אוניברסיטת חיפה
UNIVERSITY OF HAIFA
جامعة حيفا

Project in Real-Time Systems (203.4724)

RBD Lab, University of Haifa

Supervisor: Prof. Dan Feldman

GesturePilot: Real-Time Hand-Gesture Drone Control with ORB-SLAM3 Safety Mapping

Authors: Zahi Haddad • Rany Mattar

IDs: 216721795 • 216616011

August 31, 2025

Abstract — We present **GesturePilot**, a real-time system that maneuvers a DJI Tello drone in 3D using hand gestures detected by a laptop/PC webcam via MediaPipe. To improve spatial awareness, we pipe the drone's UDP video feed into OBS/VirtualCam so that an ORB-SLAM3 (Windows fork) process can treat it as a camera input and build a rough map. We use the SLAM output primarily for *security awareness* (e.g., rotation verification and potential collision-prevention heuristics), while gesture commands are sent to the drone via the Tello SDK. We document the system design, Windows-specific build/run steps, gesture vocabulary and debouncing, and real-world constraints (battery, connectivity, Misclassifications). We also discuss our pivot from initial drowning-detection and selfie-drone ideas, the "single viewer" constraint on Tello streams, and engineering workarounds.

Contents

1 Introduction and Project Overview

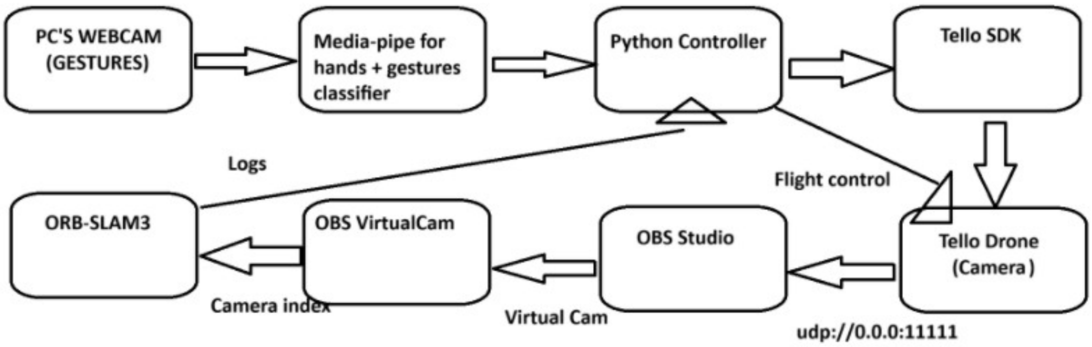
1.1 Motivation and Idea

Our original idea was a **drowning detection** concept; however, we realized it did not leverage ORB-SLAM's outputs meaningfully within the course scope. We then pivoted to a **selfie drone**, and finally converged on **GesturePilot**: a *gesture-controlled* Tello where a webcam plus MediaPipe detects hand gestures on the PC, while a separate ORB-SLAM3 pipeline consumes the drone's `udp://0.0.0.0:11111` feed to maintain basic environmental awareness. This balances *responsiveness* (gestures → direct SDK commands) with *situational context* (SLAM-based rotation/mapping awareness).

1.2 Contributions

- A practical **gesture vocabulary** mapped to Tello SDK commands (takeoff, land, forward/back, yaw, altitude).
- A Windows-friendly **ORB-SLAM3 workflow** using a community fork so we can treat the Tello feed as a camera.
- A **360° rotation verification** workflow (**30°** steps) that tails SLAM logs to confirm rotation coverage.
- Engineering around **single viewer** constraints on Tello streams using OBS/VirtualCam as a bridge.

1.3 High-Level Architecture



2 Related Tools and Libraries

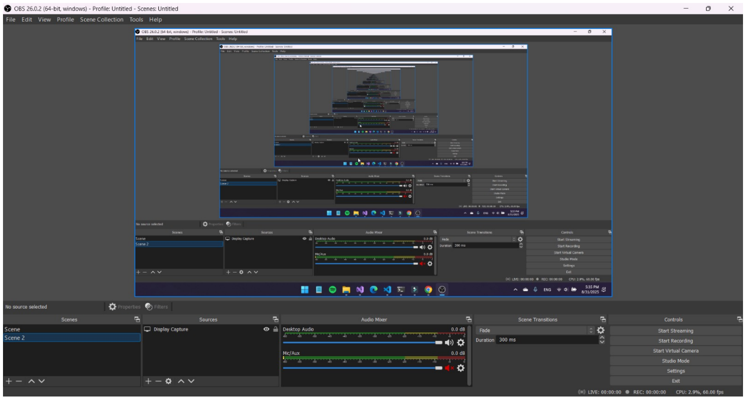
2.1 Core Components

- **DJI Tello SDK** (UDP commands, video on 11111).
- **MediaPipe Hands**: robust hand landmark detection for gesture classification.
- **OBS + VirtualCam**: captures the drone's UDP feed and exposes it as a virtual webcam.
- **ORB-SLAM3 (Windows fork)**: *ORB_SLAM3_Windows* with `mono_video` example and simplified setup:
  - Fork used: mwbadran/ORB_SLAM3_Windows (Windows-adapted, mono_video, easy build).
  - Based on: rexdsp/ORB_SLAM3_Windows.
  - Original project: UZ-SLAMLab/ORB_SLAM3 (GPLv3).
- **OpenCV**, **NumPy**, **SciPy Rotation**, and standard Python libs.

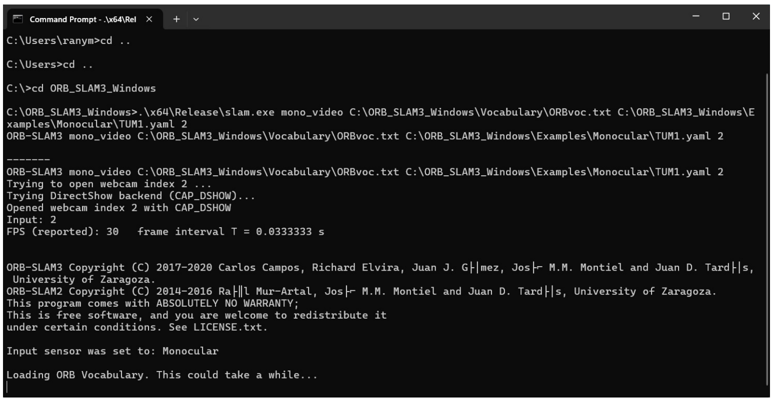
2.2 Why This Stack

- MediaPipe is fast and cross-platform for hand landmarks on a standard webcam.
- OBS+VirtualCam solves the Tello "single viewer" limitation by duplicating/bridging the stream.
- The Windows fork of ORB-SLAM3 reduces setup friction on VS 2022, letting us focus on integration.

2.3 Tooling Screenshots



(a) OBS scene and VirtualCam.



(b) ORB-SLAM3 mono_video tracking window.

Figure 2.1: Screenshots of the tooling used in the project.

3 System Overview and Design

3.1 Data Flow

1. **Gesture pipeline:** webcam → MediaPipe Hands → gesture classifier → debouncing → mapped command → Tello UDP command.
2. **SLAM pipeline:** Tello UDP video → OBS → VirtualCam → ORB-SLAM3 `mono_video` → KeyFrameTrajectory logs → Python tails logs.
3. **Safety/awareness:** Python optionally uses SLAM yaw coverage to confirm 360° rotation.[1]

3.2 Gesture Vocabulary (current mapping)

Gesture	**Action**	**Notes**
Palm (all fingers up + thumb)	`takeoff`	Entry to flight
Call Me (thumb+pinky up)	`land`	Safe landing
Thumbs Up	`forward 20`	Reliable; easy to hold steady
Thumbs Down	`back 20`	Reliable; paired with forward
Peace (index+middle up)	`cw 30`	Yaw right in 30° steps
Okay (thumb-index loop)	`ccw 30`	Yaw left in 30° steps
Fist	`down 20`	Altitude down
Spiderman (thumb+index+pinky)	`cw 360` (12×30°)	Triggers 360°; we tail SLAM

Table 3.1: Gesture map used in our Python controller (distances in cm).

[1]We primarily use SLAM as a *security awareness aid*, not as a hard collision-avoidance controller.

3.3 Debouncing and Cooldowns

We maintain a fixed-size deque of recognized gestures and only accept a *stable* gesture after N consistent frames. We also apply a command cooldown (∼2.5 s) to prevent rapid repeated commands.

3.4 360° Rotation Verification (30° steps)

On a long-press of *Spiderman*, we execute `cw 360` as twelve `cw 30` steps with small pauses. A background thread tails `KeyFrameTrajectory.txt`, converts quaternions to yaw, and accumulates absolute yaw change to confirm ≈360° coverage. This is used as a *sanity check*, not a full safety proof.

4 Implementation on Windows

4.1 Environment and Dependencies

- Windows 11 Pro; Visual Studio 2022 for the ORB-SLAM3 fork.
- Python 3.9+, packages: `opencv-python`, `mediapipe`, `numpy`, `scipy`, (optional) `pyvirtualcam`.
- OBS with VirtualCam (for bridging the UDP feed into a camera device).

4.2 ORB-SLAM3 Windows Fork (Mono Video)

We used the **mwbadran/ORB_SLAM3_Windows** fork (*ORB-SLAM3 Windows Fork with Mono Video & Simplified Setup*). Compared to prior Windows ports, this adds a `mono_video.cpp` example and a ready-to-build solution. After building (Release), usage is (examples shown):

Listing 4.1: Running ORB-SLAM3 mono_video on Windows

```
cd C:/ORB_SLAM3_Windows/

# Webcam index 0 with calibration TUM1.yaml
.\x64\Release\slam.exe mono_video ^
  C:\ORB_SLAM3_Windows\Vocabulary\ORBvoc.txt ^
  C:\ORB_SLAM3_Windows\Examples\Monocular\TUM1.yaml ^
  0

# Video file with TUM1.yaml
.\x64\Release\slam.exe mono_video ^
  C:\ORB_SLAM3_Windows\Vocabulary\ORBvoc.txt ^
  C:\ORB_SLAM3_Windows\Examples\Monocular\TUM1.yaml ^
  "C:\Users\W10\Desktop\video.mp4"
```

Notes from the fork: Eigen 3.3.7, OpenCV 3.2.0 (DLLs included), binary/text vocabulary toggle, Windows-specific `_WINDOWS` defines. See the References section.

4.3 OBS Setup (Tello UDP → VirtualCam)

1. Add a *Media Source* pointing to `udp://0.0.0.0:11111`.
2. Start *Virtual Camera* in OBS.
3. Launch ORB-SLAM3 `mono_video` and pass the VirtualCam index (often 0/1/2 on Windows).

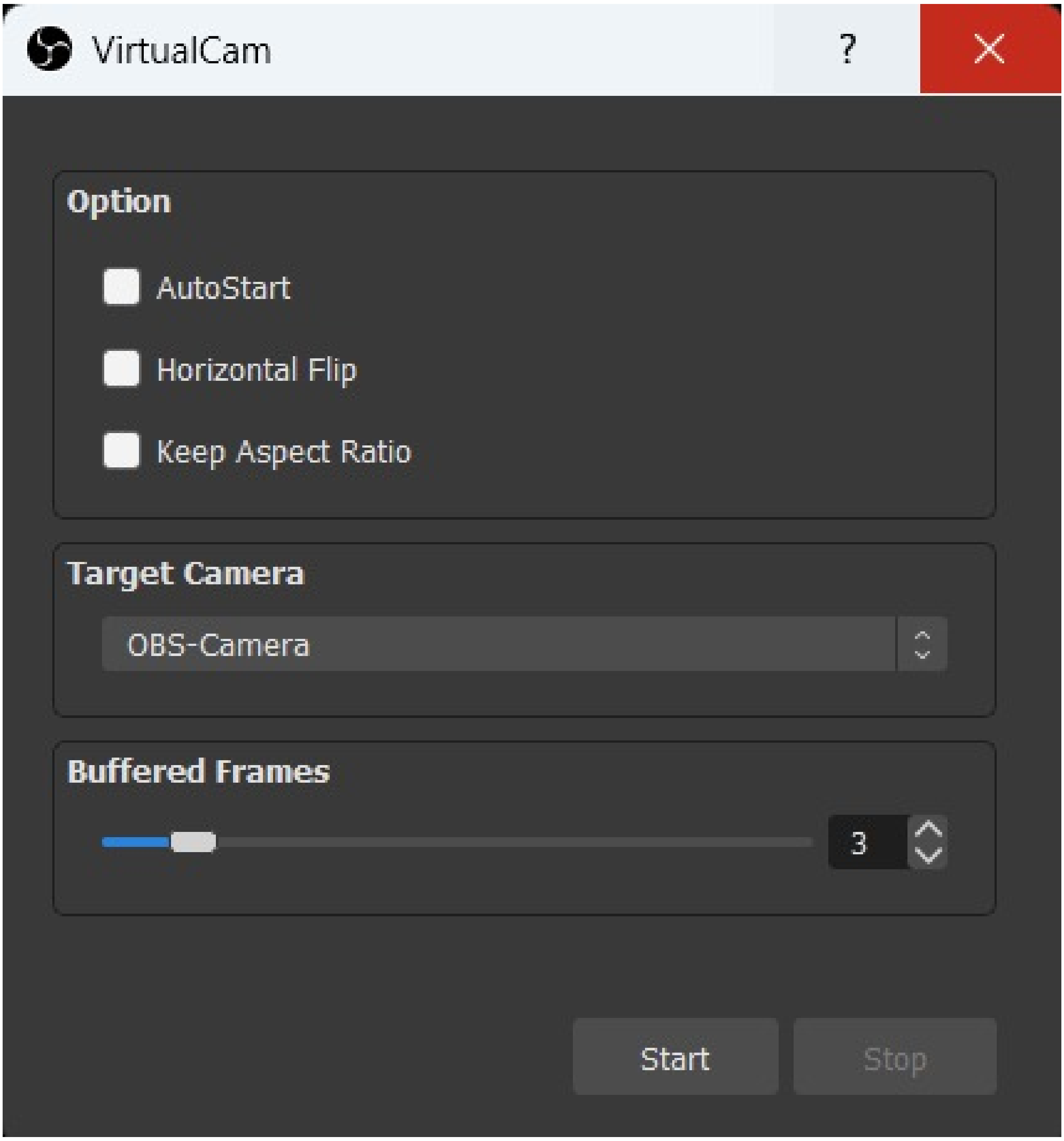


Figure 4.1: OBS configuration for Tello UDP → VirtualCam.

4.4 Python Controller (How to Run)

1. Connect to Tello Wi-Fi (`TELLO-xxxx`), ensure Windows Firewall allows Python (UDP).
2. `pip install opencv-python mediapipe numpy scipy` (and `pyvirtualcam` if you plan to mirror video).
3. Run the controller:

```
python gesture_controller.py --webcam-index 0
```

4. Show gestures to the PC webcam; watch console logs for SDK responses (`ok/error`).

4.5 Conventions and Structure

- **Debounce:** fixed-length history, require a minimum of stable frames.
- **Cooldown:** 2.5 s between accepted commands for safety.
- **SDK State:** track `in_flight`, suppress moves on motor fault, robust handshake.
- **Rotations:** consistently **30°** steps for predictability; 360° is 12 steps.
- **Logging:** console prints on incoming responses (e.g., "not joystick"), retry logic for UDP sends.

4.6 Selected Code Snippets

For brevity, we include minimal excerpts; paste your full file in Appendix **??** later.

Gesture Mapping (excerpt)

```
GESTURE_MAP = {
  "palm": "takeoff",
  "call_me": "land",
  "thumbs_up": "forward 20",
  "thumbs_down": "back 20",
  "peace": "cw 30",
  "okay": "ccw 30",
  "fist": "down 20",
  "spiderman": "cw 360",
}
```

SLAM Log Tailing (yaw accumulation, excerpt)

```
from scipy.spatial.transform import Rotation as R
# ... read KeyFrameTrajectory.txt lines ...
qx, qy, qz, qw = map(float, parts[4:])
yaw = R.from_quat([qx, qy, qz, qw]).as_euler('xyz', degrees=True)[2]
# accumulate |delta yaw| to confirm ~360 coverage
```

5 Experiments

5.1 Bench Tests

- Verified reliable **thumbs up/down** detection for forward/back (or altitude, depending on mapping).
- **Peace/Okay** consistently triggered 30° yaw steps.
- 360° routine executed as 12×30° with expected ok responses from the SDK.

6 Difficulties and Lessons Learned

6.1 Pivoting the Idea and Project Scope

We began with *drowning detection* and then a *selfie drone*. Both directions turned out to be ill-fitting: (i) drowning detection did not substantively use ORB-SLAM output within the scope/timeline of a real-time systems course; (ii) selfie mode collided with Tello's *single viewer* video constraint and with the need to run both Python and SLAM consumers. This led us to **GesturePilot**: PC-webcam gestures for control, and a parallel SLAM pipeline used for awareness and 360° verification.

6.2 Tello "Single Viewer" and Stream Bridging

Tello's UDP stream practically allows only one stable consumer. Running Python (OpenCV), ORB-SLAM3, and OBS directly on the same stream caused contention and intermittent frame starvation. **Mitigation:** We made OBS the primary consumer of `udp://0.0.0.0:11111`, then used *VirtualCam* to expose a camera device that SLAM could open reliably. This adds a hop and CPU overhead, but it eliminated stream races and simplified device selection from SLAM.

6.3 ORB-SLAM3 on Windows: Build and Runtime Issues

- **Build friction.** Dependency versions (Eigen/OpenCV) and Visual Studio toolchains had to match the Windows fork. We spent time untangling missing DLLs and ABI mismatches.
- **Camera backends.** The fork's `mono_video` tries multiple backends (DirectShow, MSMF, FFMPEG). On some machines, only DirectShow opened the VirtualCam; on others, MSMF worked but with unstable FPS.
- **Calibration.** Using TUM1.yaml as a generic monocular calibration works, but if the virtual camera's FOV/distortion differs, tracking jitter increases. Real calibration would improve map consistency.
- **Log hygiene.** We tail `KeyFrameTrajectory.txt`. If SLAM is restarted without clearing the file, offsets/old samples pollute the yaw accumulation. We added a clear-log step before scans.

- **Timing.** FPS reported by backends can be inaccurate; we guard against it by clamping and sleeping to avoid flooding SLAM.

6.4 OBS/VirtualCam Quirks

- **Device index churn.** VirtualCam's index can change between reboots or when other apps claim cameras.
- **Color space/format.** Some combinations (NV12/YUY2) yielded washed colors or higher CPU usage.
- **Latent buffering.** OBS sources can buffer UDP; we disabled extra buffering for lower end-to-end lag.

Mitigation: We documented a fixed OBS scene (single Media Source $\rightarrow$ VirtualCam), and we probe backends in SLAM starting from DirectShow.

6.5 Python Environment, Firewalls, and Networking

- **Firewall.** Windows Firewall occasionally blocked UDP commands; first run requires allowing Python.
- **Shell differences.** Installing `mediapipe` and `pyvirtualcam` behaved differently across `cmd`, Git Bash, and Windows Terminal; PATH and Visual C++ Build Tools matter.
- **Tello handshake.** The first `command` sometimes times out. We added retries and a re-enter-SDK path when "not joystick" or transient errors appear.

6.6 Gesture Recognition and Human Factors

- **Lighting and background.** Backlighting and busy backgrounds reduce hand landmark quality.
- **Hand orientation.** Rotated palms or partial occlusion (sleeves, bracelets) confuse finger state.
- **Index-only instability.** Isolating a single finger is less stable than thumbs/-peace/okay.
- **Left vs right hand.** Mirroring and angle dependence can flip vertical tests (up/down) if thresholds are too tight.

Mitigation: We use debouncing (history window), a cooldown ($\approx 2.5 s$), $and favor robust gestures (thu$
$correct posture before a command is accepted$.

6.7 Safety, Fault Handling, and Recovery

- **Motor.** Upon "motor stop" , we mark `in_flight=false` and allow only `land`.
- **Command storms.** Without debouncing/cooldown, rapid repeats (e.g., `cw 30` $\times$ N) can queue too aggressively.
- **360° verification.** Early on we used 90° steps; we switched to **12×30°** with pauses and yaw accumulation from SLAM logs.

7 Solutions to the Problems

This chapter consolidates each major problem we hit and the practical solution we implemented, quoting the relevant parts of our code.

7.1 Single-Viewer Stream → OBS & VirtualCam Bridge

Problem: Tello's UDP stream is unreliable with multiple direct consumers (Python, SLAM, OBS). **Solution:** Make OBS the *single* consumer of `udp://0.0.0.0:11111` and expose a VirtualCam that SLAM opens. In our C++ `mono_video` we try backends so VirtualCam opens reliably:

```
#ifdef _WIN32
// Prefer DirectShow for VirtualCam
if (cap.open(camIndex, cv::CAP_DSHOW)) { opened = true; }
#endif
if (!opened) {
  if (cap.open(camIndex, cv::CAP_ANY)) { opened = true; }
}
```

This avoids stream contention and keeps SLAM stable.

7.2 SDK Handshake Reliability and Retries

Problem: First command sometimes times out; occasional "not joystick" errors. **Solution:** Robust send with retries and special handling:

```
def send_command(self, command, timeout=7.0, retries=1):
    ...
    while time.time() - start < timeout:
        if self.response is not None:
            text = self.response; self.response = None
            if text == 'ok': return True, "ok"
            elif 'error' in text.lower():
                self.last_error = text
                if 'not joystick' in text.lower() and i < retries:
                    self.send_command('command') # re-enter SDK mode
                    time.sleep(1.0)
```

```
                break
            return False, text
```

We also set speed after entering SDK and toggle the video stream to a known state.

7.3 Fault-Safe Control (Motor)

Problem: Movement after a motor fault is unsafe. **Solution:** Suppress moves when faults are active; only allow land (and queries):

```
if self.motor_fault and cmd_root not in ('land', 'battery?'):
    since = time.time() - self.last_motor_fault_time
    if since < 6.0:
        return False, "Motor Fault"
    else:
        self.motor_fault = False
```

7.4 Gesture Stability: Debounce & Cooldown

Problem: Noisy, flickering gesture classifications. **Solution:** Use a history window and accept only stable gestures; impose cooldown:

```
DEBOUNCE_HISTORY = 8
STEADY_REQUIRED = 4
COMMAND_COOLDOWN = 2.5

self.gesture_history.append(current_gesture)
if len(self.gesture_history) == DEBOUNCE_HISTORY:
    last_gest = self.gesture_history[-1]
    if last_gest is not None and list(self.gesture_history).count(last_gest) >=
        STEADY_REQUIRED:
        stable_gesture = last_gest
```

Why it helps: Requires several consistent frames before a command is emitted; then enforces a short pause before accepting the next one.

7.5 Robust Gesture Rules (Thresholded Landmarks)

Problem: Misclassifications from finger ambiguity (lighting, occlusion). **Solution:** Finger states use relative distances scaled by hand size; e.g., extended if tip–PIP gap > ratio:

```
def _is_extended(lm, tip_idx, pip_idx, hand_size, min_ratio=0.22):
    return (lm[tip_idx].y < lm[pip_idx].y) and \
        (_dist(lm[tip_idx], lm[pip_idx]) > hand_size * min_ratio)
```

We prefer robust gestures: thumbs_up/down, peace, okay, fist for altitude down.

7.6 360° Rotation in 30° Steps + SLAM Log Hygiene

Problem: Earlier scans mixed old and new log samples; rotation steps too coarse. **Solution:** Clear logs before scanning; rotate in **12×30°** with pauses; tail only new lines:

```
def clear_log(self):
    if os.path.exists(self.logfile):
        open(self.logfile, 'w').close()
    self._last_len = 0
    self.sector_counts.clear()
...
steps = 360 // INCREMENTAL_ANGLE # INCREMENTAL_ANGLE = 30
for i in range(steps):
    if not self.tello.safe_rotate_cw(INCREMENTAL_ANGLE): break
    time.sleep(INCREMENTAL_PAUSE)
```

7.7 Forward/Back Movement Policy (Gating Toggle)

Problem: Early design blocked forward/back until mapping was "ready". **Solution:** Make it configurable (SLAM_GATING_ENABLED) and allow movement when off:

```
SLAM_GATING_ENABLED = False

def _mapped_direction_ok(self, direction):
    if not SLAM_GATING_ENABLED:
        return True
    # else check sector coverage by yaw...
```

7.8 UDP Receiver Thread and Timeouts

Problem: Lost responses and blocking reads. **Solution:** Dedicated receive thread with timeouts; shared self.response buffer:

```
def _receive_response(self):
    while not self.stop_event.is_set():
        try:
            self.sock.settimeout(1.0)
            data, _ = self.sock.recvfrom(1024)
            text = data.decode(errors='ignore').strip()
            if text:
                self.response = text
        except socket.timeout:
            continue
```

This keeps the control loop responsive and avoids deadlocks.

8 Hardware Constraints

Below are the practical constraints we observed with small indoor quadcopters like Tello, and the mitigations we adopted.

Battery and Thermal

- **Short sessions.** Real flight windows are much shorter than nominal due to hover, frequent yawing, and retries.
- **Voltage sag.** Rapid maneuvers can trigger brown-out behavior sooner at low charge.
- **Heat.** Extended hover for mapping warms motors and SoC, increasing drift and fan noise.

Mitigation: Keep flights short; plan scripts to be testable in <10 minutes; cool-down between sorties; start at high state-of-charge and warn under a minimum battery threshold.

Radio Link (Wi-Fi)

- **Interference.** Congested 2.4 GHz environments cause UDP loss and latency.
- **Range and occlusion.** Walls/metal cabinets attenuate the control link and the video downlink.

Mitigation: Test in a quiet RF room; keep the operator close; face antennas toward the drone; avoid large crowds and microwaves/routers.

Sensors and State Estimation

- **No obstacle sensors.** Tello has no front depth; SLAM mapping is *awareness*, not hard avoidance.
- **Vision/flow limits.** Downward optical flow and stabilization degrade on dark, shiny, or textureless floors and in low light.

- **No GPS.** Indoors drift accumulates; yaw corrections can still translate.

Mitigation: Keep clear perimeters; add visual texture to floors (checkerboards/rugs); ensure bright, even lighting; keep yaw steps modest (30°) with pauses.

Camera and Perception

- **Latency and compression.** The downlink is compressed, adding delay and artifacts for SLAM.
- **Auto-exposure.** Rapid brightness changes harm SLAM feature stability.

Mitigation: Stable lighting; fewer fast rotations; allow a pause after each step before sampling SLAM.

Control Envelope and Airflow

- **Tight spaces.** Wash from the propellers near walls/ceilings induces drift.
- **Floor effects.** Very low altitude amplifies ground effect and optical-flow errors.

Mitigation: Maintain a safe standoff from walls/ceiling; operate at moderate altitude; use small movement distances (e.g., 20 cm) indoors.

Maintainability and Spares

- **Propellers/guards.** Minor nicks create vibration and noise.
- **Batteries.** Aging cells reduce usable flight time and are more sensitive to cold.

Mitigation: Keep spare props and at least two batteries; visually inspect props between runs.

Summary of Constraints and Mitigations

Constraint	Symptoms Observed	Mitigation Adopted
Single video viewer	SLAM/Python/OBS contention, frame starvation	OBS as primary, Virtual-Cam to SLAM
Battery/thermal	Short flights, drift after long hover	Short sorties, cooldowns, battery threshold
Wi-Fi interference	Command/video lag, packet loss	Quiet RF space, short range, line-of-sight
No obstacle sensing	Risk near objects	30° steps, pauses, operator line-of-sight
Flow/lighting limits	Hover wobble on dark/shiny floors	Add texture, increase light, moderate altitude
Gesture ambiguity	Index-only instability, occlusions	Debounce/cooldown, robust gestures, HUD feedback

Table 8.1: Key constraints we hit and how we addressed them.

9 Future Work

Integrate depth sensing and/or Visual–Inertial SLAM for true obstacle avoidance. Today our SLAM usage is a heuristic awareness aid (e.g., 360° rotation verification). The most impactful next step is to augment mapping with *depth* (stereo or RGB-D) and/or fuse IMU measurements (VI-SLAM) to achieve robust, real-time *collision avoidance*. Concretely:

- Add a depth source (stereo or RGB-D) or switch to a VI-SLAM configuration to stabilize scale and pose.
- Maintain a short-horizon occupancy/safety "bubble" around the drone and gate commands by live depth checks.
- Calibrate intrinsics/extrinsics and profile latency to keep end-to-end control under real-time constraints.

This would turn GesturePilot from *awareness with operator-in-the-loop* into a system with *active avoidance*, which is the single most important improvement for indoor safety and reliability.

10 Potential Emergency Applications

Although GesturePilot was developed for coursework, the approach lends itself to **rapid indoor maneuvering during emergencies** (e.g., scouting a *building on fire* where GPS is unavailable). A small quadcopter can be flown down hallways and into rooms while the operator issues *gesture* commands from a safe distance. The built-in 360° scan routine (30° steps) provides quick situational awareness, and the depth/VI-SLAM upgrade in Chapter 9 could support basic collision avoidance in low-visibility environments.

Caveats: commodity platforms like Tello have short battery life, limited radio range, and no thermal or smoke-penetrating sensors; MediaPipe and SLAM degrade in heavy smoke or poor lighting. Our system is a *concept* for fast exploration and telepresence, not a replacement for professional rescue equipment.

11 Conclusion

GesturePilot demonstrates an end-to-end **gesture-controlled** drone with basic SLAM-informed awareness on Windows, navigating real engineering constraints: OS builds, single-viewer streams, and noisy hand gestures. The result is a practical, reproducible stack suitable for real-time systems coursework.

Acknowledgments

We thank Prof. Dan Feldman for supervision and the RBD Lab for support. We also acknowledge the maintainers of ORB-SLAM3 and the Windows forks used in this project.

Bibliography

[1] C. Campos, R. Elvira, J. J. Gómez Rodríguez, J. M. M. Montiel, and J. D. Tardós, *ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM*, arXiv:2007.11898, 2020. https://arxiv.org/abs/2007.11898.

[2] M. W. Badran, *ORB_SLAM3_Windows* (Windows-adapted fork with mono_video), GitHub repository, 2024. https://github.com/mwbadran/ORB_SLAM3_Windows.

[3] rexdsp, *ORB_SLAM3_Windows* (earlier Windows port), GitHub repository. https://github.com/rexdsp/ORB_SLAM3_Windows.

[4] UZ-SLAMLab, *ORB_SLAM3 (original project, GPLv3)*, GitHub repository. https://github.com/UZ-SLAMLab/ORB_SLAM3.

[5] Google, *MediaPipe Hands*, https://developers.google.com/mediapipe.

[6] OBS Project, *Open Broadcaster Software & VirtualCam*, https://obsproject.com/.

[7] Ryze/ DJI, *Tello SDK Documentation*, (online).