Extended Abstract

Motivation Autonomous landing is a critical capability for UAVs operating in GPS-denied planetary environments like Mars, where GNSS signals and human teleoperation are unavailable. To address this, we propose a learning-based framework that enables a UAV to perceive, and land on visually designated targets using onboard sensors.

Method The approach decomposes the landing task into two learnable modules: a CNN-based perception system for visual localization and an LSTM-based control policy that outputs thrust and torque commands. Training is performed in two stages: (1) imitation learning with a model predictive control (MPC) expert, followed by (2) dataset aggregation (DAgger) to improve robustness. The modular architecture enables the control policy to be trained independently using privileged state data before integrating perception.

Implementation The drone is modeled as a 6DoF rigid body in simulation, equipped with a downward-facing RGB camera, radar altimeter, and IMU. The expert uses convex optimization to solve for optimal controls, while the learner network comprises a CNN-MLP for image-based position regression, and an LSTM-MLP for control prediction. See Figure 2 for an overview of the architecture.

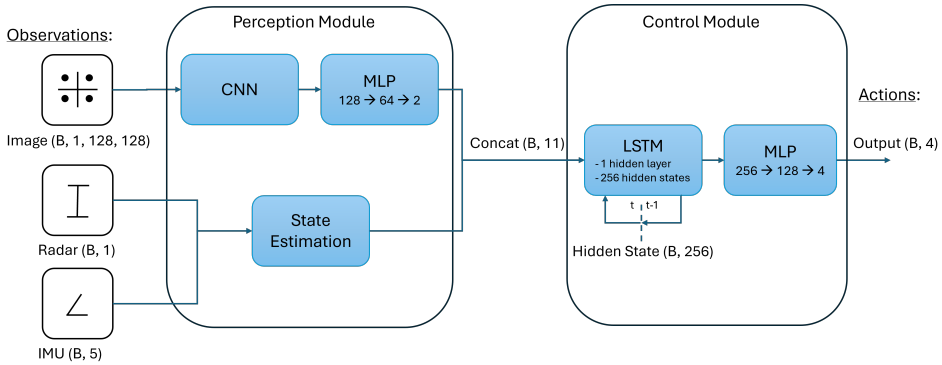

Figure 1: Modular policy architecture combining CNN-based perception and LSTM-based control.

Results Evaluation on 300 test trajectories shows that behavior cloning alone yields poor generalization (5.74m average error). DAgger reduces this to 0.70m. Integrating vision (CNN+LSTM) achieves a competitive 0.84m landing error despite perception uncertainty. Control smoothness and convergence to target positions remain consistent with expert behavior.

Discussion Compared to optimization-based policies (runtime: 0.125s), the learned policy offers a 10× speedup (0.013s), enabling real-time deployment on embedded systems. The modular design allows flexible swapping of perception models for different environments, while maintaining the same control policy interface.

Conclusion This work demonstrates a modular, learning-based vision-and-control pipeline for precision UAV landing. The combination of imitation learning, DAgger, and architectural separation yields robust, interpretable, and real-time policies suitable for GPS-denied autonomous flight.

Vision-Based Autonomous Landing through Imitation Learning

Mike Timmerman
Department of Aeronautics and Astronautics
Stanford University
mtimmerm@stanford.edu

Abstract

This work presents a modular learning-based framework for autonomous precision landing of drones in GPS-denied environments, with a focus on planetary exploration scenarios such as Mars. The proposed system combines a convolutional visual encoder with a recurrent control module to map onboard camera, radar, and inertial sensor data directly to thrust and torque commands. A model predictive controller (MPC) provides expert demonstrations, which are used to train the policy via imitation learning. To address distribution shift, we employ Dataset Aggregation (DAgger). The modular design allows independent training of perception and control components, facilitating interpretability and flexible deployment. Evaluation in a high-fidelity 6-DoF simulation demonstrates that the learned policy achieves sub-meter landing accuracy from diverse initial conditions, while running nearly 10× faster than the expert MPC baseline. The architecture offers a viable path toward vision-based autonomy for aerial robots in remote and communication-limited environments.

1 Introduction

Unmanned aerial vehicles (UAVs) offer unique advantages in mobility and terrain accessibility compared to ground-based systems, making them a promising technology for future planetary exploration. This potential was demonstrated by NASA's Ingenuity Mars Helicopter, which achieved the first powered flight on another planet. However, extending the capability of aerial systems beyond demonstration missions presents several challenges. One of the most critical limitations is the absence of Global Navigation Satellite System (GNSS) infrastructure on Mars, which precludes standard global positioning techniques. Additionally, the significant communication delay between Earth and Mars makes real-time human teleoperation infeasible. As a result, planetary UAVs must operate with a high degree of onboard autonomy, relying solely on local sensing and decision-making to perform complex tasks such as navigation and landing.

This work addresses a core component of that autonomy: vision-based precision landing in GPS-denied environments. We propose a learning-based framework for training an autonomous drone to identify and land on designated safe zones using onboard camera, radar altimeter, and inertial measurements. By leveraging imitation learning, the drone is trained to perceive visual features, estimate its position relative to a target, and execute a safe and controlled descent. The policy is implemented as a neural network architecture consisting of a convolutional encoder, recurrent memory, and thrust and torque predictor, reflecting the perception-action loop required for real-time control.

Stanford CS224R 2025 Final Report

2 Related Work

As in many other domains, deep learning has attracted growing interest for visual navigation and control in autonomy tasks involving space systems. One such task—autonomous landing in extraterrestrial environments using visual sensors—has been explored in prior work. In particular, Ghilardi et al. (2020) trained a policy using a CNN-LSTM architecture through imitation learning. Their scenario involved a lunar descent of a lander modeled as a point mass. The policy receives RGB images as observations, which are processed by a convolutional encoder. The encoder outputs from multiple timesteps are then passed as a sequence of embeddings to an LSTM module which in turn outputs acceleration commands. The policy is trained purely via supervised learning on pre-generated trajectory data.

This work's approach does not incorporate dataset aggregation (DAgger), nor does it fine-tune the policy using reinforcement learning methods—two key limitations that I aim to address in my work. Moreover, a simplified system without attitude dynamics is assumed. This simplification overlooks a critical aspect: since the visual sensor is mounted on the vehicle, the system's attitude directly affects the observed image stream. To bridge this gap, my project aims to develop a methodology that accounts for full six degrees of freedom motion, enabling more realistic and robust learning for vision-based landing policies.

The work by Chekakta et al. (2022) adopts a similar policy architecture, employing a CNN to extract spatial features from image data and an RNN based on LSTM to capture temporal dependencies. However, their approach leverages LiDAR-based imagery, which offers significantly richer spatial information through point cloud data. This data is pre-processed into grayscale elevation, slope, and range images, providing detailed representations of the environment. While this can enhance perception, it also introduces additional system constraints—LiDARs are generally more power-intensive, complex, and less commonly flight-proven compared to standard cameras. The policy ultimately outputs a full pose estimate, including both attitude and position.

Similar to the previous work, the policy is trained purely through supervised learning—i.e., imitation learning—without further refinement via policy optimization. This presents an opportunity for improvement. Additionally, the goal of their system is to recover complete pose information, which is more information-rich than necessary for the task of autonomous landing. In contrast, my objective is to directly output control commands—specifically accelerations—required to guide the vehicle to a safe landing. For this, only relative state information is needed, which I hypothesize to be easier to infer from visual data than absolute pose estimates. This distinction allows for a potentially simpler learning problem.

3 Method

3.1 Environment

The drone is modeled as a rigid body with six degrees of freedom (6DoF), capturing its full translational and rotational dynamics in three-dimensional space. The system state includes position (x, y, z), orientation (roll, pitch, yaw), linear velocity, and angular velocity, all expressed in an inertial or body-fixed frame as appropriate. The equations of motion are integrated using a 4th order Runge-Kutta scheme.

The drone is equipped with the following onboard sensors:

- **Downward-facing RGB camera**: captures 128×128 pixel images used for visual navigation and landing pad detection.
- **Radar altimeter**: provides altitude measurements relative to the surface beneath the drone.
- **Inertial Measurement Unit (IMU)**: measures attitude and angular velocity.

The actuators consist of four independent rotors generating vertical thrust and differential torques, allowing full control over roll, pitch, yaw, and collective thrust. The outputs of the policy correspond to normalized vertical thrust and torque commands, which are mapped to rotor speed inputs through a low-level controller. The dynamics also model external forces such as gravity and aerodynamic drag.

The landing target is a planar surface marked with a high-contrast red cross against a green background. This design facilitates visual detection and is visible within the camera frame at the start of each trajectory. A summery of parameters is given in Table 1

Table 1: Drone Parameters Used in Simulation

Parameter	**Symbol**	**Value**
Mass	m	1.0 kg
Arm length (motor to center)	l	0.125 m
Moment of inertia (diagonal)	I_{xx}, I_{yy}, I_{zz}	0.009, 0.009, 0.016 kg·m^2
Max rotor thrust	T_{max}	25 N
Image resolution	-	128×128 px
Camera FoV	-	45 deg
Camera frame rate	-	10 Hz
Radar range	-	0–150 m

3.2 Expert Agent

To generate expert demonstrations for imitation learning, we implement a model predictive control (MPC) framework based on linear time-invariant (LTI) dynamics. The MPC controller plans over a finite time horizon t_N, discretized with control timestep Δt, and solves a convex optimization problem to minimize a quadratic cost function involving control effort and terminal state error.

The drone is modeled as a 6-DoF rigid body with 12 states: Euler angles (ϕ, θ, ψ), angular velocities (p, q, r), position (x, y, z), and linear velocities (v_x, v_y, v_z). The control inputs consist of a collective thrust and three independent torques about the body axes. The nonlinear dynamics $f(x, u)$ are linearized around a steady-state hover using Jacobians computed via automatic differentiation, resulting in a local approximation of the form $\dot{x} = A(x - x_0) + B(u - u_0)$, where $f(x_0, u_0) = 0$.

At each control cycle, the MPC solves a convex quadratic program using CVXPY to compute the optimal control sequence. The objective includes a quadratic penalty on control deviations from hover as well as final state error. The controller adapts its cost structure depending on proximity to the landing target: if the drone is within a threshold distance, a final-state-tracking cost is enforced; otherwise, control efficiency is prioritized.

The optimization is subject to several constraints, including:

- Saturation limits on the total thrust ($\leq 2.5\, mg$) and body torques (≤ 0.01 Nm),
- Tilt angle constraints (roll and pitch $\leq 30^\circ$),
- Altitude constraint to ensure $z \geq 0$.

Importantly, the expert policy has access to privileged information during control: it observes the full 12-dimensional state of the system, including position, orientation, and their time derivatives. This level of observability is not available to the learning agent, which must instead infer position from onboard vision and radar sensors. As such, the expert serves as an idealized teacher from which the policy can learn despite partial observability.

3.3 Policy Architecture

A modular deep learning architecture is designed that separates perception and control. The system is composed of two primary components: a visual encoder and a control module. The visual encoder processes image observations to infer positional information, while the control module combines this positional encoding with additional onboard sensor data to produce control commands.

Specifically, gray-scale images are first passed through a convolution neural network (CNN) followed by a multilayer perceptron (MLP) to extract a compact positional encoding. This learned representation captures relevant visual cues in the scene, such as the location of the landing pad. This perception pipeline outputs an estimated 2D position in the image plane, which is then concatenated with auxiliary measurements from the drone's onboard sensors—namely, attitude (from an inertial measurement unit), altitude (from radar), and attitude rates (from gyroscopes).

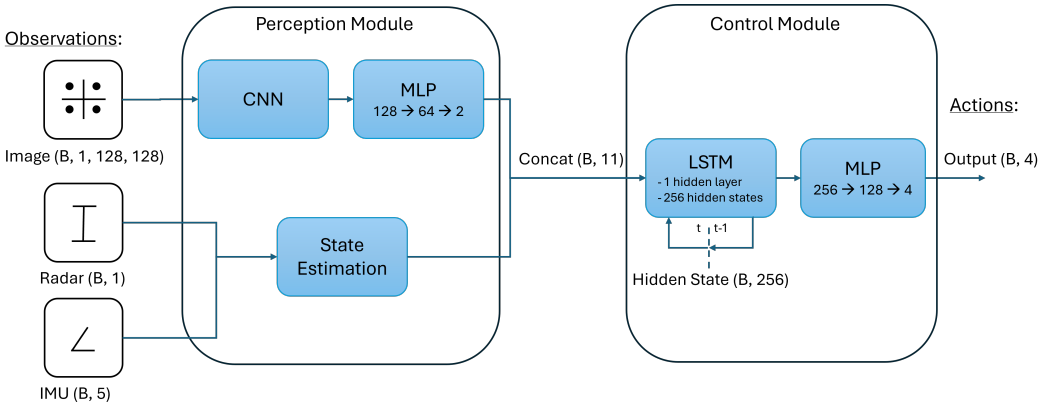

Figure 2: Policy Architecture.

The concatenated state vector is fed into a single-layer LSTM network to capture temporal dependencies in the drone's motion. The output of the LSTM is processed by another MLP that maps the internal state to the final control actions: vertical thrust and body-frame torques. This modular structure allows the perception and control components to be trained independently or jointly, providing flexibility and interpretability. The separation also supports progressive training schemes, such as pretraining the LSTM on privileged state information before substituting it with vision-based positional estimates. The complete architecture is shown in Figure 2.

3.4 Imitation Learning

The initial policy is trained using *behavior cloning* (BC), which formulates policy learning as a supervised learning problem. Given a dataset $\mathcal{D}_{\text{expert}} = \{(\tau_1), (\tau_2), \ldots, (\tau_N)\}$ of expert trajectories, where each trajectory $\tau_i = \{(o_t, a_t)\}_{t=1}^{T}$ consists of observations o_t and corresponding expert actions a_t, the goal is to learn a policy $\pi_\theta(a|o)$ that mimics the expert. This is achieved by minimizing the the mean squared error:

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \|\pi_\theta(o_t^{(i)}) - a_t^{(i)}\|^2, \tag{1}$$

where π_θ is the policy parameterized by θ, and T is the length of each fixed-length segment sampled from the replay buffer. These segments are used to ensure consistent temporal context, especially when training recurrent architectures such as LSTMs.

After the initial behavior cloning stage, the policy is further refined using the *Dataset Aggregation* (DAgger) algorithm. At each DAgger iteration k, the current policy $\pi_\theta^{(k)}$ is deployed to collect new rollouts $\tau^{(k)}$, which are then relabeled by querying the expert for the optimal action $\pi^*(o_t)$ at each timestep. These relabeled transitions are aggregated into the dataset:

$$\mathcal{D}_{\text{agg}} \leftarrow \mathcal{D}_{\text{agg}} \cup \{(o_t, \pi^*(o_t))\}_{t=1}^{T_k}, \tag{2}$$

and the policy is retrained by minimizing the same MSE loss over the expanded dataset $\mathcal{D}_{\text{agg}}$. This iterative process helps the policy recover from distribution shift by training on the states it is likely to encounter under its own control.

4 Experimental Setup

4.1 Expert Data Collection

To generate training data for imitation learning, we collected a total of 300 expert trajectories using the MPC-based controller described in Section 3. Each trajectory spans 10 seconds and contains approximately 100 timesteps, captured at a frequency of 10 Hz. To ensure diversity and broad coverage of the state space, we sampled initial conditions using a hybrid strategy combining low-discrepancy Sobol sequences with uniform sampling. The initial state distribution includes randomized roll and pitch angles within $\pm 1.5^\circ$, horizontal positions spanning ± 5 meters from the landing pad, altitudes ranging between 75 and 100 meters, and horizontal and vertical velocities sampled from ranges of $[-1, 1]$ m/s and $[-0.5, 0]$ m/s respectively. Angular rates were initialized to zero to focus the learning task on position and attitude control.

Each trajectory includes the full 12-dimensional system state, partial observations and expert actions computed by the MPC. These rollouts form the initial expert dataset $\mathcal{D}_{\text{expert}}$ used for behavior cloning. This systematic data generation process ensures that the learning agent is exposed to a representative distribution of landing scenarios.

4.2 Replay Buffer and Data Sampling

To support training across multiple DAgger iterations, we employ a replay buffer that stores full expert and policy-generated rollouts. The buffer maintains episode-level trajectories and allows efficient sampling of fixed-length segments tailored for recurrent policy training. To limit memory usage, the buffer holds a maximum of 400 episodes, removing the oldest rollouts in FIFO order when this limit is exceeded.

During training, a batch is constructed by first determining the number of segments as the batch size divided by the fixed segment length (typically 25 timesteps). Eligible rollouts—those with sufficient length—are selected at random, and for each, a contiguous segment of 25 timesteps is extracted starting from a randomly sampled offset. This segment-wise sampling ensures that each batch contains temporally coherent sequences necessary for LSTM-based policies while also diversifying coverage over different phases of the trajectories. This structured sampling mechanism enables stable learning from sequential data and supports effective reuse of demonstrations and learner experience.

4.3 Training Strategy

The policy is trained using a two-step process across perception and control. This approach isolates the submodules to simplify learning and ensure stability.

Stage 1: LSTM Control Module Pretraining. The control module, consisting of an LSTM and output MLP, is first trained independently using ground-truth state inputs. In this stage, the drone's position and velocity are directly accessible, allowing the LSTM to learn a control policy without perception noise. The training follows a behavior cloning (BC) phase, followed by two Dataset Aggregation (DAgger) iterations. Each iteration is trained over 1500 epochs using batches of 20 trajectory segments with fixed length of 25 timesteps, resulting in a total of 2,250,000 training datapoints (90,000 trajectory segments). In each DAgger round, an additional 10,000 timesteps (approximately 100 learner rollouts) are collected and relabeled using the expert.

Stage 2: CNN Perception Module Pretraining. In parallel, the CNN perception module is pretrained to regress 2D position estimates from image observations. The training uses expert rollouts split 90/10 into training and evaluation sets. Over 1000 epochs with a batch size of 64, the network is trained on 64,000 total examples. The goal is to enable the CNN to encode visual cues—such as the landing pad appearance—into accurate spatial representations.

5 Results

5.1 Expert Trajectories

Figure 3 and Figure 4 illustrate the expert trajectories in terms of linear position, velocity, and corresponding control inputs. These figures highlight the diversity of initial conditions sampled during data collection, resulting in a wide distribution of trajectories. This variation ensures that the expert dataset provides a rich and informative training signal, covering a broad portion of the state space relevant for robust policy learning.

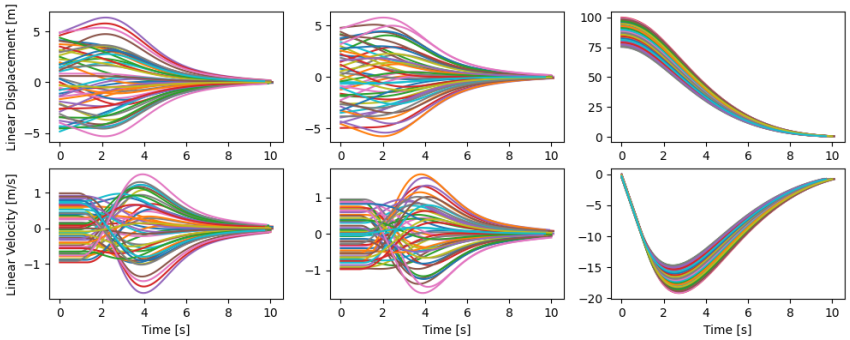

Figure 3: State trajectories covered by the expert training dataset.

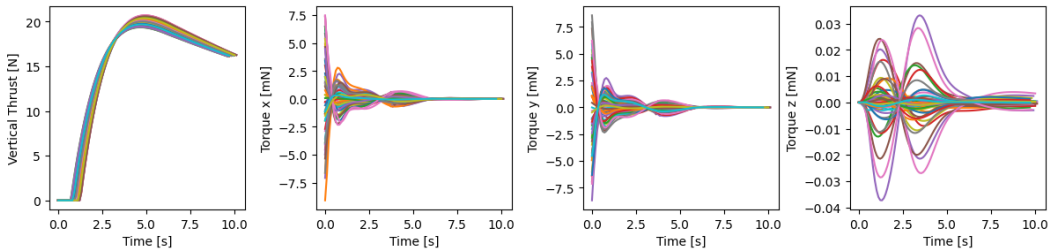

Figure 4: Input trajectories covered by the expert training dataset.

5.2 CNN Perception Module

The loss over the training epochs for both training and validation sets is shown in Table 2. The CNN was trained to predict 2D image-plane positions from grayscale camera inputs using a mean squared error loss. Rapid improvements are observed in the early training phase, with both training and validation losses dropping significantly within the first 200 epochs. By iteration 400, performance appears to plateau, and after 600 epochs, the loss stabilizes around 0.034 on both datasets.

This stagnation suggests that the CNN converges to a local optimum and further training does not yield meaningful improvements. The similar loss values between the training and validation sets indicate that the model is not overfitting, but rather reaches the capacity of its representational power under the current architecture and data regime. The achieved loss corresponds to an average squared error of approximately 0.034. This implies a root mean squared error (RMSE) of about ≈ 0.18 meters in the xy-plane.

This plateau in performance motivated additional training of the CNN in conjunction with the LSTM module in the second stage of training, to better align the CNN's output distribution with the downstream control network's expectations.

Table 2: Performance Comparison

Iter	Training Set		Validation Set	
	Avg. Loss	Std. Loss	Avg. Loss	Std. Loss
Iter 0	0.1241	0.0581	0.1276	0.0542
Iter 200	0.0351	0.0247	0.0354	0.0258
Iter 400	0.0342	0.0238	0.0340	0.0252
Iter 600	0.0342	0.0238	0.0339	0.0252
Iter 800	0.0343	0.0239	0.0343	0.0252
Iter 1000	0.0343	0.0238	0.0342	0.0252

5.3 LSTM Control Module

Figure 5 shows the loss over training epochs. Annotated in the figure are the points at which the dataset is aggregated with expert labeled learner rollouts. In the inital behavior cloning, a strong decline in loss is observed. Once DAgger is initiated, the loss shoots up rapidly again, while slowely converging to an optimal point. Notice that the same final loss is not achieved as in the initial behavior cloning, however, a stronger performance is observed from the evaluation trajectories. Evaluation trajectories after behaviour cloning and after the final DAgger iteration are given in subsection A.1 from which the importance of DAgger becomes apparent.

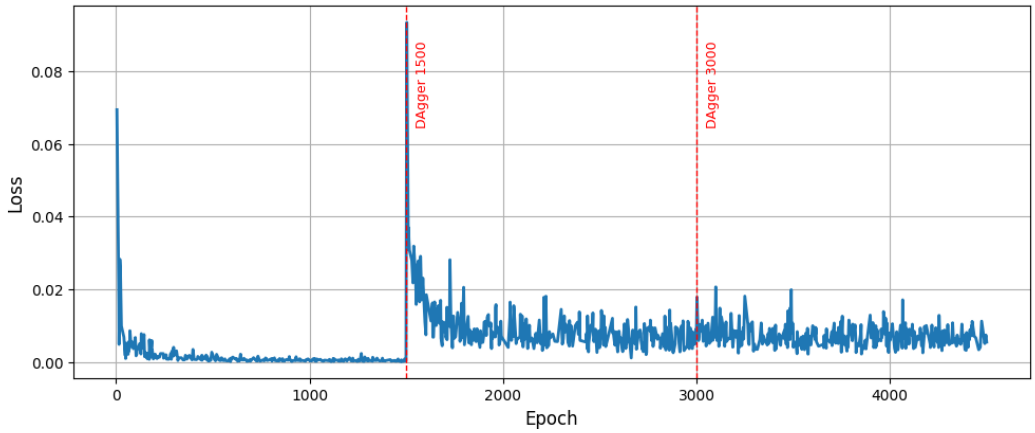

Figure 5: LSTM Control Module Loss over DAgger Iterations.

5.4 Quantitative Evaluation

Table 3 reports the final landing position errors (in meters) for three policy variations. The LSTM behavior cloning and dataset aggregation policies rely exclusively on perfect state information, evaluating the performance of the control module in isolation. As shown, the behavior cloning baseline yields the highest average error ($5.74\,\mathrm{m}$) and largest variance, suggesting poor generalization from expert demonstrations alone. Incorporating Dataset Aggregation (DAgger) significatly improves control accuracy, reducing average error to $0.70\,\mathrm{m}$ and significantly lowering the variance, confirming the value of iterative data correction. The LSTM + CNN policy additionally integrates a learned perception module that estimates position from visual input. Despite the added uncertainty from visual estimation, this model achieves comparable accuracy ($0.84\,\mathrm{m}$ average error), demonstrating that the perception module effectively supports the control pipeline.

Table 3: Landing Performance Metrics.

Policy	Min. Error	Max. Error	Avg. Error	Std. Error
LSTM behavior cloning	1.56	10.47	5.74	2.14
LSTM dataset aggregation	0.56	0.91	0.70	0.09
LSTM + CNN	0.54	1.27	0.84	0.21

5.5 Qualitative Evaluation

Figure 6 and Figure 7 show the position, linear velocity, and control inputs for the integrated perception and control policy. The trajectories closely follow the expert demonstrations, with the agent consistently converging toward the target landing position. Notably, the control inputs are smoother and lack the oscillations observed in the LSTM-only policy (Figure 11). However, the state trajectories exhibit a wider spread in final landing positions and a higher terminal velocity of approximately $5\,\mathrm{m/s}$ compared to the LSTM-only case. This suggests that while the perception module enables effective control, further fine-tuning—such as policy gradient optimization with an explicit penalty on terminal velocity could improve robustness and landing precision.

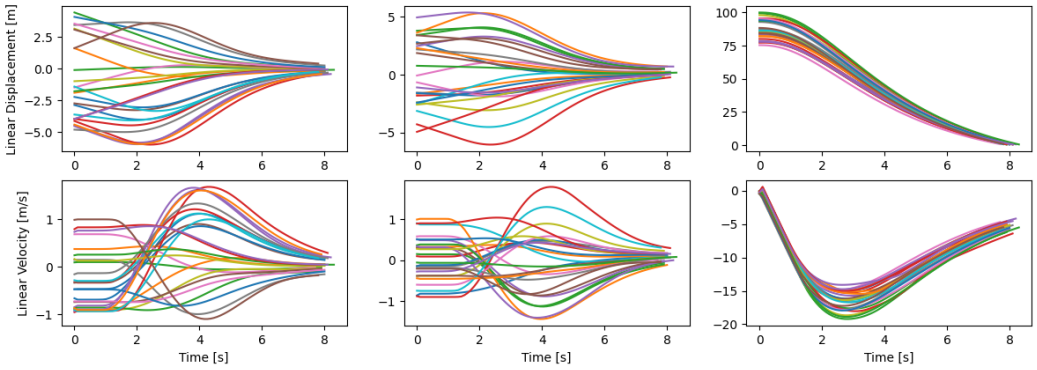

Figure 6: State trajectories for integrated perception and control modules.

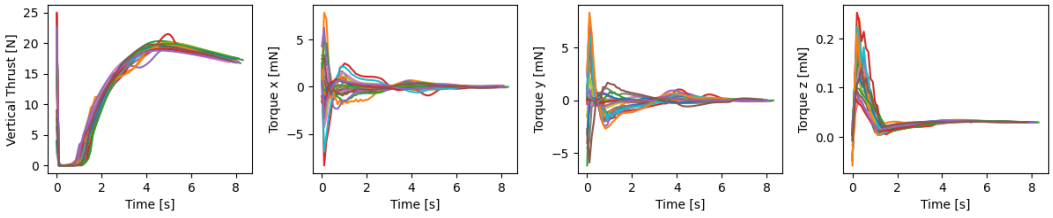

Figure 7: Input trajectories for integrated perception and control modules.

6 Discussion

The experimental results demonstrate the effectiveness of the proposed modular architecture in achieving accurate and robust autonomous landings. Beyond accuracy metrics, several practical advantages arise from the design choices made in this system.

Runtime Efficiency. A key advantages of the LSTM-based control module is its real-time inference capability. In contrast to the expert policy—which solves a convex optimization problem at each timestep and requires 0.125 ± 0.025s per control input—the end-to-end CNN+LSTM policy generates control commands in just 0.0132 ± 0.0014s. This represents an almost $10\times$ reduction in runtime, a significant improvement that is critical for deployment on embedded platforms with constrained computational resources or in high-frequency control loops where low-latency is essential.

Modularity and Adaptability. The separation between perception and control offers significant benefits in terms of modularity and adaptability. Each module can be independently improved, swapped, or adapted to new environments without requiring full retraining of the entire system. For example, in highly structured environments, a simple CNN-based perception module may suffice to estimate position. In contrast, in more unstructured or dynamic environments, the perception module could be replaced with a more complex vision system capable of semantic segmentation or visual landmark detection. As long as the output remains a relative target position, the control module can remain unchanged, effectively generalizing to new tasks with minimal retraining.

Stable Curriculum and Training Pipeline. The staged training strategy—first training perception and control independently, followed by potential alignment and fine-tuning—enables stable and interpretable learning. This progressive build-up reduces the risk of catastrophic failure in end-to-end learning pipelines and provides clear debugging and validation checkpoints. Moreover, such a structure aligns well with progressive autonomy approaches, where modules can be verified individually.

Overall, this work highlights the feasibility of combining modular deep learning components for perception and control in safety-critical robotics tasks. The architecture not only performs well empirically but also provides a principled foundation for extensibility and deployment in real-world scenarios.

7 Conclusion

This work presents a modular deep learning framework for vision-based precision landing in autonomous aerial systems operating without GPS. By combining expert MPC-generated demonstrations with staged imitation learning, our approach achieves robust and accurate landings using only onboard camera, radar, and inertial measurements. The control policy, built around an LSTM network, demonstrates strong generalization when trained with Dataset Aggregation, and retains high performance when paired with a CNN perception module trained to estimate image-plane positions. Despite the partial observability and high-dimensional image inputs, the integrated CNN+LSTM policy achieves near-expert performance while reducing runtime by an order of magnitude—an essential benefit for real-time applications.

Importantly, the system's modular design allows for independent development and replacement of perception and control components, enabling adaptability to different sensor suites and environmental conditions. This decoupling also facilitates structured training and debugging, which are critical for scaling autonomy in complex domains. Future work may explore end-to-end fine-tuning using reinforcement learning, incorporate richer visual reasoning (e.g., terrain classification or visual SLAM), and extend the system to more dynamic or unstructured planetary environments. The results demonstrate a promising step toward enabling autonomous aerial navigation and landing in GNSS-denied, communication-constrained contexts such as planetary exploration.

Changes from Proposal While it was intended to use policy gradient optimization to fine-tune the policy to achieve higher landing accuracy and lower terminal velocity, due to the changes of the policy architecture into a modular design, I did not manage to finish the further fine-tuning implementation. However, I do intend to further work on this after the class ends for own interest.

References

Zakaria Chekakta, Abdelhafid Zenati, Nabil Aouf, and Olivier Dubois-Matra. 2022. Robust deep learning LiDAR-based pose estimation for autonomous space landers. *Acta Astronautica* 201 (2022), 59–74. `https://doi.org/10.1016/j.actaastro.2022.08.049`

Luca Ghilardi, Andrea D'Ambrosio, Andrea Scorsoglio, Roberto Furfaro, Richard Linares, and Fabio Curti. 2020. Image-based Optimal Powered Descent Guidance via Deep Recurrent Imitation Learning.

A Additional Experiments

A.1 LSTM Module Training

Figure 8 and Figure 9 show the position and linear velocity, and corresponding control inputs after initial behavior cloning. The rotational states are omitted for brevity. Clearly, the learned policy does not achieve its task of landing near the desired position, and the inputs are very chaotic and diverse across trajectories, as opposed to the training data.

Additionally, Figure 8 and Figure 9 show the position and linear velocity, and corresponding control inputs after the dataset aggregation phase. Clearly, these trajectories align better with the expert trajectories, with the agent converging towards the target. One aspect to note is the apparent oscillation in the control torque trajectories.

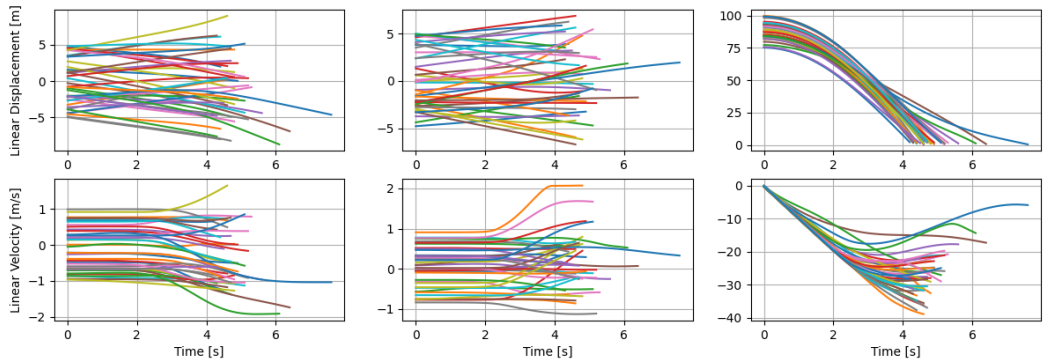

Figure 8: Learner state trajectories after initial behavior cloning.

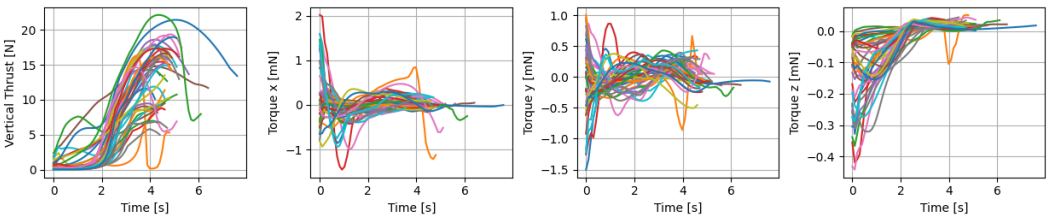

Figure 9: Learner input trajectories after initial behavior cloning.

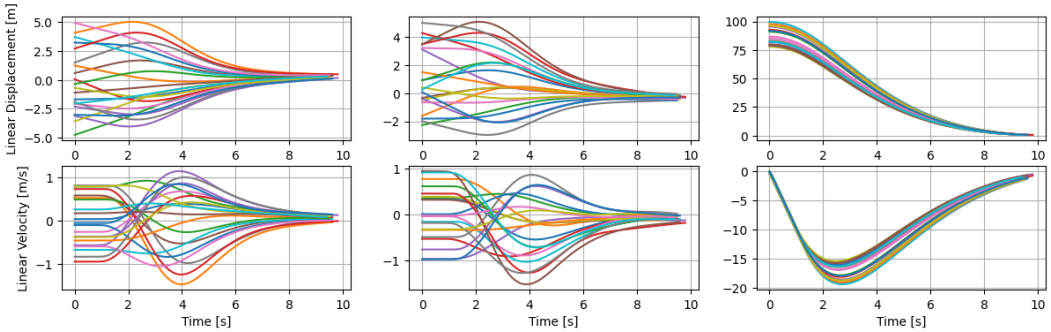

Figure 10: Learner state trajectories after dataset aggregation.

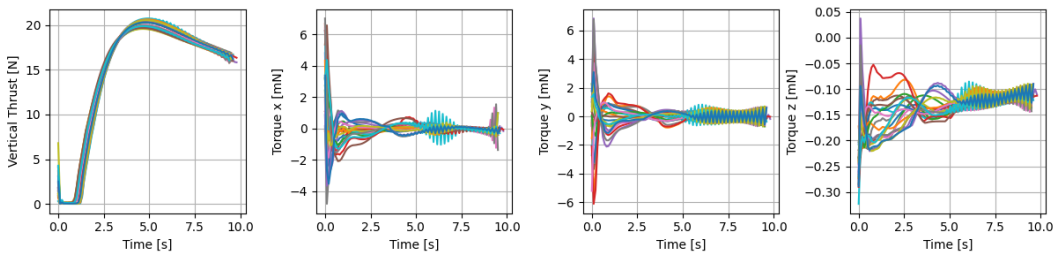

Figure 11: Learner input trajectories after dataset aggregation.

A.2 Implementation Details

Environment. The autonomous landing task is implemented using a custom `gymnasium` environment. The environment exposes a continuous action space $\mathcal{A} \subset \mathbb{R}^4$, where the agent controls normalized thrust and moment commands. The state transition dynamics follow a 12-dimensional quadrotor model, comprising attitude (Euler angles), angular velocity, position, and linear velocity.

Dynamics integration is performed using a fourth-order Runge–Kutta (RK4) solver with a fixed timestep of $\Delta t = 0.1\,\mathrm{s}$. A landing pad is rendered in the scene, and the simulation terminates either upon reaching a target landing altitude or after a fixed time horizon.

Rendering and Observations. To generate image observations, the drone is rendered in a 3D PyBullet scene. The drone is visualized as a simple cuboid with dimensions and inertia properties configured via a structured configuration object. For each simulation step, the drone's visual pose and velocity are updated to match the integrated physics state.

A virtual downward-facing camera is mounted on the drone. Its extrinsics are computed using the drone's current world-frame position and orientation, with the look-at direction aligned along the negative body Z-axis.

The camera produces RGB images at 128×128 resolution using a pinhole camera model with a field of view of $45°$. Only grayscale-converted RGB frames are used by the agent for policy inference.

Episode Execution. Each episode begins with a randomized initial state (attitude, velocity, and position) unless specified. During each timestep, the drone receives an action, integrates its state forward, updates the PyBullet rendering, and receives the next observation. The environment provides logging support for actions, states, and timestamps to facilitate offline analysis of trajectories.

Project Code. `https://github.com/MikeTimmerman-ae/Visual_Drone_Landing/`