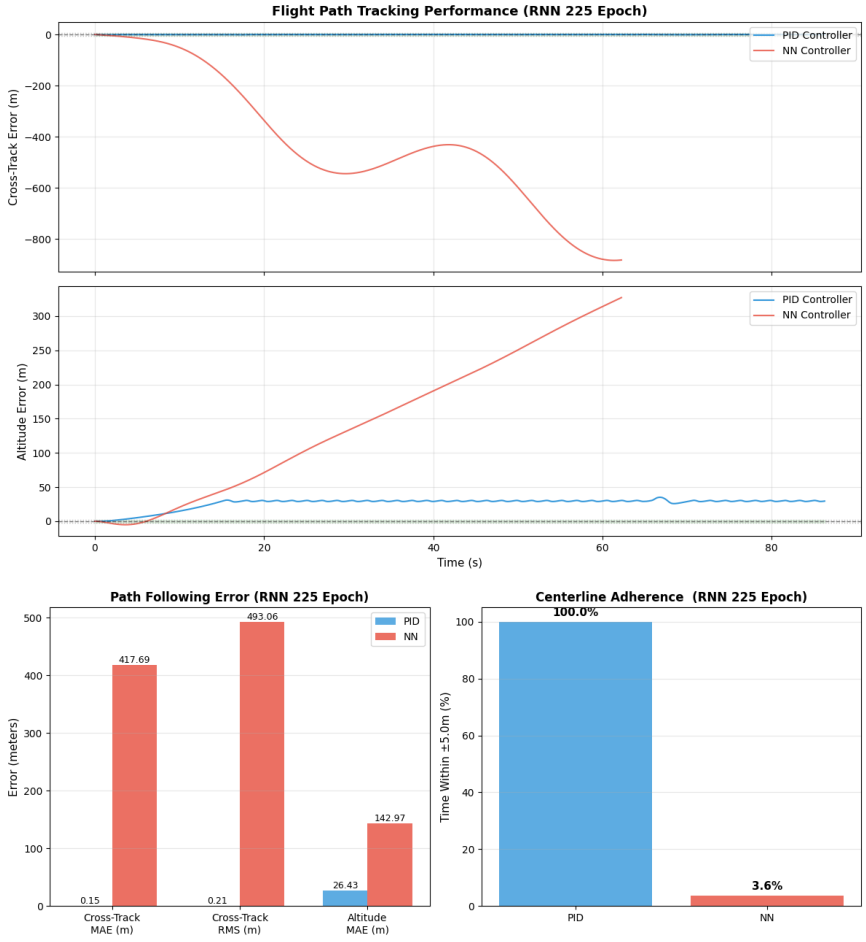

Figure 8: Wind trial results at 225 Epochs

5.3.2 Future Robustness

While the PID controller remains robust because it uses universal mathematical rules to calculate error in real-time, the RNN relied on behavioral cloning, essentially memorizing the specific correction strategy within it's training environment. As training epochs increased, the model became more stubborn, learning the teacher's specific responses so well that it was unable to adapt to novel environmental changes.

While the NN did not outperform the PID in this scenario, these series of evaluations were a success as it demonstrates that for our UAV autopilot to be truly robust, future iterations require a more diverse training set as well as improved Reinforcement Learning architectures where the model is rewarded for objective path adherence rather than simple

imitation.

6 Conclusion

This project successfully demonstrated the transition from a classical successive loop closure architecture to a unified Recurrent Neural Network autopilot for small-scale UAV control. By leveraging an LSTM-based architecture and a nonlinear sliding-surface steering law, the model achieved a high degree of fidelity in imitating its expert PID teacher. In static environmental conditions, the NN autopilot notably outperformed the PID baseline, achieving a 49.8% reduction in cross-track Mean Absolute Error and an 82.5% improvement in altitude maintenance. However, the wind perturbation trials highlighted the primary limitation of pure behavioral cloning: the tendency for high-epoch models to over-specialize on training distributions, leading to performance collapse when faced with novel environmental changes like reversed crosswinds. These results suggest that while LSTMs are highly capable of internalizing complex flight physics, achieving true robustness requires a more diverse behavioral dataset or a transition toward reinforcement learning frameworks. Ultimately, this research validates the merit of neural inference engines in autonomous flight and provides a scalable foundation for future work in adaptive, high-performance UAV control systems.

References

Git Hub Repo

Beard, Randal W., and Timothy W. McLain. (2012) Small Unmanned Aircraft : Theory and Practice, Princeton University Press. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/ucsc/detail.action?docID=832065. Kownacki, C.,

Romaniuk, S. & Derlatka, M.(2025) Applying neural networks as direct controllers in position and trajectory tracking algorithms for holonomic UAVs. Sci Rep 15, 12605. https://doi.org/10.1038/s41598-025-97215-9 D. Shukla, S. Keshmiri and N. Beckage,

(2020), "Imitation Learning for Neural Network Autopilot in Fixed-Wing Unmanned Aerial Systems," 2020 International Conference on Unmanned Aircraft Systems (ICUAS), Athens, Greece, pp. 1508-1517, doi: 10.1109/ICUAS48674.2020.9213850