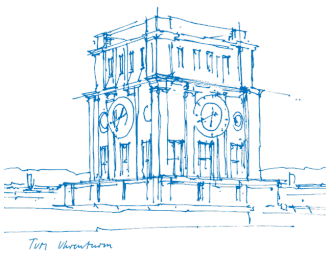
TUM Campus Guide Drone

Embedded Systems, Cyber-Physical Systems and Robotics

Iyed Lahiani, Wassim Mezghanni, Refia Daya, Suzan Cansu Celik, Fathia Ismail, Maher Bouars, and Mohamed Aziz Bouafif

July 21, 2025



Contents

Abstract — This report presents the development of a semi-autonomous drone system capable of real-time path planning and obstacle avoidance, using the DJI Tello as a test platform. Designed for indoor navigation within a simulated model of the TUM Campus Heilbronn, the system integrates the A* algorithm for efficient pathfinding and computer vision techniques via OpenCV for dynamic obstacle detection. Python and the `djitellopy` library serve as the backbone for drone control and image processing. Several implementation challenges were encountered, including limited camera resolution, calibration inconsistencies, and hardware faults. The report concludes with a discussion of potential improvements, such as sensor fusion, better calibration mechanisms, and hardware upgrades to improve robustness, accuracy, and autonomy in real-world environments.

1 Introduction

Unmanned aerial vehicles (UAVs), commonly known as drones, have seen widespread adoption across various domains including surveillance, delivery, and environmental monitoring. A key challenge in expanding their autonomous capabilities lies in robust real-time navigation and obstacle avoidance. This project addresses that challenge by developing a vision-based control system for the DJI Tello drone, enabling it to navigate a grid-based environment while dynamically detecting and avoiding obstacles.

The system is designed to simulate autonomous navigation through a prototype representation of the **TUM Campus Heilbronn**. By integrating the A* **path planning algorithm** with visual obstacle detection using OpenCV, the drone can intelligently adapt its path in real time. When an obstacle is detected through onboard camera feed analysis, the corresponding grid cell is marked as blocked, and a new path is computed on-the-fly.

Python is used as the primary programming language, leveraging the `djitellopy` library to interface with the Tello drone's SDK. The drone receives high-level commands such as directional movement and takeoff/landing, while image processing techniques—such as edge detection, region of interest (ROI) analysis, and brightness thresholding—are applied to identify potential hazards in its path.

The overarching objective of this project is to ensure safe, reliable, and collision-free drone operation in semi-structured indoor environments. This contributes to the broader vision of autonomous aerial navigation systems that require minimal human intervention while operating in complex and dynamic settings.

2 Hardware and Software Requirements

2.1 Hardware

The Tello drone is a small quadcopter with a Vision Positioning System and onboard camera. It records in 1280x720 30p mode and uses Wi-Fi for communication.

2.2 Software

Python is used with the `djitellopy` library to control the drone. The library provides a simple interface to communicate with the DJI Tello SDK, allowing for high-level drone commands such as takeoff, movement, and video streaming.

```
from djitellopy import Tello
```

Listing 1 Importing djitellopy

Additionally, OpenCV is used for image processing and computer vision tasks, such as edge detection and obstacle recognition. NumPy is used for numerical operations and matrix manipulation:

```
import cv2
import numpy as np
```

Listing 2 Importing OpenCV and NumPy

3 Methodology and Implementation

3.1 Connection with Tello Drone

To control the Tello drone, we must first establish a connection between the laptop and the drone via Wi-Fi. The 'djitellopy' Python library provides a convenient API to interact with the drone using UDP commands.

Once connected, we also initialize the control velocities and activate the video stream to start receiving live frames.

```
tello = Tello()
tello.connect()
tello.streamon()
```

Listing 3 Initialize the Tello Drone

After connection, we can also query internal status from the drone to verify readiness:

```
print("SDK version:", tello.
    get_sdk_version())
print("Battery level:", tello.
    get_battery(), "%")
print("Current flight mode:", tello
    .get_flight_mode())
```

Listing 4 State Query After Connecting

Once the video stream is started via `streamon()`, we use a background thread to grab frames:

```
frame_read = tello.get_frame_read()
frame = frame_read.frame  # BGR
    image (OpenCV format)
```

Listing 5 Accessing Video Stream

Finally, when the program ends or the drone is no longer needed, we should safely shut it down to release system resources and stop communication:

```
tello.streamoff()
tello.end()
```

Listing 6 Shutdown Sequence

This completes the connection and initialization process. The drone is now ready to accept control commands and stream video data in real-time.

3.2 Path Finding

We implemented **A* (A-Star) path-finding algorithm** which is commonly used to compute the **shortest path** between two points on a grid while avoiding obstacles. We used a priority queue to always expand the most promising node, which is the one with the lowest estimated total cost to the goal. The grid is represented as a 2D list, where cells with a value of 0 are walkable and cells with a value of 1 represent obstacles. The algorithm starts from a given start position and explores neighboring cells (up, down, left, right) by calculating the cost of reaching each neighbor, combining the actual cost so far with **a heuristic estimate** of the remaining distance to the goal. We worked on maintaining a set of visited nodes to prevent revisiting them, and the path is built step by step as it searches. Once the goal is reached, our function returns the full path from start to goal; if no path exists, it returns None. We implemented this algorithm to ensure both optimality and efficiency for grid-based navigation problems.

3.3 Obstacle Avoidance

The obstacle avoidance system in our drone project plays a critical role in enabling safe and autonomous navigation within the mapped environment of the TUM Campus Heilbronn. The drone relies on a combination of computer vision and grid-based path planning to dynamically adapt its trajectory in the presence of unforeseen obstacles.

3.3.1 Design and Workflow

The core idea is to allow the drone to follow an initial path computed using the A* algorithm, while simultaneously using real-time visual feedback from its onboard camera to detect new obstacles. Once an obstacle is detected, the corresponding location on the internal grid map is updated, and the drone recalculates a new path to the goal, bypassing the detected obstacle.

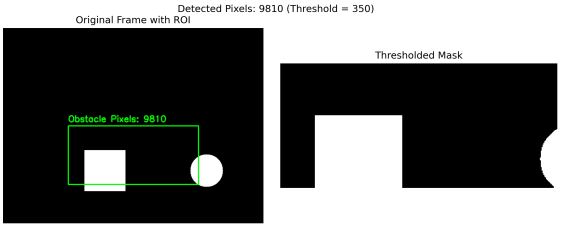


Figure 1 Simulated obstacle detection with ROI and threshold mask

1. **Path Initialization:** The drone begins by computing an optimal path from the start point to the destination using A* on the predefined grid map.

2. **Frame Capture:** During flight, each frame is retrieved from the Tello's camera feed.

3. **Obstacle Detection:** The system defines a Region of Interest (ROI) within the frame. The ROI is processed using grayscale conversion, Gaussian blur, brightness thresholding, and edge detection (Canny algorithm). The number of non-zero pixels (edges or bright regions) in the ROI is counted.

4. **Decision Logic:** If the count exceeds a predefined threshold, the region is flagged as an obstacle.

5. **Map Update and Replanning:** The obstacle's position is mapped back to a grid cell which is then marked as blocked. A* is run again to generate a new safe path.

6. **Continued Navigation:** The drone follows the new path, continuing the detection loop.

3.3.2 Code Example

The following snippet illustrates the logic used for detecting obstacles and updating the drone's path accordingly:

```
frame = tello.get_frame_read().
    frame
frame = cv2.resize(frame, (480,
    360))
gray = cv2.cvtColor(frame, cv2.
    COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray,
    (5, 5), 0)
_, thresholded = cv2.threshold(
    blurred, 60, 255, cv2.
    THRESH_BINARY)
edges = cv2.Canny(thresholded, 30,
    100)
h, w = edges.shape
roi = edges[h//3:2*h//3, w//3:2*w
    //3]
non_zero_count = cv2.countNonZero(
    roi)
if non_zero_count >
    OBSTACLE_THRESHOLD:
     print("Obstacle detected")
     map_grid[next_position[0]][
        next_position[1]] = 1
     path = a_star_search(map_grid,
        current_position, goal)
```

Listing 7 Obstacle detection and avoidance logic

3.3.3 Visualization

To demonstrate the visual detection mechanism, we provide a simulated frame with artificially placed obstacles. The system overlays the ROI, displays edge detection, and prints the count of detected pixels. An example of this visualization can be seen in Figure 1.

4 Issues

Throughout the development and testing phases of our autonomous drone control system designed for safe navigation across the prototype map of the TUM Campus Heilbronn we encountered several significant technical challenges that impeded the optimal performance of the system. These issues primarily revolved around sensor reliability, calibration accuracy, and hardware stability. Below is an in-depth overview of the critical problems faced:

- **Degraded Obstacle Avoidance Performance**
  One of the core functionalities of the project real-time obstacle avoidance was severely compromised due to the substandard quality of the vision sensor (camera) used for environmental perception. The low-resolution imaging capability limited the drone's ability to accurately detect and interpret surrounding obstacles, particularly in dynamic or complex environments. This deficiency undermined the effectiveness of autonomous navigation and increased the risk of potential collisions, defeating a primary objective of the system: safe and intelligent mobility.

- **Inconsistent and Unreliable Calibration Procedures**
  Accurate drone calibration is fundamental to achieving precise control and stable flight behavior. However, the calibration process in our implementation was notably inconsistent, resulting in erratic drone responses and unpredictable flight patterns. These inconsistencies were primarily due to software-hardware synchronization issues and limitations in the calibration interface. The unreliability of this process introduced operational inefficiencies and made repeatable, controlled testing significantly more difficult.

- **Critical Motor System Malfunctions**
  A recurring motor-related error posed a major obstacle to system reliability. In multiple instances, the drone failed to initiate due to a motor malfunction, effectively halting the startup sequence. This error was likely linked to a combination of hardware defects and power distribution irregularities within the propulsion system. The inability to consistently activate the drone's motors not only delayed the experimental timeline but also raised concerns about the long-term viability and robustness of the drone's mechanical systems.

These technical difficulties underscore the importance of investing in high-quality sensory equipment, establishing reliable calibration protocols, and ensuring mechanical resilience when designing autonomous aerial systems. Addressing these areas is vital for achieving the project's overarching goal: a dependable, collision-free drone capable of safely navigating complex and unstructured environments with minimal human intervention.

5 Improvements

While the current system provides a foundational implementation of real-time obstacle avoidance using computer vision, several improvements could significantly enhance the drone's autonomous capabilities and robustness.

5.1 Multi-directional Obstacle Detection

At present, the obstacle avoidance system relies solely on the drone's front-facing camera. This limits the field of view and leaves the drone vulnerable to collisions from the sides, above, or below. A more comprehensive system would integrate visual or infrared sensors in all directions to ensure full 360-degree situational awareness. This would enable the drone to detect and respond to hazards regardless of their approach angle.

5.2 Integration of Additional Sensors

The reliance on a single low-resolution camera impacts the accuracy and reliability of obstacle detection. Incorporating additional sensors such as ultrasonic range finders, time-of-flight sensors, or LiDAR modules could significantly improve depth perception and environmental understanding. Fusing these sensory inputs with visual data would enhance detection accuracy, reduce false positives, and enable operation in low-light or visually complex environments.

5.3 Improved Calibration Mechanism

To address the inconsistency in drone behavior due to calibration issues, a more robust and user-friendly calibration process should be developed. Automated calibration routines or the use of external motion capture systems could ensure more precise control, stability, and reproducibility during flight.

5.4 Hardware Upgrades

Replacing the onboard camera with a higher-resolution module or using a more capable drone platform would improve visual feedback quality and system performance. Additionally, addressing motor system faults—possibly by upgrading power delivery components or using a more stable propulsion system—would reduce startup failures and improve operational reliability.

5.5 Software Optimization and Fail-Safes

Implementing smoother software-to-hardware synchronization and introducing fail-safe mechanisms (e.g., emergency stop when path is unclear or inconsistent) would further reinforce system robustness and user safety.

Together, these improvements would move the system closer to the goal of safe, intelligent, and fully autonomous drone navigation in real-world environments.

6 Conclusion

The implementation of a computer vision–based navigation system for the Tello drone has demonstrated the feasibility of low-cost, semi-autonomous indoor flight in structured environments. Through the use of A* path planning and real-time visual obstacle detection, the system effectively adapts its trajectory to dynamic changes in its surroundings. However, several limitations such as low sensor resolution, unreliable calibration, and mechanical instabilities highlight the need for more robust hardware and refined algorithms. Addressing these issues through the integration of multi-directional sensors, higher-quality cameras, and improved fail-safes would significantly enhance system reliability. Future work should explore multi-agent coordination, SLAM integration, and outdoor adaptation to move closer toward fully autonomous aerial navigation systems.

References

[1] Ryze Robotics, "Tello SDK 2.0 User Guide," https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf, 2024, accessed: 2024-05-07.

[2] DJITelloPy, https://github.com/damiafuentes/DJITelloPy

[3] TIERS Tello Driver for ROS, https://github.com/TIERS/tello-driver-ros

[4] Hart, P. E., Nilsson, N. J., Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107.

[5] Badrloo, S., Varshosaz, M., Pirasteh, S., Li, J. (2022). Image-Based Obstacle Detection Methods for the Safe Navigation of Unmanned Vehicles: A Review. *Remote Sensing*, 14(15), 3824.

[6] Rodriguez, A. A., Shekaramiz, M., Masoum, M. A. S. (2024). Computer Vision-Based Path Planning with Indoor Low-Cost Autonomous Drones: An Educational Surrogate Project. *Drones*, 8(4), 154.

[7] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal*, 25, 120–123.

[8] Ryze Robotics. (2018). Tello SDK 2.0 User Guide. Retrieved from https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf

[9] Damiafuentes. (2019). DJITelloPy Python Library. Retrieved from https://github.com/damiafuentes/DJITelloPy