ROB498 – Final Design Report

Mobile Landing Pad for MAVs

Seyed Eilia Dastmalchian, Kevin Hu, Sean O'Rourke, Leon Qian

22 April 2025

Design Overview

The quick delivery of life-saving supplies is critical in emergency response situations. An ambulance may not have certain supplies and cannot spare the time to stop or wait until it gets to a hospital. In this scenario, it would be incredibly beneficial to have a method of delivering supplies to an ambulance in motion. Our solution to this problem is an MAV capable of autonomously landing on a mobile landing pad. This design successfully detected and tracked down the landing pad, pathed towards it, and touched down without any manual control inputs.

Our team's philosophy was to independently develop and test each design component to achieve a minimum viable product. This minimized the number of potential points of failure, allowing us to streamline the testing and debugging process for each component. This compartmentalized structure permitted group members to work on different elements in parallel, speeding up the design process. Once an aspect was complete and thoroughly tested, we would combine it with the other components and test further to ensure seamless integration.

Background and Related Work

All of the code created for this project runs on the MAV's NVIDIA Jetson Nano, which acts as the main computer for the system. This code includes the landing pad detection, current pose estimation, and handling commands from the user for the MAV to execute.

The MAV's pose estimation and vision system are powered primarily by the Intel RealSense Tracking Camera T265. The T265 consists of two monochrome fisheye image sensors and an inertial measurement unit (IMU), containing an accelerometer and gyroscope [1]. The RealSense firmware provides on-board pose estimation using simultaneous localization and mapping (SLAM), which this design uses for its camera-based pose estimation.

The MAV's motor control and waypoint navigation is performed by the CubePilot Cube Orange Flight Controller. The Orange Cube comes equipped with the PX4 Autopilot flight control software, and has additional IMUs with redundancies to more accurately measure the MAV's movement [2]. Combining this IMU data with the pose estimation from the T265 through the use of a Kalman filter, the Orange Cube is able to estimate the MAV's position with incredible accuracy. By using the Orange Cube's PID controller, the MAV uses its current position and desired position to automatically travel along that path. The QGroundControl application acts as the system's flight control station, acting as an interface for a user to interact with the Orange Cube. Various MAV flight parameters can be configured from this control station, such as setting the MAV's extrinsic properties, tuning the maximum velocities and accelerations of the MAV for different use cases, and supplying the user with debugging software. QGroundControl helps with the scalability and adaptability of the design by allowing the MAV to be tuned by each potential client to suit their own needs.

In conjunction with the Realsense T265 camera, the MAV's vision system uses the OpenCV library to detect the landing pad and to determine its relative position. The landing pad is mounted with an ArUco marker for ease of detection. ArUco markers are a type of ARTag, which were developed by Garrido-Jurado et al. for camera pose tracking for robot localization applications [3]. We used these markers to assist in the objective of tracking the pose of the landing pad. The size and shape of the ArUco marker is known, so when the vision software detects the marker, the distance to the marker is calculated using the size and orientation of the marker in the image. The OpenCV library comes with built-in ArUco marker detection, allowing for quick development and seamless integration of the vision system with the rest of the MAV software. ArUco markers each have a unique ID associated with them, so if this design were to be scaled to multiple landing pads in the future, the MAV could be commanded to land on any specific landing pad in view based on the ArUco ID of the desired landing pad.

Design Objectives and Solution Details

The primary objective has not changed from the design proposal. The overarching objective is to develop an autonomous landing system for the MAV that can land on a low-speed moving platform indoors. The secondary objectives that support this are as follows.

Specific:

The MAV should be capable of identifying and locating the landing pad using the onboard Realsense T265 Tracking Camera while both the MAV and the pad are in motion. It then executes a controlled landing maneuver.

The changes from the design proposal is that the MAV does not employ a bottom-mounted LiDAR system. A Vicon system provides pose estimation for the MAV rather than a bottom-mounted LiDAR system or the Realsense camera. This change was made as the Vicon provided much better pose estimation than the Realsense, and the Realsense alone is sufficient for locating and landing on the pad.

Measurable:

There are several main components of the MAV that have quantifiable error. The first is the flight controller and the pose estimation. This error can be measured as the error in the positioning of the drone after it touches down. The second is the ArUco tracking. There are several factors to the ArUco tracking that are critical to a successful landing. The first element is the tracking error. The tracking error can be quantified by comparing the ArUco localization against a physical measurement. The second element of the ArUco system is, broadly speaking, the "responsiveness" of the system. This consists of the average update speed as well as the maximum interval between consecutive ArUco readings. Finally, the core objective is a measurable pass/fail: whether or not the MAV successfully touches down and remains within the bounds of the landing pad.

The landing accuracy can be measured after the drone touches down and settles, where it can be measured against the ArUco tag to determine the deviation from the programmed landing offset. The purpose of these measurements is that they reflect the total accuracy of the MAV, encapsulating the errors of the ArUco tracking, the pose estimation, and the flight controller.

The ArUco responsiveness can be measured in two ways. First, the maximum time between ArUco readings is the maximum time that the MAV is essentially flying blindly towards a last best-guess of the targeted pad. Secondly, the average time between ArUco readings shows how quickly the MAV can respond to unexpected changes in either its own pose estimate or any changes to the location of the landing pad.

In conclusion, there are three quantified objectives and one qualified objective.

1. The MAV must be able to land and remain on the platform.
2. The landing accuracy, physically measured against the ArUco tag on the platform.
3. ArUco maximum interval, pulled from data analysis of the recorded rosbags.
4. ArUco average polling rate, pulled from data analysis of the recorded rosbags.

Achievable:

Since it does not make much sense to talk about how achievable objectives were after the fact, this section will briefly touch on hypothetical future objectives and next steps, of which there are two primary next steps.

The first is to test a flight using the Realsense camera for both ArUco tracking/localization as well as the pose estimation for the MAV. The MAV has been shown to be able to fly accurately with the Realsense alone and the code was written to land at an ArUco marker with just the Realsense, but the team ran out of time to test this. Only the flight using the Vicon system was successfully tested under the principle of adding one component at a time.

The second step is to try to land the MAV on moving platforms of increasing speeds moving longitudinally, laterally, vertically, and rotationally. This step is of far greater scope and complexity. It is believed that the current system can handle slow speeds of longitudinal and lateral movement but more work would be needed to handle large vertical movements in the landing pad and faster movements in general. Significantly more work would be necessary to account for rotating landing pads.

Relevant:

This prototype demonstrates the feasibility of adapting this approach for integration with actual MAVs with a wide variety of use cases. Additionally, it also highlights some potential limitations in the specific tools and approaches used for this prototype.

Time-Bound:

The majority of the objectives were completed, although the design needed more flight testing flights and likely code debugging to be able to fully test all the objectives specified in the design proposal. The uncompleted sections mainly consisted of the aforementioned Vicon-less

flight and landing on moving platforms. The primary cause of the delays was the error in the Realsense ArUco tracking system at distance, which had multiple factors that were mutually exacerbated by each other. The factors included large errors inherent in the ArUco tracking at longer distances due to the fisheye camera, a logical error in our code where the ArUco localization was not properly updating, and a test suite that was not rigorous enough, missing the limit testing that could have identified either of the first two factors earlier in development.

Design Prototype Overview

The solution primarily consists of two parts. The first is the Realsense T265 that is responsible for identifying and locating the landing pad, marked by an ArUco tag. The second part is the on-board firmware that autonomously pilots the drone to the pad and lands on it.

Physical Prototype and Hardware Integration

Our capstone prototype is a fully functional Micro Aerial Vehicle (MAV) system designed to autonomously land on a moving platform marked with an ArUco fiducial tag. The aerial platform is a custom-built quadrotor equipped with a robust flight control stack and multiple onboard sensors. The drone's frame (a 250 mm-class quadrotor kit) was assembled according to the course guide, ensuring proper mounting of the power distribution board (PDB) on spacers to avoid shorting against the carbon fiber frame. Four brushless motors (with electronic speed controllers already pre-soldered to the PDB in our kit) provide lift, and the "wide side" of the frame is oriented as the forward direction to match the default chassis design. For flight control, we installed a CubePilot Cube Orange autopilot (Pixhawk family) running the open-source PX4 firmware. This Cube Orange flight controller contains the IMU, onboard controller loops, and interfaces for radios and actuators; PX4 was chosen because it is a widely adopted autopilot firmware actively developed by thousands of engineers worldwide. The Cube is powered via the PDB and connected to the motors and servos through its I/O ports (following the standard Pixhawk channel assignment for quadrotor X configuration). We also attached the required safety peripherals (GPS module for outdoors, though not used in Vicon lab tests, and an arming switch/buzzer as per PX4 standard setup).

A NVIDIA Jetson Nano (4 GB) serves as the companion computer mounted on the drone, providing onboard computing for vision processing and high-level decision making. The Jetson is powered from the drone's battery through a regulated power module and communicates with the Cube Orange via a serial UART (Telem2 port) using the MAVLink protocol. We installed ROS 2 on the Jetson along with the MAVROS package (the ROS–MAVLink bridge) to interface with the PX4 flight stack. The Jetson is also connected to an Intel RealSense T265 tracking camera (USB tether), which is a stereo fisheye camera with an IMU that provides visual-inertial odometry. In our design, the T265 is front-mounted and angled downward so its wide-angle lenses can observe the ArUco landing marker during the descent. The moving

landing platform itself is instrumented with a fiducial marker (ArUco tag) for visual detection. In indoor tests, the platform was tracked by the Vicon motion capture system as well, using reflective markers, to provide a ground-truth global pose. The Vicon system, comprised of multiple infrared cameras around the flight arena, delivers sub-millimetre position accuracy with very low latency [9], providing rapid and accurate pose estimation. This allowed us to obtain precise global position of the drone (and the platform, if needed) during development. Figure 1 shows an example of a ground vehicle outfitted with a flat landing pad and ArUco pattern (in our case, a simpler moving platform was used, but the concept is similar). All components – the Cube (PX4 autopilot), Jetson (ROS companion), RealSense camera, and Vicon – are integrated into a single system via ROS 2, enabling real-time data exchange and control commands throughout the autonomous landing mission.

Software Architecture and Autonomous Landing Workflow

The software system is structured around ROS 2 nodes that handle state estimation, target detection, and flight control, orchestrated by a high-level state machine for the landing sequence. The Cube Orange autopilot runs PX4's internal estimator and controller for stability (the low-level attitude and rate loops, receiving IMU data at ~1 kHz and outputting motor PWMs). The Jetson Nano runs the high-level control in ROS 2 user-space, which communicates with PX4 through MAVROS. We configured MAVROS to stream pose and sensor data from PX4 and to allow offboard control inputs (setpoint commands) to the drone. A *Vicon bridge node* publishes the drone's global pose (position and orientation) into ROS at ~100 Hz, which we use as the drone's state estimate in the world frame (essentially ground-truth state). This global pose can also be fed into PX4's EKF as a vision position estimate, aligning the autopilot's coordinate frame with the motion-capture frame for precise navigation [6]. Meanwhile, the RealSense T265 camera is utilized for relative localization to the landing marker. We run a visual processing node (built with OpenCV's ArUco detection library) that subscribes to the T265 fisheye images and detects the known ArUco tag placed on the moving platform. Because the T265 uses a wide-angle fisheye lens (Kannala-Brandt model), we perform image rectification before marker detection – the AprilTag/ArUco algorithms assume pinhole camera images [6]. We implemented a rectification pipeline (using the camera's calibration parameters) to undistort the fisheye images in real-time. This step was necessary for accurate pose estimation, although it incurs a slight reduction in effective frame rate compared to raw fisheye data [6].

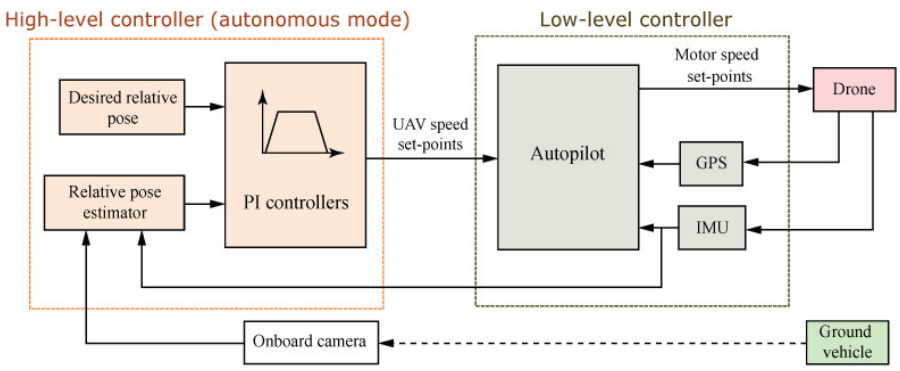

Figure 1: Data flow and control logic of the drone

The overall data flow and control logic are illustrated in Figure 2, which is a high-level diagram of our system's architecture. The autonomous landing sequence proceeds as follows:

Takeoff and Hover: The MAV is armed and takes off to a safe altitude (e.g. 2 metres) under manual or scripted control. Once airborne, it switches to an autonomous mode where the Jetson's high-level controller takes over, if not already in control. Using the Vicon-based state estimate, the drone can hold a steady hover or navigate to a predetermined area where the moving platform is expected to be. (In our indoor test, the platform moved within a known region, so the drone initially hovered above that region to search for the target.)

Target Detection and Tracking: The forward-facing RealSense camera continuously streams images to the Jetson. The ArUco detection node searches for the unique square fiducial pattern on the platform. When the ArUco tag comes into view, the node computes the relative pose of the tag with respect to the camera/drone frame (yielding relative x, y, z offset and yaw angle difference). The initial pose estimate of the tag is a vector from the ArUco to the Realsense camera in the frame of the camera, which is converted to a vector from the camera to the ArUco tag in the world (Vicon) frame. This can be added to the pose estimate of the camera in the world (Vicon) frame to compute the position of the ArUco tag in the world (Vicon) frame. At this point, the Jetson's state machine transitions to an approach phase. The relative pose estimator provides updates at roughly 8–12 Hz in our setup, which is somewhat limited by the image processing pipeline on the Nano. (Notably, on a more powerful companion like a Jetson Xavier NX, the same fiducial detection can run at ~14 Hz, and it is recommended to have at least ~10 Hz for smooth tracking [6].) The detected marker pose is fused with other data: the drone's global pose from Vicon and knowledge of the tag's size/orientation allow us to sanity-check the relative estimate. Once the tag is reliably detected, the system begins to actively track it.

Approach and Alignment: The high-level controller (running on the Jetson) now generates set-point commands to guide the drone toward the moving platform. We implemented a simple proportional controller that uses the ArUco marker's relative x–y offset to command lateral velocity adjustments, and the relative yaw to command heading changes, such that the drone yaw aligns with the platform. These desired motions are sent to the Cube Orange via MAVROS as Offboard commands (geometry messages converted to MAVLink setpoints). PX4's position controller treats these as target velocities/positions and produces the corresponding

motor outputs. During this phase, the Vicon system still provides a global reference to ensure stability; for example, the drone's altitude is primarily regulated by the altimetre (or Vicon Z data) to maintain a steady height above ground until ready to descend. As the drone closes in and the horizontal error reduces, the controller refines the alignment – effectively "locking onto" the platform. If the platform moves, the drone will follow, using continuous feedback from the marker. The onboard autonomy is governed by a state machine node, which monitors conditions (e.g., whether the horizontal offset is below a threshold and the marker is persisting in view) to decide when to initiate landing. We also incorporated safeguards: if the marker is lost (e.g., due to occlusion or the platform moving out of view), the drone will hover in place or ascend slightly and attempt to reacquire the target rather than blindly continuing.

Descent and Landing: Once the drone is directly above the platform (within a small error margin) and stable, the state machine commands a descent. In our implementation, this was done by switching the control mode to a downward velocity setpoint or using a PX4 LAND mode command when above the platform [8]. The drone begins to descend slowly towards the platform while the vision system continues to update relative pose. During final approach, the ArUco marker may go out of the camera's field of view (if the drone gets very close and the tag is near center, it actually disappears under the drone). At this stage, the assumption is that the drone is sufficiently well-aligned; the autopilot continues the descent at a fixed vertical speed. Touchdown is detected either by the autopilot (via its landing detection on contact) or by a predefined altitude threshold (since the platform's height is known). The motors are then disarmed after landing. Throughout the landing, the Cube Orange's internal stabilizers handle the fine attitude control – for example, if there is any wind or motion (such as due to propellor ground effect), PX4's IMU-driven controller will attempt to keep the drone level and execute the velocity commands being sent. The companion computer thus functions as the high-level navigator, and the flight controller as the low-level stabilizer, in a cascaded control structure [5]. In our tests, this division of responsibilities worked well: the PX4 autopilot's Offboard mode allowed us to send smooth position or velocity targets, and PX4 in turn handled the motor outputs to achieve those targets.

The roles of each component in the data flow can be summarized: the Vicon system provides precise global state of the drone (and can track the platform) which is crucial for initial localization and safety – essentially playing the role that GPS would in an outdoor scenario. The RealSense T265 (acting as an intelligent camera) provides the relative pose of the landing pad which is needed for fine alignment that global sensors alone cannot give. The Jetson Nano (ROS 2) runs the decision-making and vision processing, effectively bridging between the sensor inputs and the flight controller commands. The Jetson also runs any onboard logging and high-level logic. The Cube Orange (PX4) is responsible for all low-level flight dynamics: it takes the high-level commands and ensures the drone physically executes them within safe limits (using its IMU, barometer, and PID loops to maintain stability). MAVROS facilitates this integration by translating ROS messages to MAVLink MAV_CMDs and vice versa, enabling our custom Python/C++ ROS 2 nodes to control the drone without dealing with low-level serial

parsing. This modular architecture made it straightforward to implement and adjust each part (for instance, we could tweak the vision processing algorithm or tuning gains in ROS without modifying the flight controller firmware).

Performance in Demonstration

During the final demo, the prototype achieved autonomous landings on the moving platform with a high degree of accuracy and repeatability. The system detected the ArUco marker from approximately 2.5–3 m above the platform, and tracking was maintained as the platform moved at a moderate speed (on the order of 0.2–0.3 m/s in our tests). The overall landing accuracy observed was on the order of centimetres. In successful trials, the quadrotor touched down with a lateral offset of about 5–10 cm from the center of the ArUco target. This is comparable to other vision-based landing results reported in literature, where fiducial marker approaches often achieve roughly 0.1 m precision in real-world conditions [5]. Our system's accuracy was aided by the Vicon's precise positioning, but importantly, the relative positioning via the camera was also quite precise when the tag was in view at close ranges. The ArUco localization yielded minimal error in the horizontal plane and a reliable yaw alignment. We noted that the marker's orientation (yaw) was estimated very robustly (within a few degrees error) by the ArUco algorithm, which helped ensure the drone wasn't landing at an angle. However, the system did not heavily rely on estimating the platform's tilt (roll/pitch). The platform was approximately level, and any small tilt would be corrected by the drone's attitude controller. This matches findings by Morales *et al.* that ArUco-based pose is very accurate for position and yaw, though roll/pitch estimates can be noisy (not a significant issue for a flat landing pad) [5].

The ArUco tracking rate on the Jetson Nano averaged around 10 Hz. This update rate proved to be just sufficient for the controller to track the moving platform without instability. According to the PX4/ArduPilot precision landing documentation, a target perception update of at least ~10 Hz is recommended for smooth landings [6], and our results align with that guideline. At ~10 Hz, there is roughly a 100 ms latency between visual updates. In practice, this introduced a slight lag in the feedback loop. If the platform suddenly moved, the drone's correction would come at least a tenth of a second later. For the speeds and distances in our demo, this was manageable, but it hints at a limitation for higher-speed scenarios (discussed later). The drone's stability during approach and landing was generally good after careful tuning. Initially, in early test runs, we observed a tendency for the drone to "wobble" or oscillate/circle when trying to aggressively correct its position over the target. This was especially evident when we left the default PX4 position control gains unchanged – the combination of high gain and delayed vision updates caused overshoot. Through iterative tuning, we reduced the aggressiveness of the approach: we limited the maximum horizontal speed and lowered the P-gains in the offboard position controller. The result was a much smoother trajectory – the drone would gently align over the platform rather than jerking back and forth. By the time of the demo, the vehicle could descend onto the platform with only minor adjustments and no visible

oscillation. We also tuned the descent speed to be fairly slow (~0.2 m/s) to give the vision system time to adjust laterally during the downward motion.

Another performance metric of interest is the landing repeatability and success rate. Out of several attempts, the system consistently landed on the platform when the marker was detected and lighting conditions were stable. In only a couple of runs did we have to abort – once due to a false detection (the camera picked up a spurious pattern and the drone started to drift off course) and once due to a network latency hiccup in Vicon data. In ideal conditions, the state machine correctly guided the UAV to a safe landing. The final approach looked stable from an external perspective: the drone would hover, then smoothly translate above the moving target, then descend almost vertically. The landing impact was gentle and within the tolerance of the platform (which was a flat surface on a wheeled base). We measured the final resting offset manually and confirmed it was within 10 cm. The heading alignment was nearly perfect (the drone landed roughly facing the same direction each time, as intended). It's worth noting that Vicon provided an external measure of performance. We logged the drone and platform positions to calculate the landing error more precisely. These logs showed the drone's trajectory converging with the platform's trajectory as expected. Furthermore, despite the relatively low image frame rate, the control loop handled the moving target well: there was no scenario where the drone "chased" the platform in an unstable way. We attribute this to the conservative tuning and the inherently slow dynamics of the platform. Overall, the demonstration showed strong promise that the prototype could fulfill the primary objective: autonomously land the MAV on a moving tagged platform in real time, especially given reasonable speeds for the movement of the platform.

Limitations and Challenges

While the prototype mostly met its functional goals, we encountered several real-world limitations and challenges that constrained performance:

- **Fisheye Camera Distortion and Noise:** The use of the RealSense T265's fisheye cameras required special handling that introduced some noise into the system. Because standard ROS image pipelines did not support the T265's fisheye model out-of-the-box, we implemented a custom rectification node to undistort the camera images for ArUco processing [6]. This extra step (and the ApproximateTime synchronization of stereo frames we performed) came at the cost of a slightly reduced image frame rate [6]. More importantly, any imperfections in the calibration/rectification can lead to errors in the detected marker corners. We noticed that when the marker was near the edge of the camera's field of view, the pose estimate would jitter by a few centimetres, likely due to lens distortion effects that were not fully corrected. This manifests as sensor noise in the relative pose. Additionally, we found that beyond certain ranges (5-7m, depending on lighting and angles), the pose estimate of the ArUco tended to have a high degree of

error, especially if the ArUco was not in the direct center of the FOV. In an ideal scenario, one would use a camera with a well-modeled lens or avoid rectification by using a fiducial algorithm that supports fisheye directly. In our case, we managed with the rectified images but it is important to acknowledge that the added processing and potential calibration error introduced noise that slightly reduced the precision of the relative localization. The issue of high amounts of error at range is largely fixed by constantly updating the ArUco pose estimate as the drone approaches, as the error drastically decreases within the specified range, so even an extremely noisy ArUco pose estimation at range is not an issue.

- **Limited Update Rate of Vision Measurements:** As mentioned, the effective update rate of the vision pose (ArUco detection) was around 10 Hz on our hardware. This is a relatively low frequency compared to the drone's flight control loops (the PX4 attitude control runs at 250 Hz, and even its position loop can run 50 Hz or higher). The low update rate means the system has inherent latency. The drone could move several centimetres in the time between camera frames, especially if it's responding to wind or its own control commands. This limitation forced us to keep the drone's motions slow and deliberate. If the controller had attempted aggressive maneuvers, the 100 ms delay could lead to overshooting the target before the next correction comes in. In essence, the low vision update rate reduces the effective bandwidth of the outer control loop. We partly mitigated this by relying on the IMU (via PX4) to stabilize fast disturbances and by tuning the system assuming ~0.1 s delay. Nonetheless, this was a bottleneck; we even explored configuration options like increasing the T265 exposure (to potentially raise FPS) and the number of threads in the ArUco detector. According to ArduPilot's documentation, a landing target update below ~10 Hz starts to noticeably degrade landing performance [6]. Our experience was consistent with this, at one point when the detection was only ~5–6 Hz (before optimization), the drone could not smoothly follow the platform, whereas improving it to ~10 Hz made the landing possible. This underscores that the camera/processing speed was a limiting factor. A higher-power companion computer or more efficient vision algorithm would be needed to push beyond the current target speeds.

- **Controller Aggressiveness and Tuning Trade-offs:** Achieving a smooth landing required extensive tuning of the control parameters to account for the above limitations. Early in development, we had the drone use relatively aggressive setpoints (for example, quickly nullifying position error) which led to a lot of shaking and overshoot. The overshoot was evident as the drone would move past the platform and then have to correct back, or oscillate around the target. This behavior is a classic symptom of an under-damped control loop, exacerbated by delay and noise in the feedback signal. We realized that we had to "tune down" the drone's aggressiveness – effectively trading off

speed for stability. We reduced the proportional gains in the offboard controller and added a small damping term; we also capped the maximum velocity so that the drone approached more slowly. These changes eliminated the oscillations but made the approach take longer and be more sensitive to platform motion (since the platform could move out from under the drone if we went too slow). It was a balancing act. In the final system, the tuning was on the cautious side, prioritizing a stable landing over a fast one. This approach of tuning for stability given the system's limits aligns with guidelines from similar projects – for instance, an ArduPilot-based study emphasizes finding the "limits of the system" (detection range, data rate) and tuning within those limits for desired results [6]. We indeed determined that our vision data rate and noise levels would not permit very fast response, so we constrained the controller accordingly. The downside is that this would not scale well if the platform moved faster or if we needed to land more quickly (we discuss potential improvements for that in the next section).

- **Reliance on Motion Capture (Indoor Localization):** Another limitation of our prototype is its dependency on the Vicon motion capture system for precise localization. While Vicon gave us excellent performance in the lab, it is obviously not a feasible solution in most real-world scenarios. We treated Vicon as a stand-in for a perfect GPS or an ideal state estimator. In reality, outdoor GPS is far less precise (typical error ~±1–2 m, or ~±1 cm with RTK corrections) and has much lower update rates than Vicon [6, 9]. Moreover, our platform's global position in the tests was known; in the real world, the drone would have to find the moving platform without an external system feeding its exact coordinates. This limitation means our current system was not fully self-contained. It leveraged an external tracking system. Any real deployment would need to incorporate onboard global localization (or cooperative localization) to replace Vicon. During our demo, a subtle effect of Vicon was that it provided very clean state feedback; if we turned Vicon off and tried to use only the T265's internal odometry, the drone's navigation became noticeably worse due to drift. So, while not a flaw per se, the reliance on motion capture highlights that additional development is required for autonomy beyond the lab.

- **Sensor and Environment Factors:** We also faced some minor challenges with sensors and the environment. The T265's internal IMU and VIO sometimes conflicted with the external Vicon data – careful frame alignment and timing synchronization was needed to avoid jumps in the pose estimate. We resolved this by resetting the T265 odometry on takeoff and using Vicon as the primary pose source. Lighting conditions in the lab were generally good, but if reflective surfaces appeared (e.g., a glossy floor panel), the camera could get false corners. The ArUco detection is quite robust to lighting compared to, say, pure color tracking, but it can fail if the tag is motion-blurred or at extreme angles. We noticed that if the drone rolled more than ~30° while the tag was in view (which seldom happened since PX4 kept the drone relatively level), detection confidence dropped. This

suggests the need to keep the marker roughly face-on to the camera. These practical quirks introduced extra setup steps and are noted as limitations that we would address with more robust calibration procedures in a polished system.

In summary, the main limitations experienced were rooted in sensor limitations (distortion, update rate) and control tuning constraints. We managed these issues through conservative design choices, but they illustrate the gaps between a working prototype in the lab and a hardened system ready for all conditions.

Scaling to Real-World Outdoor and High-Speed Operation

Adapting this autonomous landing system for outdoor or high-speed real-world use would require several enhancements and careful engineering beyond the current prototype. The key areas to address are global localization, robustness of vision in challenging environments, control strategy for higher dynamics, and system resilience.

Replacing Vicon with Real-World Localization:

In an outdoor scenario, we cannot rely on motion capture for precise positioning. A common substitute is to use GPS (GNSS), potentially augmented with Real-Time Kinematic (RTK) corrections to improve accuracy. Modern RTK GPS systems can achieve on the order of ~1–3 cm horizontal accuracy under good conditions [6], which begins to approach the precision needed for landing on a small platform. For instance, Morales *et al.* implemented their outdoor landing system with a Here+ RTK GNSS on the drone, allowing the autopilot to hold a steady position in guided mode [5]. In our case, we would integrate an RTK GPS to provide a global pose estimate of the UAV. However, even centimetre-level GPS may not be sufficient alone to guarantee a bullseye landing – the final metres of the descent still benefit enormously from vision. Therefore, the likely approach is a fusion of global and relative localization: use GPS (or an onboard SLAM system) to get the UAV near the moving platform, then use the fiducial marker (or another precision sensor) for the final approach. This is analogous to how our indoor system used Vicon to get close and vision to finalize the landing. We might also consider using the Intel T265's visual-inertial odometry as a supplement to GPS. The T265 can provide odometry at 200 Hz [6], which could be fused in the autopilot EKF to smooth out GPS noise and provide high-rate state estimates (as some projects have done for non-GPS navigation [6]). Another consideration is that in a truly self-contained system, the drone would need to find the moving platform on its own. This could involve the platform broadcasting its GPS coordinates via a radio link, or the drone conducting a search pattern with its camera if the approximate area is unknown. For example, if the platform is a UGV, it could communicate its position to the UAV (vehicle-to-vehicle communication), effectively giving an initial guess for the UAV to point its camera and pick up the ArUco tag. In summary, moving to outdoor use means integrating a

reliable global pose source (RTK GPS or onboard SLAM) and possibly an inter-vehicle communication system, to replace the indoor motion capture.

Robust Vision and Sensing for Outdoors:

Outdoor environments introduce challenges like variable lighting, weather, and larger spaces. Our ArUco marker approach would need to be robust against sunlight glare, shadows, and longer distances. One improvement would be to use a higher-resolution or global-shutter camera for detection, ensuring the marker can be seen from higher altitude or further away. The RealSense D435 or a similar RGB camera could be added to capture a clear downward view (the D435 was used by Castelo *et al.* for their marker detection [5]). ArUco markers themselves can work outdoors, but ensuring detectability might involve using a larger marker or an array of markers (a pattern) so that at least one marker is visible from a range of angles [5]. In fact, an array of multiple fiducials arranged on the platform could improve detection reliability; our prototype used a single tag for simplicity, but a multi-marker pattern (like a cluster of ArUco tags forming a composite marker) would allow detection at various orientations and provide redundancy [5]. Another approach is to switch to AprilTag markers, which are designed for robustness and have integrated error-correction; AprilTags are widely used in outdoor robotics due to their strong detection performance under difficult lighting and their flexible tag family (AprilTag 3 offers different sizes and even circular layouts [6]). The trade-off is computational – AprilTags might be slightly heavier to detect, but on a stronger companion computer that should be manageable. Speaking of computation, for outdoor/high-speed usage, we would likely upgrade the companion computer from a Jetson Nano to something like a Jetson Xavier NX or Orin. This would provide the necessary horsepower to run vision algorithms at higher frame rates or even incorporate more advanced perception (e.g., running a neural network to detect the platform or using optical flow in parallel). High-speed landings might also benefit from other sensors: for instance, a downward-facing LIDAR or radar could directly measure range to the ground/platform, complementing the visual data for altitude hold during landing (as seen in other works that added a LiDAR for more reliable altitude data in landing sequences [5]). Additionally, environmental robustness entails weather-proofing. The cameras and electronics would need protection from rain or dust, and one might need to consider the effect of wind on the drone. Wind gusts could cause sudden drift; to mitigate this, a tighter integration of the IMU data in the state estimate (e.g., using the autopilot's wind estimation or airflow sensors) might be required. In essence, scaling to outdoors means hardening the perception: better cameras or markers for long-range detection, possibly multiple sensor modalities (vision + depth or RF beacons), and computing resources to handle these in real time.

Enhanced Control for High Dynamics:

If the moving platform travels faster (say a car or a UAV carrier moving at several m/s) or if we want to land more quickly, our control approach must be more advanced. A simple

P-controller that was sufficient at low speeds might not catch a fast-moving target. Research has explored more sophisticated controllers and planners for this problem. For example, Wang *et al.* (2022) designed a gradient-based motion planner for landing on a moving platform, which can rapidly generate collision-free trajectories and even anticipate the platform's motion [7]. Such a planner, coupled with a state machine, allowed their quadrotor to track and land on a moving platform autonomously at higher speeds [7]. We could incorporate a similar idea: use the predicted trajectory of the platform (extrapolated from its current velocity) and plan an intercept path for the drone, rather than just reactive feedback. This would reduce the chance of falling behind a fast target. Furthermore, a cascaded control loop with feed-forward terms could improve response. In the literature, a common approach is to use a velocity setpoint generator with a trapezoidal profile (for smooth acceleration/deceleration) feeding into decoupled PID controllers for each axis [5]. Our system currently uses a basic decoupled P controller; extending it to include feed-forward (based on target velocity) and integral terms could help handle constant biases (like wind or systematic offset). Additionally, at high speeds, the drone might need to bank and tilt significantly while tracking – this could complicate marker detection (the camera might not always face the tag). One way to handle this is to mount the camera on a gimbal or use a wider field-of-view lens so that even during aggressive maneuvers the tag stays within view. There has also been interesting work using reinforcement learning controllers trained in simulation to handle the landing task under dynamics that are hard to model. Rodriguez-Ramos *et al*. used an OptiTrack system to train a drone with deep reinforcement learning for landing on a moving target [12], demonstrating the feasibility of learned control policies. While implementing an RL policy on our platform is non-trivial, it's an avenue for achieving very fast reactions (since a neural policy could infer control outputs directly from sensor observations in a few milliseconds). In any case, for high-speed interception, we might need to offload some computation or use higher-frequency processes – for example, running part of the control loop in the flight controller (which is C++ and real time) rather than in ROS Python. PX4 does support a feature called Precision Landing with IRLock (infrared beacon system) which uses a custom sensor and internal control; a similar concept could be applied where the autopilot is given more autonomy in final stages. In summary, reaching higher-speed capability calls for better prediction and planning in the controller, potentially leveraging advanced algorithms or even learning-based methods, and ensuring the physical drone can handle aggressive maneuvers (strong motors, properly tuned rate controllers, etc.).

System Resilience and Safety:

A real-world deployment demands higher reliability. This means adding layers of fault tolerance that we might have overlooked in a prototype. For instance, if the vision fails (marker not detected) at a critical moment, the UAV should have a safe fallback – perhaps enter a loiter pattern rather than wandering off. If the moving platform suddenly stops or changes velocity beyond expected bounds, the drone's controller should adapt or abort. In outdoor scenarios, one must also consider lost GPS or lost communication. Our system could be extended with a

behavior tree or more complex state machine that handles these contingencies (return-to-home or land safely if something goes wrong). Scalability also implies thinking of different platform types: what if the landing target is a moving vehicle like a truck bed or a ship deck? The vision system might need different markers (a large painted target or flashing LEDs at night). There are examples of drones using LED beacons for landing on moving platforms (for instance, IRLock uses an IR LED and sensor for precision landing in GPS-denied situations). Such alternatives could be integrated to assist when the camera marker is not reliable (e.g., in darkness, an IR beacon could guide the drone when cameras fail). Additionally, to truly operate at high speeds or outdoors, we'd ensure the communication link (telemetry) is robust, possibly employing a direct microcontroller-to-microcontroller link for low-latency control updates (instead of routing everything through ROS on Linux, which can have unpredictable timing). Fortunately, ROS 2's middleware can be configured for real-time performance (using DDS QoS settings), but extra testing would be needed.

To ground these considerations in existing work: Chang *et al.* (2018) demonstrated an autonomous landing using sensor fusion of IMU, stereo vision, ArUco markers, and even a YOLO object detector for the landing pad [10]. Their approach highlights that multi-sensor fusion can yield a more robust system, at the cost of complexity. Gautam *et al.* (2020) used a combination of color segmentation (to detect a colored pad) and AprilTags to handle detection from high altitude to low altitude [11], which shows the value of blending methods for different stages of approach. These studies, along with our findings, suggest that scaling up will likely involve hybrid solutions: integrating the best of global navigation (GPS/odometry) with precision local sensors (vision, LiDAR), and switching control strategies as appropriate during the mission (e.g., a high-speed pursuit phase vs. a fine landing phase).

In conclusion, to transform our prototype into a real-world, high-speed autonomous landing system, we would need to implement precise global localization (such as RTK GPS or onboard SLAM), more robust and faster vision processing (potentially with improved hardware and algorithms), and advanced control logic to handle faster-moving targets. The current design provides a solid foundation – a working integration of fiducial vision with an autopilot – but significant engineering and testing would be required to ensure it works outside the lab. With these enhancements, an MAV landing on a moving platform can move from a controlled demo to an operational capability, as evidenced by recent research prototypes that have achieved outdoor autonomous landings using similar building blocks [5, 7]. The path to real-world deployment will involve not only scaling the technology but also rigorous validation under diverse conditions to ensure the system's reliability and safety when GPS signals are imperfect, when the platform is moving erratically, or when the wind pushes the drone – scenarios that go beyond the calm indoor environment for which our current design was tailored. By addressing these challenges with improved sensors, algorithms, and fail-safes, the autonomous landing system can be scaled up to meet real-world demands.

Results and Evaluation

During the live demo, we showcased the working aspects of our design. The demo involved launching the drone vertically and tracking its position with the Vicon vision system. The Realsense camera tracked the ArUco marker, obtaining the vector from the drone to the ArUco in the Realsense camera frame. This vector was transformed into the Vicon vision frame and added to the drone's position to obtain the position of the landing zone in the Vicon frame. The drone flew to a designated hover zone, which was a constant offset away from the marker to avoid landing and crashing into it. Once the drone hovered sufficiently, it attempted to drop its altitude to land on the landing zone. Figure 2 shows the position of the drone on the landing zone after the demo, and Figure 3 shows the trajectory of the drone during the landing demo.

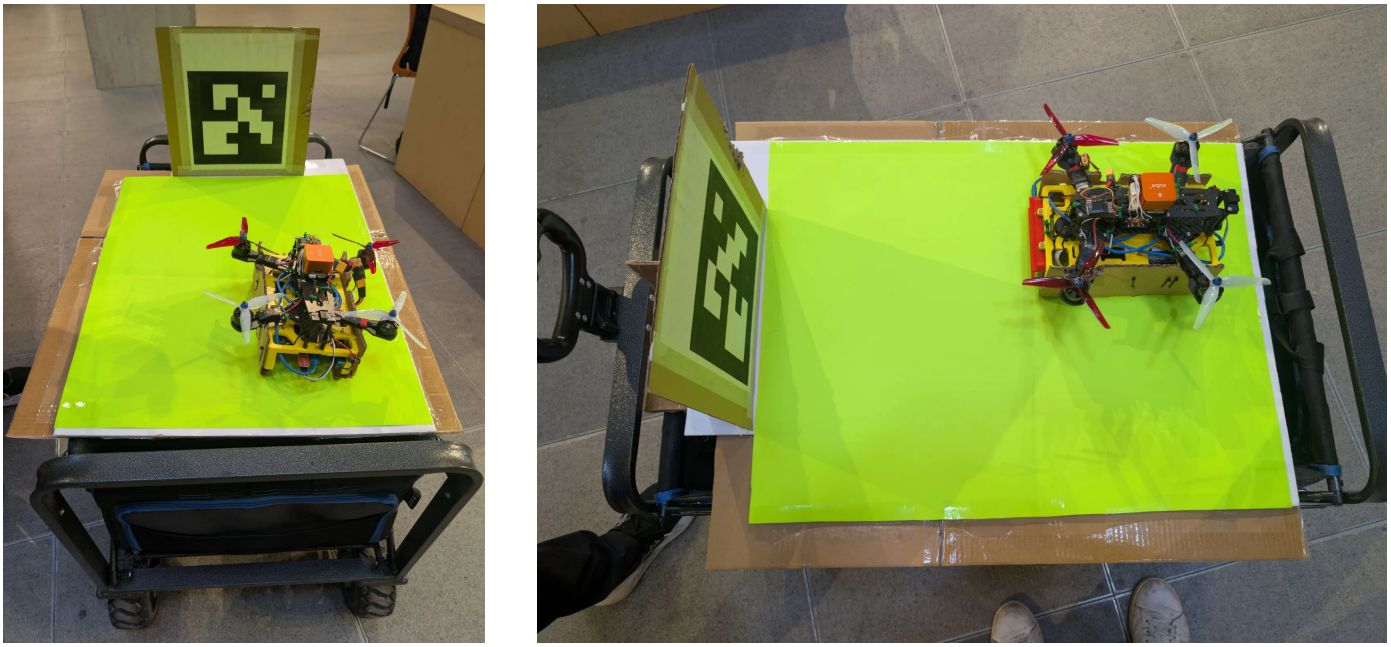

Figure 2: Landing position of the zone after the live demo

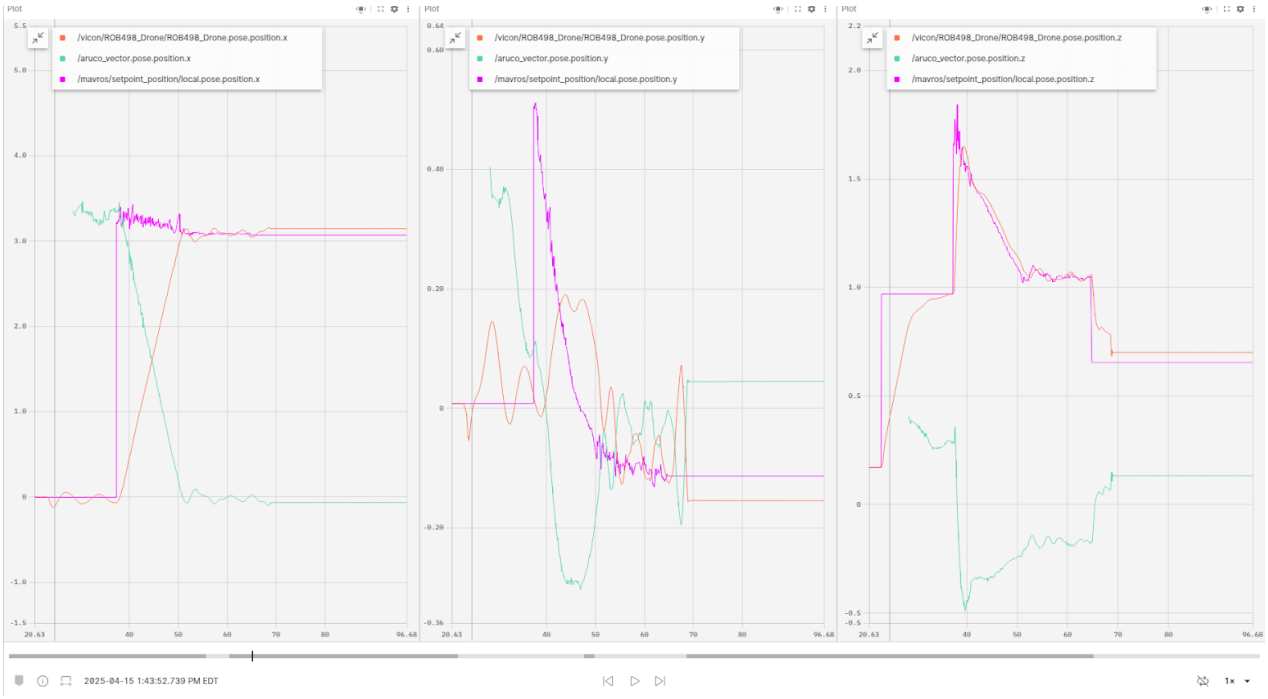

Figure 3: Flight trajectory of the drone. The leftmost graph shows position along the x direction, the middle graph shows position along the y direction, and the rightmost graph shows position along the z direction (altitude). The position of the drone (in Vicon frame) is given in orange, the position of the ArUco marker (in Realsense frame) is given in green, and the position of the landing zone (in Vicon frame) is given in purple.

Our primary set of design criteria had to do with the successful landing of the drone:

1. The drone must be entirely on the pad after disarming.
2. Parts must not be broken.
3. There must be no contact other than landing skids to the landing pad.

Our secondary criteria involved a more quantitative analysis of the performance:

1. The landing should be accurate.
2. The ArUco should be consistently detected throughout the entire flight.
3. The Realsense should report distances with accuracy.

Our metrics were designed to evaluate the design based on those criteria. Quantitative design requirements for position errors were given as a percentage of the length of the landing pad (71.10 cm). Table 1 shows the evaluation of the flight demo under our metrics:

Metric	**Measured Value**	**Design Requirement**

Primary Metric (Live Demo)		
Successful landing as per primary design criteria	PASS	PASS
ArUco Detection Accuracy (Offline, Stationary)		
Minimum distance from ArUco for detection	28.53%	< 30%
Euclidean error of detected ArUco vector	9.64%	< 10%
Landing Accuracy (Live Demo)		
Longitudinal error of drone from landing point	8.56%	< 10%
Lateral error of drone from landing point	11.39%	< 10%
ArUco Detection Reliability (Live Demo)		
Longest time span between ArUco detections	1.102 seconds	2 seconds
Average frequency of ArUco detections	11.878 Hz	15 Hz

Table 1: Evaluation of live demo under our metrics.

Lessons Learned

The primary lesson learned was the necessity of robust test suites that can effectively test limits and edge cases of subsystems in complete isolation. Crucially, our test suites were missing the capabilities of generating Vicon data that could be replayed or hard-coded into tests. This excluded some parts of our systems from being testable offline, out of the lab space, limiting what could be tested without having to wait in a queue to fly. Part of this lesson is also that a robust simulation environment should be set up, such as a Gazebo testing environment. We expected the bulk of the debugging to be around the vision system rather than basic flight issues, which may have been true near the end of the course. However, we clearly underestimated the difficulties and technical challenges associated with basic flight and probably would have benefitted from being able to run tests in an environment like Gazebo, despite the large initial hurdle of setting it up.

The main collaboration issue was the team falling behind in crunch times. We did not effectively plan our time around midterms and thesis presentations. This, in conjunction with frame transformation issues for Flight Exercise 2, meant that the team fell behind on both Flight Exercise 2 and 3, leaving us scrambling approaching the final deliverables. The thing we would do most differently is to aim to finish the flight exercises early rather than on-time, especially given the relative surplus of time early in the semester. We found the recurring theme was that technical issues were inevitable but could always be solved, given just a little bit more time.

References

[1] “Intel RealSense Tracking Camera T265 Datasheet Revision 004,” Sep. 2019. https://www.intelrealsense.com/wp-content/uploads/2019/09/Intel_RealSense_Tracking_Camera_Datasheet_Rev004_release.pdf (accessed Apr. 20, 2025).

[2] “CubePilot Cube Orange Flight Controller | PX4 User Guide.” https://docs.px4.io/main/en/flight_controller/cubepilot_cube_orange.html (accessed Apr. 21, 2025).

[3] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognition*, vol. 47, no. 6, Jun. 2014, doi: https://doi.org/10.1016/j.patcog.2014.01.005. (accessed Apr. 20, 2025)

[4] X. Wang and J. Kelly, “Setting Up Your Drone to Fly *Safely*,” Capstone Course Manual, Univ. of Toronto, Toronto, ON, Canada, 2025. [Online]. Available: https://github.com/utiasSTARS/ROB498-support (accessed Apr. 20, 2025)

[5] J. Morales, I. Castelo, R. Serra, Á. Hervella, and P. González-De-Santos, “Vision-Based Autonomous Following of a Moving Platform and Landing for an Unmanned Aerial Vehicle,” *Sensors*, vol. 23, no. 2, Art. no. 829, Jan. 2023. [Online]. Available: https://doi.org/10.3390/s23020829 (accessed Apr. 20, 2025)

[6] ArduPilot Development Team, “Precision Landing with ROS, RealSense T265 Camera and AprilTag 3 (Part 2/2),” *ArduPilot Blog*, 11-May-2020. [Online]. Available: https://discuss.ardupilot.org/t/precision-landing-with-ros-realsense-t265-camera-and-apriltag-part-2-2/51493 (accessed Apr. 20, 2025)

[7] P. Wang, J. Man, B. Li, X. Zhang, and S. Shen, “Quadrotor Autonomous Landing on Moving Platform,” *arXiv* preprint arXiv:2208.05201, Aug. 2022. [Online]. Available: https://arxiv.org/abs/2208.05201 (accessed Apr. 20, 2025)

[8] PX4 Developer Team, “Companion Computers,” *PX4 Developer Guide*, 2025. [Online]. Available: https://docs.px4.io/main/en/companion_computer/ (accessed Apr. 20, 2025)

[9] “Motion Capture Systems for Drone Tracking and uavs from Vicon,” Vicon, https://www.vicon.com/applications/engineering/autonomous-vehicles/#:~:text=Image (accessed Apr. 22, 2025).

[10] C.-W. Chang et al., “Proactive guidance for accurate UAV landing on a dynamic platform: A visual–inertial approach,” MDPI, https://www.mdpi.com/1424-8220/22/1/404 (accessed Apr. 22, 2025).

[11] A. Gautam, M. Singh, P. B. Sujit, and S. Saripalli, “Autonomous Quadcopter Landing on a moving target,” MDPI, https://www.mdpi.com/1424-8220/22/3/1116 (accessed Apr. 22, 2025).

[12] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, P. de la Puente, and P. Campoy, “A deep reinforcement learning strategy for UAV Autonomous Landing on a moving platform - Journal of Intelligent & Robotic Systems,” SpringerLink, https://link.springer.com/article/10.1007/s10846-018-0891-8 (accessed Apr. 22, 2025).