Final Report

Table of Contents

Autonomous Drone landing using Aruco Marker

Supervisor:

Dr Daniel Franklin, Senior Lecturer, School of Electrical and Data Engineering.

Introduction

Drones have revolutionized industries by offering versatile solutions in logistics, agriculture, surveillance, and search-and-rescue operations and many more. Their ability to access hard-to-reach areas and perform tasks autonomously has led to increased efficiency and cost-effectiveness in these sectors. However, as drone usage expands, challenges such as limited battery life and the need for precise operational efficiency become more pronounced. One critical capability for drones is precise autonomous landing, essential for tasks like automated battery swapping, package delivery, and docking on predefined platforms. In this project, we focus on developing a system that enables a drone to perform high-accuracy landings using ArUco markers as visual cues. By leveraging advanced computer vision techniques and real-time pose estimation, the drone can identify and align with a marker to land with precision.

Background

Drone operations are often constrained by their reliance on limited battery capacity, typically allowing only 20–30 minutes of flight time depending on the battery size and motor power [1]. When the battery depletes, drones require manual intervention to replace or recharge batteries, a process that is labor-intensive, time-consuming, and impractical in remote or hazardous environments. Autonomous precision landing can address this challenge by allowing drones to land on predefined platforms for battery swapping or recharging, ensuring uninterrupted operations [2].

Precise landing is not only crucial for battery management but also for a range of other applications, including:

1. **Logistics and Delivery**: Accurate placement of packages in predefined zones enhances the efficiency and reliability of delivery services [3].
2. **Autonomous Missions**: Returning drones to specific platforms after completing inspections or mapping of critical infrastructure such as bridges or pipelines[4].
3. **Fleet Operation and Maintenance** : Docking drones on mobile or fixed platforms for refueling or task synchronization or maintenance in multi-drone missions [5].
4. **Remote Operations**: Allowing drones to perform repeated autonomous landings in environments with limited or no human access, such as offshore rigs or disaster zones [6].

Objectives

Traditional GPS-based navigation systems provide approximate positioning but fall short of achieving the precise landing. Vision-based solutions, particularly those leveraging fiducial markers like ArUco, bridge this gap by enabling real-time, high-accuracy localization. These markers offer robust detection and pose estimation capabilities even in environments where GPS signals are unreliable or unavailable [7][8].

we aim to address these challenges by focusing on the following objectives:

1. **Develop a Vision-Based Precision Landing System**:
    - Employ ArUco markers and computer vision techniques to enable the drone to detect, identify, and align with a landing zone.
2. **Achieve High-Accuracy Autonomous Landing**:
    - Ensure the drone lands within a few centimeters of the designated target, suitable for critical operations like battery swapping.
3. **Enhance Autonomous Operations**:
    - Reduce or eliminate the need for manual intervention in repetitive or remote tasks by enabling autonomous landing capabilities.
4. **Prepare for Real-World Applications**:
    - Lay the groundwork for integrating precision landing systems into practical use cases such as logistics, infrastructure maintenance, and autonomous fleet management later.

By achieving these objectives, the project demonstrates a vital capability required for advancing autonomous drone operations, enhancing efficiency and reliability in a variety of real-world scenarios.

Literature review

Vision-Based Landing Techniques

Vision-based landing techniques leverage cameras and computer vision algorithms to enable precise UAV navigation and landing. Among these, fiducial markers such as ArUco and AprilTags have emerged as widely used solutions due to their robustness, computational efficiency, and ease of integration with UAV systems. ArUco markers, introduced by Garrido-Jurado et al. [9], are binary square fiducial markers designed for real-time applications. These markers are lightweight, easy to generate, and allow reliable pose estimation, making them particularly suitable for resource-constrained UAV platforms. Each ArUco marker contains a unique binary ID surrounded by a black border, enabling fast and accurate detection of multiple markers simultaneously. In UAV applications, these markers

are placed on landing zones, and a downward-facing camera captures images, which are processed to detect and estimate the UAV's relative position and orientation to the marker. This information is then used to guide the UAV’s descent and ensure precise landing.

Fig: ArUco markers generated from the original Aruco dictionary

ArUco markers offer several advantages in vision-based landing systems. They are computationally efficient, allowing real-time processing even on low-power onboard systems such as Raspberry Pi. Their lightweight design and ease of implementation through open-source libraries like OpenCV make them a cost-effective solution compared to other approaches such as lidar or infrared systems. Additionally, their high accuracy in controlled environments enables sub-centimeter precision, making them ideal for indoor or static outdoor applications. However, their performance is limited in environments with low light, glare, or shadows. Partial occlusions or motion blur caused by UAV movement can also result in detection failures or inaccurate pose estimation, presenting challenges for their application in dynamic or outdoor environments.

To address these limitations, various enhancements to ArUco marker-based systems have been proposed. Adaptive lighting techniques, such as dynamic thresholding and contrast enhancement, can improve detection under varying lighting conditions. Deploying multiple markers in an array can increase redundancy, ensuring that at least one marker remains visible during landing. Researchers have also explored integrating ArUco detection with other sensors, such as inertial measurement units (IMUs) or depth cameras, to improve robustness. For example, Zhang et al. [10] demonstrated that combining fiducial markers with depth sensors significantly enhances landing reliability in GPS-denied environments, although this approach increases computational demands. Furthermore, incorporating deep learning algorithms like YOLO or Faster R-CNN for marker detection has shown potential to improve robustness in complex environments with occlusions or poor lighting.

An alternative to ArUco markers is the AprilTag system, introduced by Olson [11], which offers improved robustness against perspective distortion and lighting variability. AprilTags are particularly effective in outdoor scenarios, making them suitable for applications requiring greater flexibility. However, both ArUco and AprilTags rely heavily on clear visual conditions, and their performance can degrade in extreme environmental settings such as glare, heavy shadows, or rain. The choice between ArUco and AprilTags often depends on

the specific application, with ArUco being preferred for cost-sensitive projects and AprilTags for environments requiring greater tolerance to visual distortions.

Kocer et al. [12] demonstrated a system that combined ArUco markers with predictive control algorithms for UAV landings on moving platforms, achieving sub-centimeter accuracy. While the system excelled in controlled environments, it struggled in fluctuating lighting conditions, underscoring the need for adaptive systems that can respond to real-world challenges.

Alternative Sensor-Based Landing Techniques

Beyond vision-based approaches, various sensors have been investigated for UAV precision landing. Infrared (IR) sensors, ultrasonic sensors, and lidar systems are among the most explored alternatives. Hrabar [13] demonstrated the effectiveness of IR Lock sensors, which rely on infrared beacons for precise positioning. These sensors are particularly effective in low-visibility conditions, such as fog or low-light environments, where cameras may fail. However, the high cost of IR systems makes them less accessible for large-scale or budget-sensitive applications.

Ultrasonic sensors have been employed for altitude measurement during landing. Zhou et al. [14] demonstrated their reliability in providing vertical height data, which is crucial for maintaining safe descent rates. However, their inability to provide horizontal position information limits their utility for precise alignment with a landing zone. Additionally, ultrasonic sensors are prone to interference in windy conditions or over reflective surfaces.

Lidar systems, as explored by Chen et al. [15], offer a comprehensive solution by providing both altitude and horizontal position data. Lidar achieves high accuracy in obstacle-rich environments, making it suitable for complex landing scenarios. However, the high cost and significant onboard processing requirements of lidar systems pose challenges for lightweight UAVs. Hybrid systems that combine lidar with cameras or other sensors have shown promise in balancing accuracy and resource demands.

Limitations and Comparisons

Each approach to precision landing has distinct advantages and limitations. Fiducial markers, such as ArUco and AprilTags, provide a cost-effective and computationally efficient solution for real-time pose estimation. They are well-suited for controlled environments but struggle with environmental challenges, such as low light, occlusions, and rapidly changing conditions. Vision-based techniques, while adaptable and widely applicable, rely heavily on favorable visual conditions, making them less reliable in harsh outdoor environments.

Sensor-based systems, such as those using IR Lock or ultrasonic sensors, address some of these limitations. IR Lock systems excel in low-visibility conditions but are prohibitively expensive for widespread adoption. Ultrasonic sensors, while affordable, lack the ability to provide comprehensive position data, limiting their effectiveness for precision landing tasks.

Lidar systems, on the other hand, offer unparalleled accuracy in both altitude and horizontal positioning but are expensive and resource intensive.

Hybrid systems that combine vision-based techniques with sensor-based approaches present a promising avenue for overcoming these limitations. For example, combining lidar with fiducial markers can enhance reliability in diverse environments, while integrating IMUs with vision-based systems can improve performance in dynamic scenarios.

Hardware & Software used

Hardware:

1. Holybro X650 development kit drone
   1.1. Pixhawk 6c
   1.2. M10 GPS Module
   1.3. SiK Telemetry Radio V3 433
   1.4. Radio Master TX16S RC Controller
   1.5. Radio Master R81 Receiver
2. Samsung 8000 mAh 6S 10C lithium-ion Battery
3. Raspberry pi4
   3.1. Sony IMX477R Pi hq camera
4. Battery pack for pi (10000mah)
5. Aruco marker

Software:

1. ArduPilot
2. QGroundControl (QGC)
3. OpenCV
4. DroneKit-Python Library
5. Aruco marker Detection Library
6. VNC Server/Viewer
7. PuTTY (SSH)
8. PiCamera2 Library
9. MAVLink Protocol

Methodology

The methodology for the project involves a systematic approach to achieving precise autonomous landing using ArUco markers. Each step integrates the drone's hardware and software components to ensure accurate detection, reliable communication, and controlled landing. The methodology is structured as follows:

1. Drone Assembly

Drone assembly is a foundational step that involves integrating various hardware components together. The Holybro X650 development kit was used, and the following components were assembled:

- The **Pixhawk 6C Flight Controller** is an advanced autopilot system that stabilizes the drone and processes input from various sensors to control its motors. It ensures precise flight control and executes autonomous maneuvers such as landing. The autopilot used is the Ardupilot.
- **M10 GPS Module**: provides accurate geolocation data, enabling the drone to navigate effectively and return to the vicinity of the landing zone during return-to-base operations
- **SiK Telemetry Radio V3**: facilitates real-time communication between the Pixhawk and the ground control system, allowing mission planning, parameter adjustments, and in-flight monitoring.
- **Raspberry Pi 4 with Pi HQ Camera v1.0**: captures and processes visual data in real time to detect the ArUco marker and perform pose estimation, guiding the drone during the landing process.
- **RadioMaster TX16S RC Controller and R81 Receiver** allow for manual control during testing and serve as a reliable backup to ensure operational safety if autonomous systems fail.

Fig: raspberry pi4 connected with Pixhawk through USB C to A cable.

The drone is also equipped with a 8000 mAh battery pack to independently power the drone and a 10000mah power bank to power the Pi4 and the Pi camera.

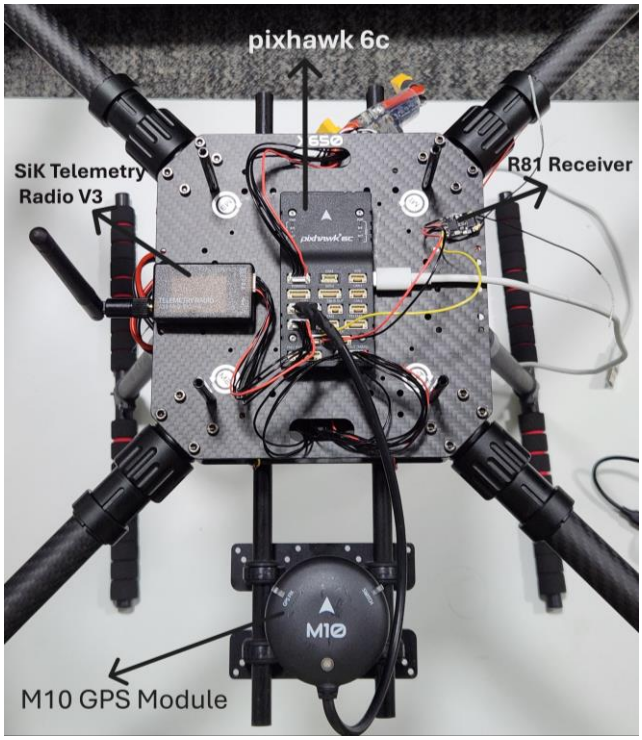

Fig: drone with all the sensors connected to Pixhawk 6c.

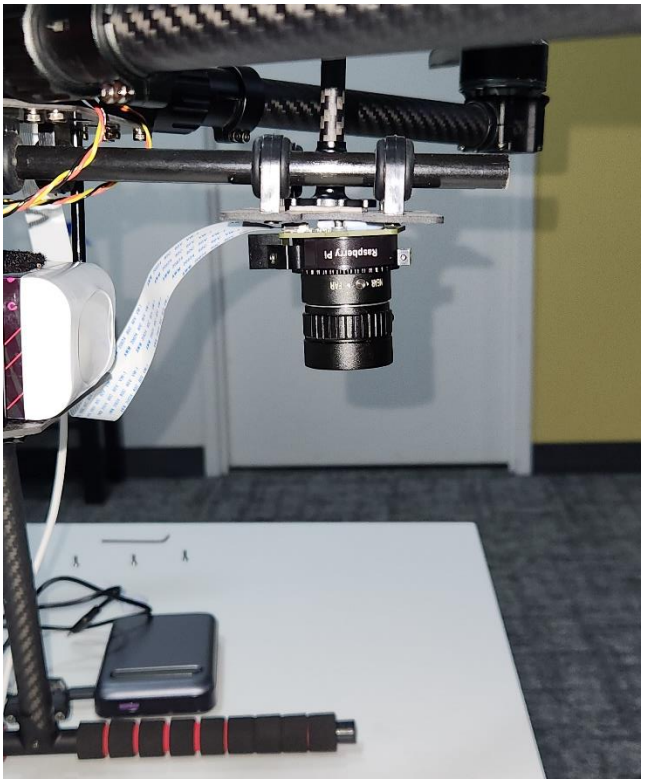

Fig: camera fixed facing downwards

2. Camera Calibration

Camera calibration was conducted to determine the intrinsic and extrinsic parameters of the Pi HQ Camera. Calibration ensures that the images captured by the camera are undistorted, which is essential for accurate marker detection and pose estimation. This process involved:

- Capturing images of a checkerboard pattern from multiple angles and distances to cover the entire field of view.
- Using the OpenCV library to compute intrinsic parameters such as focal length, optical center, and distortion coefficients.
- Verifying calibration results by undistorting test images and ensuring minimal residual distortion. The output of the calibration included the camera matrix and distortion coefficients, which were used in subsequent pose estimation processes.

3. Selection and Generation of the Marker

An ArUco marker was selected as the primary visual target for the landing system due to its computational efficiency and ease of integration. The marker was generated as follows:

- Using OpenCV's cv2.aruco.drawMarker() function to create a marker with a unique ID (e.g., ID 69) and a dictionary (DICT_6X6_250).
- Printing the marker on A4 paper at a size of 150 mm to ensure visibility at varying altitudes.

4. Detection of the Marker

Marker detection is the process of identifying the ArUco marker in the camera's field of view and determining its unique ID. This was implemented to ensure real-time detection of the landing zone. The detection involved:

- Capturing video frames from the Pi HQ Camera.
- Using OpenCV's ArUco library to identify the marker and extract its ID.
- Validating the detection process to handle environmental challenges such as varying light levels or partial occlusions. The successful detection of the marker provided critical input for pose estimation and subsequent landing maneuvers.

5. Pose Estimation

Pose estimation involves calculating the position and orientation of the marker relative to the camera. This step is critical for aligning the drone with the landing zone. The process included:

- Using intrinsic camera parameters obtained during calibration to compute translation and rotation vectors.
- Estimating the 3D pose of the marker, including its position (x, y, z) and orientation (roll, pitch, yaw).
- Continuously updating pose data during descent to dynamically adjust the drone's trajectory. The pose estimation results were sent to the flight controller to guide the landing sequence.

6. Communication Between Raspberry Pi and Pixhawk 6C

Seamless communication between the Raspberry Pi and the Pixhawk 6C flight controller was established to integrate the vision-based data with the drone's control system. This step involved:

- Using the DroneKit-Python library to communicate over the MAVLink protocol.
- Transmitting pose data (translation and rotation vectors) from the Raspberry Pi to the Pixhawk in real-time.
- Enabling the Pixhawk to adjust the drone's roll, pitch, yaw, and descent rate based on the received pose data. This communication ensured that the vision-based system effectively influenced the drone's flight path.

7. Landing Sequence

The landing sequence is a carefully designed flow of operations that transitions the drone from detecting a low battery state to completing a precise autonomous landing on an ArUco marker. The sequence adheres to the logic defined in the code and is as follows:

a) Trigger Return-to-Base (RTB)

- When the drone's battery reaches a critical level (e.g., below 20%), the onboard system triggers a return-to-base (RTB) operation.
- The Pixhawk flight controller, using GPS coordinates of the home location, navigates the drone back to the vicinity of the landing zone.
- During this phase, the drone maintains a safe cruising altitude to avoid obstacles.

b) Activation of Camera and Marker Detection

- Upon reaching the approximate GPS coordinates of the landing zone, the Raspberry Pi activates the Pi HQ Camera for real-time video processing.
- The camera scans for the ArUco marker, which serves as the visual cue for precise landing.

- Once the marker is detected, the Raspberry Pi calculates the relative position of the marker using the marker's ID and intrinsic parameters obtained during calibration.
- If the marker is not detected, the system initiates a search-and-realign process by adjusting the drone's yaw (rotation) to scan the area until the marker is found.

c) Initial Alignment with the Marker

- After detecting the marker, the drone adjusts its position to align with the marker's calculated center.
- Pose estimation data (translation and rotation vectors) is continuously sent from the Raspberry Pi to the Pixhawk over the MAVLink protocol.
- The Pixhawk uses this data to update the drone’s roll, pitch, and yaw to ensure it is properly oriented with the landing zone.

d) Descent Using Pose Feedback

- The drone begins its descent while maintaining alignment with the marker using continuous pose feedback.
- If the marker is temporarily lost during descent due to environmental factors (e.g., occlusions or motion blur), the system reverts to a marker reacquisition process. This involves maintaining the current position and reactivating marker detection until the marker is found.
- The descent rate is controlled to ensure stability, with gradual adjustments made to roll, pitch, and yaw based on the marker’s relative position.

e) Controlled Final Descent

- When the drone reaches an altitude of approximately 50 cm, it transitions to a controlled descent mode.
- Roll, pitch, and yaw adjustments are minimized at this stage to enhance stability and avoid unnecessary oscillations.
- The descent speed is further reduced to prepare for a soft landing.

f) Touchdown and Motor Throttle Reduction

- As the drone approaches the ground, the descent speed is gradually reduced to ensure a smooth touchdown on the landing platform.
- The drone's motors are throttled down sequentially after the drone has securely landed and once the landing is confirmed, the drone will disarm the motors.

Implementation

The real-time implementation details how the methodology was executed, including technical configurations.

1. Camera Calibration

We captured images of a checkerboard pattern using the Raspberry Pi HQ Camera at different angles and distances. The checkerboard had a square size of 20 mm, and the pattern consisted of 10 inner rows and 7 inner columns. We captured around 30 images for greater calibration accuracy.

This configuration ensured sufficient feature points for accurate calibration. The captured images were processed using the OpenCV library, specifically the cv.findChessboardCorners() function, to extract corner points from the images. The object points (real-world 3D points) were matched with image points (2D points in the image plane), forming the basis for the calibration process.

Using these point correspondences, the cv.calibrateCamera() function computed the intrinsic parameters (focal length, optical center, and distortion coefficients) and extrinsic parameters (rotation and translation vectors). The intrinsic parameters are essential for correcting lens distortion and obtaining undistorted images, while the extrinsic parameters define the camera's position and orientation in space.

Results:

These calibration outputs were saved in .pkl files for later use in the marker detection and pose estimation stages.

Camera Matrix-

```
[[f_x,   0,   c_x],  [[2.49592842e+03 0.00000000e+00 9.55569553e+02]
 [ 0,   f_y,   c_y],   [0.00000000e+00 2.49538438e+03 7.36974582e+02]
 [ 0,    0,    1 ]]   [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

f_x and f_y: Focal lengths of the camera (in pixels) along the x and y axes.

c_x and c_y: Optical center (principal point) of the image in the x and y directions.

Distortion Coefficients-

These coefficients are applied during undistortion to correct the distorted image from the camera.

```
[[k1, k2, p1, p2, k3]]  [[-0.6721267   0.50548234  0.00735123  0.01202911 -0.25804789]]
```

k1, k2, k3: Radial distortion coefficients that model how straight lines curve outward or inward.

p1, p2: Tangential distortion coefficients that model the "tilt" of the lens.

Total reprojection error: 0.0916 is a measure of how accurately the calibration process has estimated the camera parameters (intrinsic and distortion coefficients).

Together, these coefficients describe how much the lens deviates from the ideal pinhole camera model.

2. Communication Setup

The communication setup between the Raspberry Pi and Pixhawk 6C was pivotal for the seamless operation of the drone, ensuring that pose data and landing commands were exchanged effectively.

Camera Configuration: The Raspberry Pi HQ Camera was configured using the Picamera2 library. We enabled high-resolution still mode (2028x1520) to ensure that the ArUco marker could be reliably detected even at higher altitudes. Picamera2 provided an efficient way to stream video frames, which were then processed in real-time for marker detection.

Communication Between Raspberry Pi and Pixhawk: To enable communication between the Raspberry Pi and the Pixhawk 6C flight controller, the DroneKit-Python library was used. This library, along with the MAVLink protocol, established a robust data link for bidirectional communication:

- **Pose data transfer:** The Raspberry Pi sent translation and rotation vectors (pose estimation) to the Pixhawk over MAVLink.
- **Flight commands:** The Pixhawk executed roll, pitch, yaw, and descent commands based on the received pose data.

DroneKit Integration: The DroneKit library allowed us to:

- Access vehicle state (e.g., altitude, GPS position, battery level).
- Send custom navigation commands based on marker pose.
- Transition between flight modes, such as from GUIDED to LAND.

The communication pipeline involved streaming real-time pose data from the Raspberry Pi to the Pixhawk at a frequency of 1 Hz. This setup ensured that the drone adjusted its trajectory dynamically during the landing sequence.

3. Detection of the Marker and Pose Estimation

Marker Generation: We generated the ArUco marker with ID 69 from the original Aruco dictionary and a marker size of 150 mm using an online marker generator. The marker was printed on A4 paper and placed on the landing platform. The specific size and ID of the marker were chosen to ensure reliable detection within the operating range of the camera which is at least 10m high.

Detection and Pose Estimation: The real-time video feed from the Pi HQ Camera was processed using the ArUco library. The detection process involved identifying the marker's unique ID and its corners in the image. Pose estimation, performed using *aruco.estimatePoseSingleMarkers(),* computed the translation and rotation vectors of the marker relative to the camera.

Pose estimation relies on solving the Perspective-n-Point (PnP) problem. The process takes as **input** the 2D pixel coordinates of the marker corners, the known 3D real-world coordinates of the marker corners, and the intrinsic camera parameters obtained during calibration. The **output** comprises a translation vector (*tvec*) that specifies the marker's position in the camera frame and a rotation vector (*rvec*) that encodes its orientation.

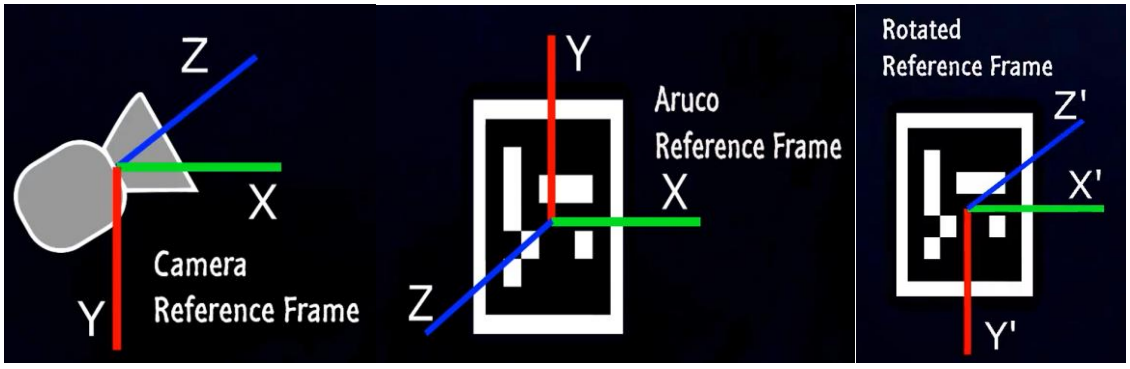

Fig: reference frames of camera and the marker

The camera reference frame is different from the marker frame as shown in the figure below. The camera frame is centered in the camera optical center with z axis going out with x axis going right and y going down. The Aruco marker instead has the z axis coming towards us and y axis going up with x axis remaining same. So they are one opposite to the other and we flip it.

The rotation vector is converted into a rotation matrix using Rodrigues' transformation. This matrix represents the marker's orientation in a more interpretable 3D form. Additionally, a flip matrix (R_flip) is applied to align the marker's reference frame with the drone's coordinate

system, ensuring consistency in interpreting spatial relationships. The adjusted rotation matrix is then transformed into Euler angles—roll, pitch, and yaw—for simpler interpretation of the marker's orientation.

Fig: pose estimated from the marker

```
Marker Attitude: roll=5.1° pitch=2.5° yaw=-12.5°
Marker Position: x=2.6 cm  y=5.0 cm  z=76.9 cm
Marker Attitude: roll=5.1° pitch=2.5° yaw=-12.5°
Marker Position: x=2.6 cm  y=5.0 cm  z=76.9 cm
Marker Attitude: roll=4.4° pitch=2.5° yaw=-12.5°
```

Fig: data calculated from the pose estimation

Results:

- **Marker detection:** Successfully identified the marker ID and its corners in real-time.
- **Pose estimation:** Provided the marker's position (x, y, z) and orientation (roll, pitch, yaw) relative to the camera.
- **Pitch**: Adjusts for forward or backward tilt of the marker. The drone pitches down or up to align.
- **Roll**: Corrects for sideways tilt. The drone rolls left or right to stabilize.
- **Yaw**: Aligns the drone's heading with the marker's rotation by rotating clockwise or counterclockwise.

4. Drone testing:

The drone is tested QGroundControl connected to Pixhawk 6c telemetry. We tested flying drone with loiter mode where the drone's position is auto stabled by using GPS and we can control the drones pitch and yaw manually. This mode is best suited when it's windy outdoors.

Fig: Interface of the QGC

We also set safety precautions in case of failure such as battery failure or general failure. In such case we made the drone return to launch place which is set as home before launching. We also made use of the geo fence feature to make the drone not go over the specified radius and altitude.

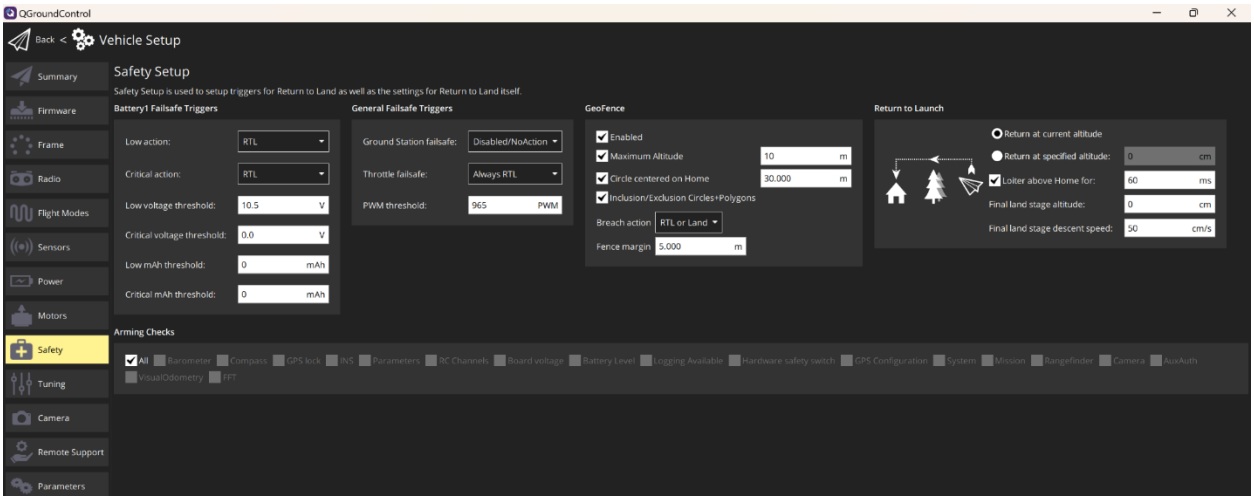

Fig: safety features enabled (RTL and Geofence)

Fig: drone testing outdoors

5. Landing Sequence

The landing sequence integrated the outputs of marker detection and pose estimation with the drone's control system to achieve precise landing.

a) **Trigger Return-to-Base (RTB)**

When the drone's battery level drops below a predefined critical threshold (e.g., 20%), the Pixhawk 6C flight controller initiates the Return-to-Base (RTB) operation. This is achieved using GPS data, where the Pixhawk navigates the drone to the home location coordinates. This initial phase is handled autonomously by the Pixhawk using its GPS and navigation algorithms.

Key Parameters:

Critical battery level: The battery monitoring system in Pixhawk continuously reports battery voltage and remaining capacity. This data is accessed through the MAVLink messages, and in DroneKit, it can be retrieved as part of the vehicle's telemetry. In DroneKit, the battery status is available in the *vehicle.battery* attribute.

Home GPS location: The home location (base station coordinates) is predefined during the initial drone setup or takeoff. The Pixhawk automatically records the GPS location as the "home location" when the drone arms or takes off. This can also be manually set using a specific MAVLink command. In DroneKit, the home location is stored in the *vehicle.home_location* attribute. This provides the latitude, longitude, and altitude of the home position.

Cruising Altitude: The Pixhawk maintains a safe cruising altitude during the Return-to-Base (RTB) phase to avoid obstacles. This altitude is determined by the flight mode (e.g., RTL or GUIDED) and the mission parameters set in the Pixhawk. The cruising altitude can be specified using the *RTL_ALT* parameter in Pixhawk. This parameter defines the altitude (in

centimeters) the drone maintains during the Return-to-Base operation. The cruising altitude is read and updated using *vehicle.parameters['RTL_ALT']*.

b) Marker Detection Activation

Upon reaching the vicinity of the home location, the Raspberry Pi activates the Pi HQ Camera to begin scanning for the ArUco marker. The real-time video feed is processed to detect the marker, and once detected, the pose estimation data is sent to the Pixhawk.

Implementation in Code: The marker detection process is initiated using the *ArucoSingleTracker* class from the provided code. This class uses the OpenCV ArUco library for marker detection and pose estimation. Below are the key operations:

- Marker Detection: The ArUco marker is detected using *aruco.detectMarkers()*.
- Pose Estimation: Once detected, *aruco.estimatePoseSingleMarkers()* computes the translation (*tvec*) and rotation (*rvec*) vectors of the marker relative to the camera.
- Marker Reacquisition: If the marker is temporarily lost, a search-and-realign process is triggered, where the drone adjusts its yaw to scan for the marker.

c) Communication Between Raspberry Pi and Pixhawk

The MAVLink protocol is the backbone of communication between the Raspberry Pi and Pixhawk. Below are the key components:

Data Sent to Pixhawk:

- Translation vectors (x, y, z) indicating the marker's relative position.
- Rotation vectors (rvec) used to adjust roll, pitch, and yaw.

Commands from Pixhawk:

- Adjustments to roll, pitch, and yaw to align with the marker.
- Descent control to maintain a steady trajectory.

DroneKit Integration: The DroneKit library facilitates the transmission of pose data and landing commands. It allows the Raspberry Pi to issue commands and monitor the drone's state during the landing sequence.

d) Alignment and Descent

Once the marker is detected, the drone adjusts its position and orientation to align with the marker. This alignment is achieved using pose feedback from the Raspberry Pi to the Pixhawk.

Pose Data Conversion: The pose data is converted from the camera's reference frame to the drone's reference frame using the following transformations:

- Camera to UAV frame: Adjusts the x and y axes based on the camera's orientation.
- UAV to NED (North-East-Down) frame: Uses the drone's yaw to convert pose data into global coordinates.

Communication: The Raspberry Pi sends pose data (translation and rotation vectors) to the Pixhawk via the MAVLink protocol. This data is used by the Pixhawk to update the drone's roll, pitch, and yaw dynamically during descent.

Key Parameters:

- Roll, pitch, yaw: Continuously adjusted based on pose feedback to maintain alignment with the marker.
- Descent rate: Gradual descent to ensure stability (controlled by the *land_speed_cms* parameter in the code).

e) Controlled Final Descent and Landing

At a predefined altitude (50 cm), the drone transitions to a controlled descent mode. This phase minimizes roll, pitch, and yaw adjustments to ensure stability and precision.

Landing Logic in Code:

- Descent Control: The descent rate is reduced to ensure a soft landing.
- Altitude Monitoring: The drone monitors its altitude and switches to LAND mode when the marker is within a safe range.

Throttle Reduction and Motor Shutdown: Once the drone touches down, the Pixhawk gradually throttles down the motors to ensure a stable touchdown.

Key Parameters:

- Altitude threshold: 50 cm (transition to controlled descent).
- Descent speed: 30 cm/s (ensures smooth touchdown).
- Landing mode: The Pixhawk switches to LAND mode to complete the operation.

Findings & Limitations

Marker Detection Reliability: Detecting the ArUco marker was affected by motion blur, low lighting, glare, and partial occlusions, especially during rapid drone movements or in outdoor environments. These factors reduced the visibility and contrast of the marker, hindering reliable detection. Adaptive thresholding helped improve contrast but was insufficient under extreme conditions.

Marker Reacquisition Challenges: Temporary loss of the marker due to occlusions or moving out of the camera’s field of view required the drone to perform a search-and-realign

process. While effective, this introduced delays in landing and was challenging in windy conditions or during fast descents.

Environmental Sensitivity: Changes in lighting, sudden weather variations, or strong winds impacted the marker's visibility and the camera's ability to detect it accurately. Adaptive filtering improved pose stability but could not fully mitigate detection inconsistencies caused by severe environmental disruptions.

Single-Marker Dependence: Reliance on a single marker posed a significant risk of losing critical pose data during occlusions or when the marker moved out of view. This created interruptions in pose estimation, particularly in dynamic scenarios, and limited system robustness.

Pose Estimation Noise: Rapid drone movements and environmental variability introduced noise in the pose estimation outputs, leading to minor trajectory deviations. Adaptive pose filtering smoothed the data but sometimes lagged behind real-time changes, limiting its effectiveness in highly dynamic environments.

Recommendations or Future improvements

Multiple Marker Implementation for Enhanced Landing: To improve reliability and robustness, incorporating multiple ArUco markers with different sizes on the landing platform can provide several advantages. This approach is inspired by the implementation demonstrated in the GitHub project RosettaDrone/vision-landing-2, which leverages multiple markers for precision landings.

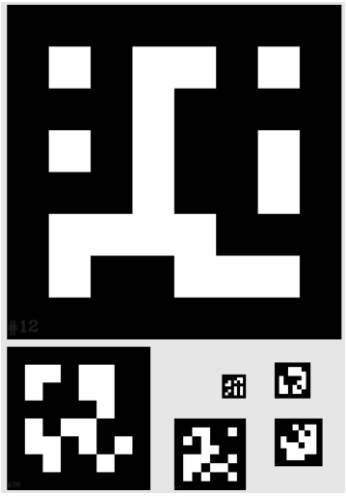

Fig: multiple Aruco markers arranged in a specific order

Here's how it could enhance our system:

1. **Redundancy and Reliability:**

- Multiple markers ensure that at least one marker always remains visible to the camera, reducing the risk of detection failure due to occlusion or lighting variability.
- Redundancy allows the system to switch seamlessly between markers if one becomes undetectable.

2. **Improved Pose Estimation:**
    - Combining pose data from multiple markers enhances the accuracy and stability of pose estimation by averaging inputs and reducing noise.
    - This approach can also help correct for errors introduced by camera distortion or environmental variability.
3. **Implementation Approach:**
    - The Raspberry Pi can detect multiple markers simultaneously using the OpenCV ArUco library, which supports multi-marker systems.
    - Pose estimation from all detected markers can be aggregated to compute a more accurate and stable final pose.
    - The MAVLink protocol can be adapted to handle additional pose data, ensuring real-time updates to the Pixhawk for trajectory corrections.

By transitioning to a multi-marker system, our project can address the limitations of single-marker dependence and achieve even greater reliability and precision, ensuring robust performance in challenging real-world scenarios.

Conclusion

This project successfully developed a precision landing system for drones using ArUco markers, addressing critical challenges in operations like battery swapping, delivery, and docking. By utilizing computer vision, camera calibration, and real-time pose estimation, the drone effectively detected and aligned with a predefined platform, ensuring accurate landings. Communication between the Raspberry Pi and Pixhawk 6C enabled dynamic trajectory adjustments during descent.

Despite its success, challenges such as sensitivity to environmental factors, motion blur, and single-marker reliance highlighted areas for improvement. Adaptive filtering and a search-and-realign mechanism mitigated some issues but did not fully resolve them.

Transitioning to a multi-marker system is recommended to improve redundancy, pose accuracy, and detection reliability, making the system more robust in varied conditions. Overall, this project demonstrates the potential of vision-based autonomous landing, laying the foundation for future advancements in efficient and reliable drone operations.

References

[1] R. Garrido-Jurado, R. Munoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marin-Jimenez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280-2292, 2014.

[2] E. Olson, "AprilTag: A robust and flexible visual fiducial system," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400-3407, 2011.

[3] S. Hrabar, "Vision-based 3D navigation and collision avoidance for UAVs," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1470-1475, 2006.

[4] X. Zhou, S. Song, and J. Huang, "A depth perception-based vision system for UAV landing," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1230-1242, 2018.

[5] M. Galeano, A. Garcia, and R. Lozano, "PID control for autonomous quadrotor: A robust approach," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1, pp. 69-82, 2011.

[6] Y. Chen, X. Xu, and L. Li, "Multi-sensor fusion for autonomous landing of UAVs," *Aerospace Science and Technology*, vol. 78, pp. 520-528, 2018.

[7] C. Chen, Z. Zhang, and Y. Zhu, "Model predictive control for quadrotor UAVs," *International Journal of Control, Automation and Systems*, vol. 16, no. 2, pp. 731-742, 2018.

[8] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme, "Vision-based autonomous landing of an unmanned aerial vehicle," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 3, pp. 371-380, 2003.

[9] R. Garrido-Jurado, R. Munoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marin-Jimenez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280-2292, 2014.

[10] X. Zhang, J. Wang, and P. Liu, "Vision-based UAV landing in GPS-denied environments," *Aerospace Science and Technology*, vol. 35, no. 8, pp. 1204-1215, 2018.

[11] E. Olson, "AprilTag: A robust and flexible visual fiducial system," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400-3407, 2011.

[12] K. Kocer, Y. Kaya, and B. Akinci, "Autonomous UAV landing on moving platforms using vision-based techniques," *Journal of Field Robotics*, vol. 37, no. 5, pp. 985-995, 2020.

[13] S. Hrabar, "Precision landing with IR Lock: A case study," *UAV Systems Journal*, vol. 22, no. 4, pp. 342-349, 2019.

[14] X. Zhou, S. Song, and J. Huang, "Ultrasonic-based altitude measurement for UAV precision landing," *Journal of Intelligent Systems*, vol. 29, no. 3, pp. 456-470, 2020.

[15] C. Chen, Z. Zhang, and Y. Zhu, "Lidar-based precision landing for quadrotor UAVs," *International Journal of Robotics Research*, vol. 36, no. 9, pp. 1123-1145, 2019.

[16] https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html

[17] https://chev.me/arucogen/

[18] https://learnopencv.com/rotation-matrix-to-euler-angles/

[19] https://www.mdpi.com/2504-446X/7/12/703

[20] https://github.com/tizianofiorenzani/how_do_drones_work/blob/master/scripts/06_precise_landing.py

[21] https://github.com/RosettaDrone/vision-landing-2

[22] https://github.com/niconielsen32/CameraCalibration/blob/main/calibration.py

[23] https://github.com/niconielsen32/ComputerVision/blob/master/ArUco/arucoDetection.py