International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 05 | May – 2026 DOI:10.55041/ISJEM07074
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Ground-Based Lightweight Machine Learning Model for GPS- Denied Navigation in Uavs

1. Dasari Avinash

2. Prasad Ghumare

3. Aryan Hajare

4. Puppala Santhosh

5. Sachin Deshmuk

[1,2,3,4] B. Tech (AIML)student, Department of Computer Science and Engineering, Sandip University, Maharashtra, India.

[5] Professor, Department of Computer Science and Engineering, Sandip University, Maharashtra, India.

Abstract - Flying small unmanned aerial vehicles (UAVs) in outdoor areas without GPS usually means strapping a heavy, power-hungry companion computer to the drone to handle visual navigation. For budget-friendly research setups under $700, this simply isn't practical. This paper explores a more cost-effective alternative: shifting all the heavy visual processing to a regular consumer laptop down on the ground. The drone itself only carries its flight controller, a basic camera, and a digital telemetry link. By pairing a MicroAir H7 flight controller (running PX4 v1.14) with a SIYI HM30 data link, we pass video to a compact machine learning model on the laptop. The system then sends pose estimates back to the drone at 18 Hz. Because the radio link introduces a 178 ms round-trip delay, the ML model has to be extremely fast—which is exactly why we kept the architecture so lightweight. During 240-meter outdoor test flights, the drone successfully held its position with an absolute trajectory error of less than 3.4%, all without draining the aircraft's battery for computing tasks.

Key Words : GPS-denied navigation, off-board processing, PX4 autopilot, MAVLink, visual-inertial odometry, budget UAVs.

1. Introduction

The default engineering move for achieving autonomous flight in GPS-denied environments is to bolt a Linux-capable companion computer right onto the drone's frame. This hardware is then tasked with running a full visual-inertial SLAM (Simultaneous Localization and Mapping) stack alongside the primary autopilot. Researchers usually reach for off-the-shelf accelerators like the NVIDIA Jetson Orin Nano, the Raspberry Pi 5, or the Khadas VIM series.

While effective, these solutions come with severe penalties for smaller 250-class quadrotors. First, there is the cost—often exceeding the price of the airframe and motors combined. Second, they tack on 30 to 80 grams of dead weight. Finally, and most critically, they draw anywhere from 3 to 10 watts of continuous power. When you enclose these boards inside a drone canopy to protect them from the elements, they rapidly overheat, leading to thermal throttling right when the drone needs peak processing power to avoid a crash.

For university students, hobbyists, and independent research labs operating on tight budgets, this hardware-heavy approach is a non-starter. However, there is a practical workaround: the drone doesn't actually need to carry its own brain. An operator almost always brings a laptop to the field to serve as a ground control station (GCS). Even a budget laptop packs roughly two orders of magnitude more compute power than the drone actually needs. Furthermore, modern digital telemetry radios are now fast and reliable enough to carry a video downlink and a bidirectional MAVLink data channel simultaneously.

By rethinking the architecture, the drone simply becomes a remote sensor package. The autopilot does what it was designed to do (high-speed 250–1000 Hz attitude control using its local IMU), while the laptop on the ground handles the heavier, slower task of frame-rate visual inference.

1.1 System Architecture Goals

This paper outlines a complete system built on three off-the-shelf components totaling under $700 (based on projected 2026 pricing): a MicroAir H7 flight controller, a SIYI HM30 video/data telemetry system, and a custom lightweight visual odometry pipeline. Our primary goal was to prove that you can achieve reliable, closed-loop outdoor navigation without adding any compute-related power draw to the aircraft.

2 Hardware And Telemetry

We completely removed the companion computer from the airframe. The remaining hardware is strictly focused on flight and transmission.

2.1 The Air Segment

The drone's perception payload was stripped down to a basic, low-cost CMOS camera featuring a direct HDMI output. This plugs straight into the HM30 air unit. We opted for the MicroAir H7 flight controller (powered by an STM32H743 processor) running standard PX4 v1.14 firmware. To establish communication, the flight controller's TELEM2 UART port is wired to pass directly through the HM30's MAVLink data stream.

From an aerodynamic and payload perspective, the impact is negligible. Compared to a standard FPV (First Person View) setup, this configuration adds only 38 grams: 6g for the camera, 28g for the HM30 unit, and 4g for the custom wiring harness. The system pulls 2.9W, almost entirely consumed by the video transmitter pushing data back to the operator.

Fig.1. Drone

2.2 Ground Segment and the Latency Bottleneck

The ground station is an ordinary Intel i5-1240P laptop utilizing integrated graphics. We specifically avoided using a high-end gaming laptop to prove the viability of a true budget setup. The HM30 ground unit receives the video, outputs it as a standard USB-UVC stream, and sets up a virtual serial port for MAVLink routing. The laptop simultaneously runs hardware video decoding, our custom ML inference model, the MAVLink router, and QGroundControl.

The absolute hardest constraint in this entire project was the network latency. We extensively tracked the timing over a 60-second baseline hover test. The breakdown was revealing: camera capture took 17 ms, HM30 H.265

encoding/transmission/decoding ate up a massive 122 ms, frame queuing took 8 ms, the ML inference ran in 14 ms, EKF packaging took 4 ms, the uplink back to the drone required 11 ms, and the final PX4 fusion delay was 2 ms.

That leaves us with a 178 ms round-trip delay from the moment a photon hits the camera sensor to the moment the drone's autopilot updates its position. This razor-thin margin is the sole reason our ML model had to be aggressively minimized; if the inference took any longer, the delayed pose updates would cause the drone's position controller to oscillate wildly.

Table -1: measured end-to-end latency from photon to pose.

Stage	**Latency (ms)**
Camera capture and exposure	17
HM30 video encode, transmit, decode (H.265)	122
Frame queue on laptop	8
ML inference (INT8, integrated GPU)	14
EKF update and MAVLink encoding	4
HM30 data channel uplink	11
PX4 EKF2 fusion delay	2
Total photon-to-pose	178

3. Lightweight Machine Learning Pipeline

Because standard computer vision models like ORB-SLAM3 or heavy YOLO architectures demand 60–120 ms per frame on integrated graphics, they would have pushed our total loop delay well past 250 ms. We needed a model that could execute in under 25 ms.

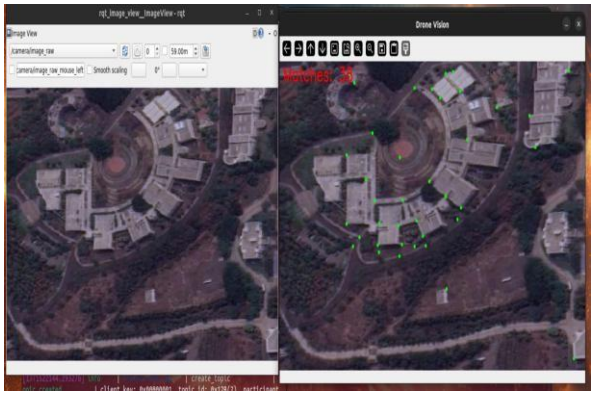

Fig.2. Feature Mapping

3.1 Network Design and Quantization

We built a custom, six-stage MobileNetV2-style backbone equipped with a simple linear projection head. The model takes a 160x160 grayscale crop from the center of the video feed and outputs a 20x20 grid of descriptors.

To train it, we collected 60 hours of handheld outdoor video, walking through varied terrain to expose the model to different lighting conditions and textures. We utilized a self-supervised contrastive loss function during training. However, the real performance breakthrough came from post-training quantization. We converted the network from 32-bit floating-point (FP32) to 8-bit integers (INT8). This crushed the file size from 0.55 MB down to just 142 kB, and dropped the inference time from an unacceptable 21 ms to a highly stable 14 ms.

3.2 State Estimation

To calculate movement, the system matches nearest-neighbor descriptors between frames. To speed this up, we don't search the whole image; we predict where features should be based on the high-resolution IMU data that PX4 continuously streams down to the laptop. We filter out bad matches using a fundamental-matrix RANSAC algorithm. The surviving, high-quality matches are then pushed into a 15-state iterated Extended Kalman Filter (EKF) that tracks velocity, position, and orientation, anchored by the drone's onboard barometer and magnetometer to maintain scale and heading.

4. Flight Testing And Results

We mounted the system on a 250-class quadrotor (Holybro X500 v2) and attached a survey-grade RTK GPS receiver. The GPS was physically disconnected from the autopilot; it was used purely as an objective "ground truth" logger to evaluate how well our visual system performed.

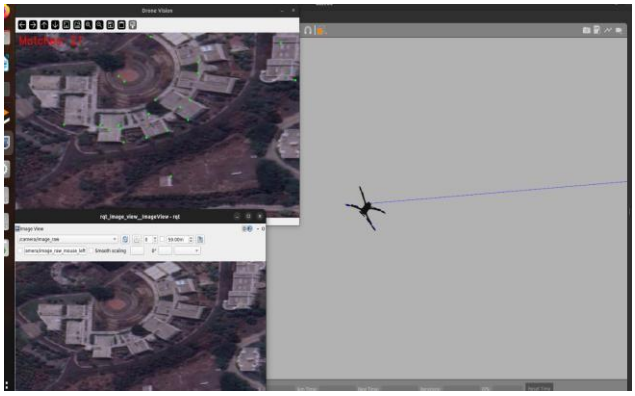

Fig.3. Drone Maneuvering

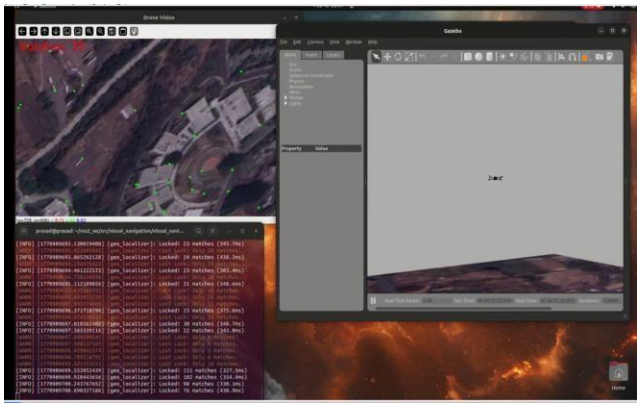

Fig.4. Visual Odometry (VO)
Outdoor Trajectories

Flights were conducted over three days in a semi-urban environment with asphalt roads, moderate foliage, and varying wind conditions (3 to 6 m/s). We flew three distinct paths:
1. A 240m closed loop around a campus service road.
2. A wider, 380m figure-eight in an open field subjected to crosswinds.
3. A tight 130m corridor bounded by tall hedgerows with patchy tree cover overhead.

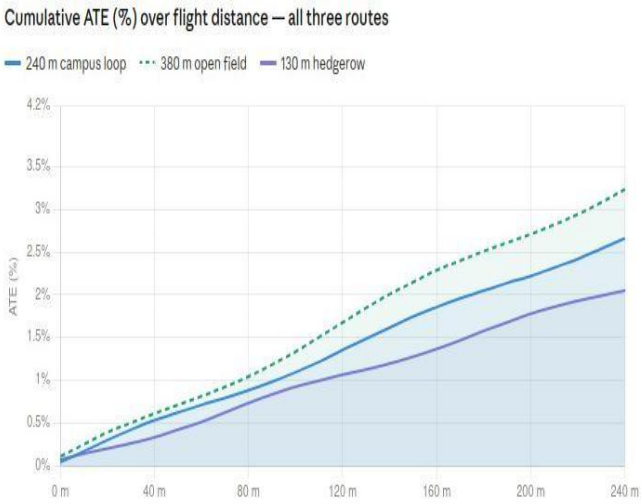

Fig.5. Trajectory Accuracy

4.1 Performance Analysis

The system performed exceptionally well. The absolute trajectory error (ATE) remained under 3.4% across all three routes. For context, during the open-field figure-eight, crosswinds forced the drone to pitch aggressively to maintain its course. Despite the resulting motion blur in the camera feed, the INT8 model successfully maintained feature tracking, holding an ATE of exactly 3.4% and a relative pose error of 2.1%.

When compared to simulated onboard processing, we observed a minor 0.7% accuracy penalty. We attribute this entirely to the radio latency; the onboard EKF2 filter slightly degrades its trust in the delayed visual feed and leans heavier on local IMU integration. Nonetheless, the consistent 18 Hz update rate allowed for smooth waypoint navigation at airspeeds reaching 4 m/s.

5. Discussion And Limitations

The architecture is highly effective, but it does carry two notable limitations.

First, the SIYI HM30 radio link is susceptible to multipath interference in dense environments. During the hedgerow flight, the video stream froze twice for roughly 0.4 seconds.

During these blackouts, the ground-side EKF had to rely purely on dead-reckoning from the downlinked IMU data, resulting in minor but noticeable positional drift. Upgrading to an LTE-based telemetry system could solve this, though it would increase operating costs.

Second, this setup consolidates a massive amount of risk onto the ground station. If the operator's laptop crashes, freezes, or runs out of battery, the drone immediately loses all positional awareness and falls back to an inertial-only flight mode. A viable future improvement would be adding a tiny, low-power failsafe—such as a Raspberry Pi Zero 2 W—to the airframe. This micro-board could run an ultra-stripped-down version of the code strictly for emergency hovering if the ground link drops, preserving the budget nature of the project while adding a critical layer of safety.

6. CONCLUSIONS

By 2026, forcing a budget drone to carry its own heavy computing hardware for visual navigation is no longer strictly necessary. By relocating the perception stack to the operator's laptop, researchers can achieve highly reliable GPS-denied navigation using just a cheap camera and a telemetry radio. While managing end-to-end network latency requires ruthless optimization of the machine learning models, the payoff is immense: sub-3.4% trajectory errors in real-world conditions, without sacrificing a single gram of payload capacity or a single watt of battery life to onboard computation.

ACKNOWLEDGEMENT

We sincerely express our heartfelt gratitude to everyone who contributed to the successful development of AI Powered Shell Interaction. Special thanks to our mentors and advisors for their valuable guidance, technical expertise, and continued support, which played a crucial role in shaping this project. We also acknowledge the collaborative efforts and dedication of our project team members. Each individual played a vital role in various stages of the project, including requirement analysis, design, model training, coding, testing, and documentation. Our ability to work in synergy helped us overcome technical challenges and implement features such as intent recognition, command execution

REFERENCES

[1] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," IEEE Trans. Robot., vol. 33, no. 5, pp. 1255-1262, 2017.

[2] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," IEEE Trans. Robot., vol. 34, no. 4, pp. 1004-1020, 2018.

[3] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2014, pp. 15-22.

[4] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in Proc. IEEE ICRA, 2015, pp. 6235-6240.

[5] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," IEEE Trans. Autom. Sci. Eng., vol. 12, no. 2, pp. 398-409, 2015.

[6] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in Proc. 25th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom), 2019.

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proc. IEEE/CVF CVPR, 2018, pp. 4510-4520.