Proceedings of the ASME 2025 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2025
August 17-August 20, 2025, Anaheim, California, USA

IDETC2025-xxxx

DRAFT: PRECISION AUTO-LANDING OF AN AERIAL VEHICLE ON A MOVING GROUND VEHICLE: A MODULAR ROS2 APPROACH

Luca Russo*, Md Safwan Mondal, Subramanian Ramasamy, Pranav A. Bhounsule
Department of Mechanical and Industrial Engineering
University of Illinois at Chicago
842 W. Taylor St., Chicago, IL, 60607
lrusso5@uic.edu, mmonda4@uic.edu, sramas21@uic.edu, pranav@uic.edu

ABSTRACT

Unmanned aerial vehicles (UAVs) have emerged as cost-effective and versatile robotic systems for a wide range of applications, including search and rescue, environmental monitoring, and surveillance and reconnaissance. However, their limited endurance—often less than 30 minutes—restricts their potential for long-term, continuous operations. To overcome this limitation, UAVs can be controlled to land on moving Unmanned Ground Vehicles (UGVs) for recharging, effectively extending their operational duration based on the UGV's ability to carry portable charging pads. While UAV landing on moving UGVs has been explored in indoor environments equipped with motion capture (MOCAP) systems, this problem has not been thoroughly addressed in outdoor scenarios where Geographical Positioning System (GPS)-based localization offers significantly lower positional accuracy compared to MOCAP systems. Our approach addresses this challenge by employing GPS to track the UGV's position during the initial phase. Once a special tag called the ArUco marker mounted on the UGV comes into view of the UAV's downward-facing camera, the system switches to a vision-based tracking mode. The ArUco marker position data is processed with a least square estimate to obtain a smooth, reliable estimate of the UGV velocity. Then, combined feedforward and feedback control strategies ensure the precision landing of the UAV on the moving UGV. Our control algorithm has demonstrated reliable landings across a range of UGV speeds. Additionally, we demonstrate that using a ROS2-based modular framework enables standardization and scalability, allowing the system to be adapted to multiple UAV-UGV platforms in future deployments.

1 Introduction

Recent improvements in computational capabilities, sensors, and mechanical design have driven automation in various fields. Robotics solutions, including the coordination of several heterogeneous robots, have been implemented to tackle increasingly challenging tasks. One significant problem in this context is the routing between an aerial vehicle (UAV) and a ground vehicle (UGV). In tasks such as surveillance, assessment, maintenance, or rescue operations, the agility and aerial perspective of a drone can help identify potential fire hazards [1] or areas affected by disasters [2] [3], allowing for fast and precise support. However, drones alone are often insufficient; their small batteries limit their overall mission time. In contrast, a ground vehicle, especially one with wheels, has movement constraints since it requires suitable terrain to navigate. Nonetheless, it has a larger storage capacity for batteries, which can be used to recharge drones and extend their operational life [4].

Throughout the course of the mission, the aerial vehicle will initiate its deployment from the ground vehicle, traverse to predetermined waypoints, and ultimately execute a precise landing atop the UGV to facilitate recharging. This application possesses

*Address all correspondence to this author.

substantial potential, as the UGV can sustain its operational objectives concurrently with the UAV's activities. This functionality suggests that the UAV has to be capable of performing landings on both stationary and dynamically moving platforms. Among the various operational phases delineated, the landing phase represents a critical juncture that illustrates the most significant synergy between the UGV and UAV. Consequently, this study will concentrate specifically on the intricacies and implications of this phase.

The problem of a UAV landing on a static platform has been extensively researched, especially in relation to landing on recharging stations [5].In order to achieve accurate landing performance, it is essential to address two principal challenges: target identification and the precise control of movement necessary for effective landing on the designated target.

For the initial challenge, a variety of methodologies have been proposed to enhance landing capabilities in diverse conditions. One approach employs infrared sensors (like IR-LOCK) to facilitate landings during nocturnal periods [6]. However, this method presents significant limitations in sunlight conditions, which may hinder the system's overall recognition capabilities. The most widely utilized technique relies on visual camera systems that leverage fiducial markers, including ArUco markers [7] and AprilTags [8]. These markers exhibit rapid and reliable detection characteristics; nonetheless, they are susceptible to distortion phenomena if the camera is not precisely aligned with the marker [9]. Furthermore, advanced vision algorithms, such as mYolo [10], have been implemented to bolster detection efficiency. Recent investigations have also explored the synergetic integration of vision and infrared techniques to enhance robustness [11].

Recent investigations have focused on dynamic precision landing, representing an advancement over static landing techniques. This concept entails the interaction with a moving target, such as an unmanned ground vehicle (UGV) or an unmanned surface vehicle (USV) [12]. In the specific context of UAV-UGV interactions, several practical implementations have been documented, particularly in indoor environments [13] [14]. However, these implementations are often constrained by the size of the drone and rely on motion capture systems for control.

Transitioning to outdoor scenarios introduces several complexities, particularly in terms of coordination between the UAV and UGV. For instance, Priambodo et al. [15] propose a hybrid GPS-vision algorithm that effectively identifies and enables landing onto static platforms. In dynamic scenarios, such as when the UAV aims to land atop a UGV moving at a constant speed, various methodologies have been explored. Some approaches focus on estimating the UGV's velocity to anticipate its trajectory, allowing the UAV to execute a free-fall landing [16]. Alternatively, other methods involve the implementation and continuous tuning of a PID controller to facilitate the landing process [17]. While this latter approach is valid, it requires the UGV to stop once the drone was above it to land correctly and we want the drone to land regardless the movement of the UGV.

In this study, we present a novel methodology aimed at addressing the challenges associated with landing maneuvers in dynamic environments where the platform is in continuous motion. By employing a least squares regression algorithm to accurately estimate the velocity of the ground vehicle, we effectively mitigate the velocity differential between this and the drone. Subsequently, we utilize a Proportional-Integral-Derivative (PID) velocity controller to establish a precise descent velocity profile for the drone to follow. Our findings demonstrate that the performance achieved under these conditions is comparable to that of the precision landing algorithm traditionally utilized for stationary landing pads, all while maintaining consistency through the application of PID controller parameters optimized in static scenarios. Furthermore, we successfully achieved a controlled landing trajectory, thereby reducing the inaccuracies typically associated with free fall dynamics.

We tested the developed algorithm for different velocities of the moving platform and proved the effectiveness by leveraging the Gazebo environment [18], always thinking of the future hardware implementation and the generic characteristics of each robot. The remaining paper is organized as follows: Sec. 2 presents the overall high-level control algorithm and its composition, Sec. 3 presents the developed algorithm for the landing procedure, Sec. 4 details the simulation models used for testing and the PID tuning procedure, Sec. 5 highlights the simulation results, Sec. 7 provides conclusions, and Sec. 8 outlines future works.

To manage drone movement effectively and facilitate smooth, rapid landings, extensive research has been conducted into model-based simulations [19] [20] and the application of digital twins [21] for estimating optimal landing trajectories.

2 Mission Planner: High level controller

To advance collaboration and communication among diverse robotic platforms, notably Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs), we have formulated a high-level architectural framework. This architecture is designed to be agnostic of specific robotic systems, thereby facilitating its applicability across a broad spectrum of robotic technologies. It is adept at interfacing with low-level controllers such as PX4 [22] and Ardupilot [23] utilized in drone operation, thus enabling seamless integration and enhanced interoperability between varying robotic entities. We capitalized on the capabilities of ROS2, which is particularly effective in facilitating the creation of a modular control structure. The central idea is to implement a mission designer that defines the mission parameters and outlines the sequence of actions each robot must undertake. This mission designer generates a YAML file [24] for each robot, detailing the actions to be performed along with the estimated time

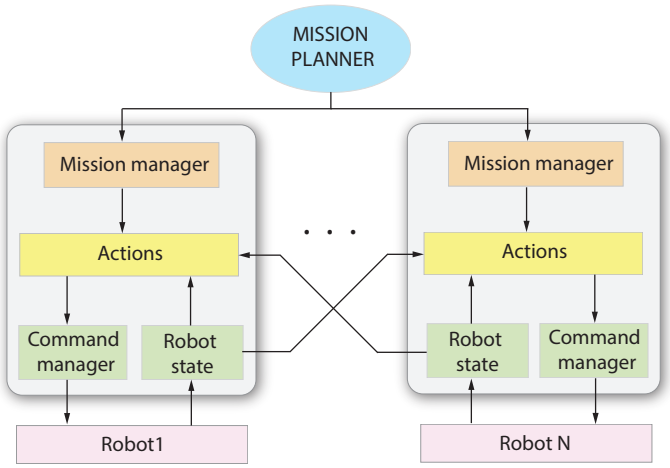

FIGURE 1: Proposed high-level control architecture for multi-robot applications. The system consists of several components: a common mission planner node that organizes and designs specific missions for each robot, which are then passed to the action node. The action node decomposes complex tasks into smaller, manageable tasks and hands them off to the command manager for the execution of low-level operations. Additionally, a robot state node provides essential information to ensure the effective functioning of the actions. Notably, all robots in the network can access each other robot's state node, promoting enhanced cooperation among them.

needed to complete it and some associated parameters. For instance, if a robot is required to navigate to a specific location, the coordinates of that destination would be included as parameters.

The architecture consists of four main components as shown in Figure 1.

1. A mission manager node that receives high-level commands either from a YAML file for pre-planned missions or via a topic for real-time commands.
2. A collection of actions that serve as commands for the robot.
3. A robot state node that continuously sends feedback to the robot.
4. A command manager node is responsible for interacting with the low-level system, such as the motors or the local low-level controller if already implemented.

The benefits of this approach are that it enables standardization and scalability. Standardization enables integration on any hardware by suitably modifying the command manager and state node. Scalability allows one to add any number of robots to the system.

2.1 Mission Manager

The mission manager node acts as the interface between the robot and the external environment. It is tasked with processing all information related to the robot, including input from the external mission planner node, and managing the actions the robot needs to perform. Throughout this process, it consistently monitors the elapsed time to ensure smooth transitions between phases and to identify any emergency situations that may require halting the robot's normal operations.

As previously mentioned, the mission manager retrieves commands from a YAML file but also listens for online commands through a specific topic. These online commands may include emergency conditions or mission replanning, enabling the node to be reinitialized during operation to adapt to changes, such as in re-routing applications.

Upon startup, the mission manager loads all commands from the YAML file and processes them sequentially, executing each corresponding action. Once a command is successfully executed, the mission manager advances to the next command in the queue. This process continues until all commands have been completed.

If any action is interrupted before it is fully executed, the mission manager sends an emergency command to the robot, prompting it to either execute an emergency landing (in the case of a drone) or come to a complete stop (for ground robots).

2.2 Actions

The actions are used to break down complex tasks into manageable components. They serve as the connection between the mission manager, which acts as the high-level controller, and the command manager, responsible for executing simpler actions. They also connect with the robots' state nodes and obtain from them any information that is required to complete the mission correctly.

Several actions, such as *takeoff*, *moving to location*, and *landing*, have been implemented for drones, along with *moving to location* for ground vehicles.

2.3 Command Manager

The command manager node, like the state node, is highly dependent on the specific robot that is being used. Its primary function is to convert low-level control commands, such as position or velocity commands, into a format that the robot can execute. It does not perform complex logic; instead, it simply executes the commands as they are received. This node acts as a bridge between the ROS2 algorithm and the low-level controller.

For example, in the case of a drone, the command manager is responsible for sending any command received to PX4 in the format required by the controller. This integration can be one of the most challenging aspects, as it is highly dependent on the specific robot. However, if a family of robots is controlled in a

consistent manner, such as drones using PX4, this node can be reused effectively.

2.4 Robot State

Lastly, the robot state node is responsible for keeping track of the important characteristics of each robot and all the information that may be needed. It interfaces directly with the robot and the actions that are being performed. Since the logic is executed within the action framework, the state node provides all relevant information about the robot. For example, it can convey details such as the robot's battery level or whether an emergency condition has been triggered. The node also retains information regarding the robot's odometry, specifically the estimation of its position, linear and angular velocity, and linear acceleration. For instance, in the case of landing, the drone's landing action interacts with the ground vehicle state node to acquire the vehicle's global position.

Given that the important information provided by the robots is largely similar, we decided to utilize a single message type for all of them. This message includes essential data that all robots share, such as their global positions, velocities and accelerations, as well as specific information pertinent to individual types of robot, such as whether a drone is currently flying or not. The advantage of having a uniform message across different robot platforms is that all robots in the network can access this information. This is particularly beneficial in scenarios like precision landing, where the drone needs to know the estimated position of the ground vehicle during the initial phase of landing to successfully reach it. The state node has been designed to keep sending the robot information at a rate of 10 Hz thus to ensure a fast and continuous stream of data.

3 Landing Algorithm

We discuss the operational framework that enables the unmanned aerial vehicle (UAV) to land on the moving unmanned ground vehicle in a diverse array of scenarios.

Initially, the UAV is positioned at a considerable distance from the UGV, such that it cannot detect the latter visually through its onboard camera. Upon the initiation of the landing command, the UAV undertakes a four-phase process:

1. **Search for Unmanned Ground Vehicle (UGV)**: The initial phase involves the Unmanned Aerial Vehicle (UAV) determining the absolute coordinates of the UGV by establishing a connection to the UGV state node, enabling continuous acquisition of the UGV's estimated GPS location. Subsequently, the UAV navigates toward this predetermined position while concurrently executing a search for the UGV. The UGV is equipped with an identification tag, specifically an ArUco marker. During its approach to the UGV, the UAV utilizes its downward-facing camera to detect the marker; upon successful detection, the UAV initiates a tracking sequence to estimate the UGV's velocity.
2. **Estimation of UGV Velocity**: Upon the detection of the ArUco marker affixed to the UGV, the UAV uses a least squares regression algorithm to estimate the UGV's velocity. Once the UAV reaches a satisfactory confidence level in its velocity estimation, it proceeds to the next operational

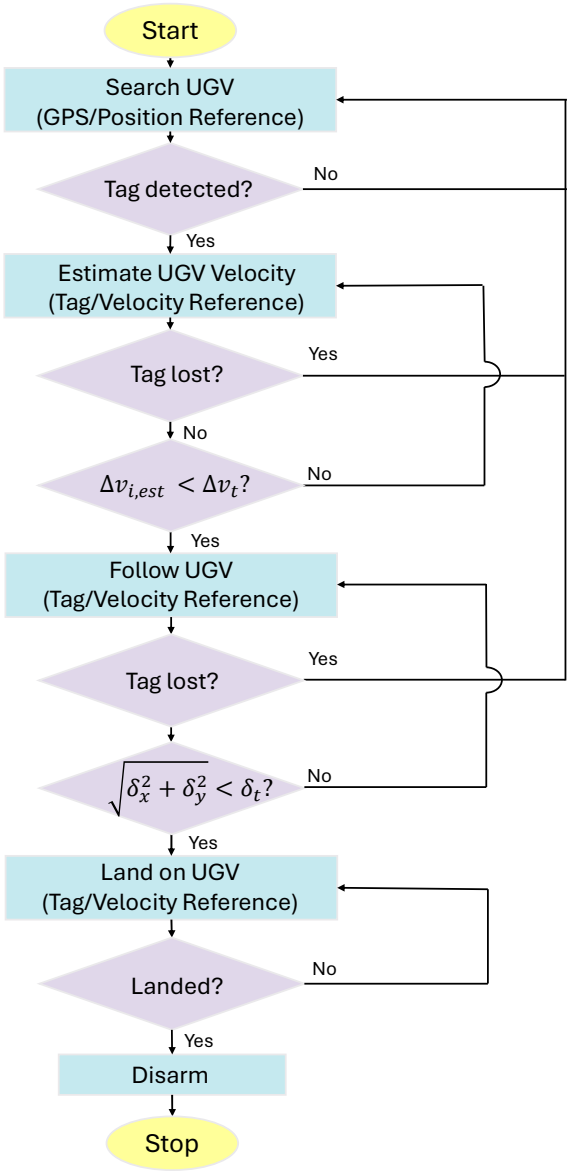

FIGURE 2: Flowchart of the landing action algorithm. δ_x and δ_y represent the distances between the center of the tag and the center of mass of the drone in the x and y directions, respectively. δ_t is the hyperparameter threshold that activates the landing condition.

phase.

3. **Follow the UGV**: To facilitate the following maneuver, the UAV adapts its control strategy from GPS-based position reference to ArUco marker-based velocity reference. In this configuration, the estimated velocity of the UGV is communicated to the PX4 flight control system, aligning the drone's x-y velocity with that of the ground vehicle while maintaining a zero vertical velocity. A Proportional-Integral-Derivative (PID) controller is subsequently utilized to minimize the distance between the UAV and the UGV.
4. **Landing on the UGV**: Finally, when the UAV's horizontal distance from the UGV reduces below a defined error threshold for a predetermined duration—signifying the UAV's position directly above the UGV and synchronization of their velocities—the UAV initiates its landing maneuver. This maneuver involves a descent toward the UGV with a constant downward velocity.

Figure 2 illustrates the finite state machine utilized for the landing algorithm. Upon a thorough examination of the control scheme presented, the following delineates the primary components identified within the framework.

3.1 Position Control

This phase employs a GPS-based position reference control, whereby the coordinates of the ground vehicle serve as the reference position for the drone. Once the drone determines the global position of the ground vehicle, this information will be transmitted to the low-level PX4 control algorithm, responsible for executing the navigation process. Here, the PX4 controller evaluates the torques of the propellers' motors by taking as input the difference between the desired position and the actual position of the drone. The latter is estimated by using an extended Kalman Filter to combine the GPS position and the estimation from the drone's IMU.

3.2 Tag Detection

After careful consideration, we selected ArUco markers for our simulation, primarily owing to their compatibility with the PX4 Software In The Loop (SITL) simulation environment and the availability of robust software libraries for their detection.

We utilize the *aruco tracker* package offered by ARK Electronics [25] as it offers seamless integration with both ROS2 and PX4.

The velocity control in PX4 operates by providing the software with velocity values in the North-East-Down (NED) reference frame (where the x-axis is oriented along the North, y-axis along East and z-axis points downward). Consequently, to perform our calculations, we need to transform the position and orientation of the tag from the camera frame to the NED reference frame. Using the notation $\vec{x}_b^a$ to denote the position $\vec{x}$ of object b in reference frame a, and R_d^c to represent the rotation matrix R of frame d with respect to frame c, we can express the transformation as:

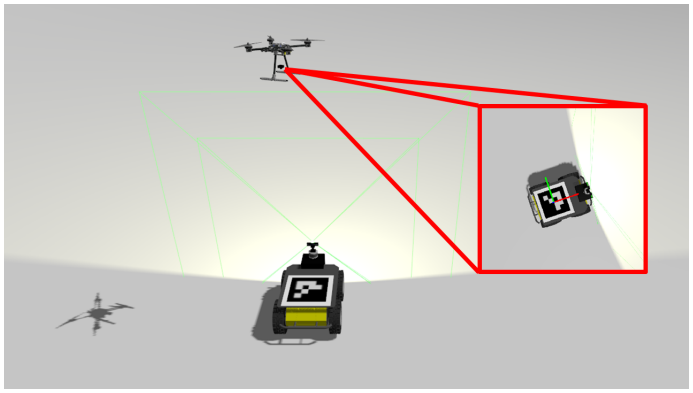

FIGURE 3: Simulation of a drone (Holybro X500) and ground vehicle (Husky A200) operating in a Gazebo environment. The ground vehicle is equipped with an Aruco Marker, facilitating the implementation of the precision landing algorithm. The image on the right displays the live feed from the drone's camera, the ArUco marker reference frame can be seen in green, red and blue.

$$\vec{x}_{tag}^{NED} = R_{cam}^{NED}\vec{x}_{tag}^{cam} + \vec{x}_{cam}^{NED} \tag{1}$$

In this equation, R_{cam}^{NED} and $\vec{x}_{cam}^{NED}$ denote the rotation matrix of the camera frame and the position of the camera in the NED reference frame, respectively, both of which are fixed values. Meanwhile, $\vec{x}_{tag}^{cam} = \{x_{tag}^{cam}, y_{tag}^{cam}, z_{tag}^{cam}\}$ represents the position of the center of the tag as detected by the camera. We have not considered the orientation of the tag since our primary objective is for the drone to land on the target, irrespective of the tag's attitude.

3.3 Estimation phase

During the estimation phase, the drone is maintained at an altitude optimized for the detecting algorithm to effectively identify the target. This altitude is strategically selected to ensure it is not excessively low, thereby mitigating the risk of the ground vehicle exiting the camera's field of view (FOV) before the algorithm can accurately assess the UGV's velocity.

Upon detection of the tag ID, the drone will stabilize its position and continuously track the tag's location as the UGV traverses the landscape. Concurrently, the system logs the current time, and for each localization instance, it stores both the time of localization and the corresponding positional data. At each discrete time step, a least-squares logistic regression analysis is

conducted for both the x and y components of the tag's trajectory. Once the discrepancy between the newly estimated velocity and the previously determined velocity falls below a specified threshold, the slope of the resulting estimated curve is subsequently documented. This slope is then used as an estimate of the velocity of the UGV.

In the process of updating localizations, three critical parameters, t_N, x_N, and y_N, are utilized to refine a set of six cumulative summations: $\sum_i^N t_i$, $\sum_i^N t_i^2$, $\sum_i^N x_i$, $\sum_i^N x_i t_i$, $\sum_i^N y_i$, and $\sum_i^N y_i t_i$. Additionally, the variable N represents the total number of observations recorded. We keep all recorded values for estimation as convergence is fairly quick as noted in the results.

$$v_{est,x,N} = \frac{N\sum_i^N x_i t_i - \sum_i^N t_i \sum_i^N x_i}{N\sum_i^N t_i^2 - (\sum_i^N t_i)^2}, v_{est,y,N} = \frac{N\sum_i^N y_i t_i - \sum_i^N t_i \sum_i^N y_i}{N\sum_i^N t_i^2 - (\sum_i^N t_i)^2}$$

3.4 Velocity Control

For the Tag-based velocity reference control, we divide it into two parts: the horizontal plane (xy plane) and the vertical direction(z).

3.4.1 Horizontal movement The horizontal control is employed in two phases: the following phase and the landing phase. The approach differs slightly between these two scenarios. During the following phase, the objective is to minimize the distance in the x and y directions between the drone's center of mass and the tracking tag. In contrast, during the landing phase, the drone aims to land while maintaining the estimated UGV's velocity.

We control the x-axis velocity separately from the y-axis velocity, thus simplifying the control. Our objective is twofold: we aim to minimize both the position and velocity differences between the drone and the landing platform.

To achieve velocity and position tracking of the UGV, we use a feedforward compensation on the UGV velocity and proportional-integral-derivative control on the UGV position. Our control is:

$$v_i(t+1) = v_{i,UGV} + v_{i,PID}(t),$$

where i designates either the x or y axis.

$$v_{i,PID}(t+1) = P \cdot \delta_i(t) + I \cdot \sum_{j=0}^{t} \delta_i(j) + D \cdot (\delta_i(t) - \delta_i(t-1))$$

Here, P, I, and D represent the proportional, integral, and derivative gains. The term $\delta_i(t)$ is instead the error between the position of the drone and the tag as obtained by the detection algorithm.

The condition for transitioning from the following phase to the landing phase is determined by ensuring that the quantity $\sqrt{\delta_x^2 + \delta_y^2}$ remains below a defined hyperparameter δ_t for at least one second.

3.4.2 Vertical movement In the analysis of vertical dynamics, two distinct phases are delineated: the following phase and the landing phase. During the following phase, it is observed that the drone does not align its position with that of the ground vehicle; however, the velocities between the two entities remain approximately equivalent, as previously estimated. It is imperative to maintain the drone at an altitude sufficiently low to ensure the continued detectability of the tag and its proximity to the camera apparatus. In this context, the drone may be positioned relatively closer to the ground vehicle, as our objective is to achieve the centrality of the tag within the camera's field of view (FOV). To facilitate this alignment, we have implemented a function that adjusts the reference altitude downward at a rate of 0.2 m/s, that is to help the drone achieve optimal centering with respect to the UGV. The reference altitude is defined as the positive hyperparameter z_{ref}, which is calibrated in accordance with the dimensions of the tag and the specific parameters of the implementation. The control of the command velocity in the z-direction is governed by a proportional control algorithm.

$$v_z(t+1) = P_z \cdot (z_{tag}^{NED} - z_{ref})$$

In this equation, P_z represents the proportional gain, which depends on the specific drone. It is important to note that, due to the definition of the North-East-Down (NED) reference frame, a negative velocity indicates an upward movement.

4 Simulation Environment

To test the algorithm, we opted to utilize the functionalities of Gazebo, specifically Gazebo Harmonic, since PX4 already has a Software-in-the-Loop (SITL) implementation with it. Unfortunately, the ground vehicle we planned to use only supports Gazebo Ignition, which required us to develop the algorithm from the ground up. We chose Gazebo for its interoperability with ROS2 and its powerful graphical interface, which was beneficial for testing. For ROS2, we used the Humble version, as it is one of the latest releases, and the previous version has reached its end of life. Below is a description of all the models used during the simulation and how we integrated them.

4.1 UAV Model

The drone model we utilized is the default Holybro X500 drone used for simulation. We chose the

version with a downward-mounted camera, referred to as *x500_mono_cam_down* because it was necessary for detecting the tag of the ground vehicle.

To integrate PX4 with ROS2, we used XRCE-DDS to create a bridge between the two systems, allowing for seamless connectivity with the drone.

4.2 UGV Model

For the UGV model, we aimed to make the simulation as close to real hardware as possible, particularly concerning the dimensions of the ground vehicle. This understanding was crucial for determining the appropriate size of the tag to use and for better calibrating the altitude at which the drone should begin searching for the tag.

Gazebo provides a variety of robot models for use in the simulator, and we chose to use a basic version of the Husky robot. This version has the fewest components, and we placed a 0.4mx0.4m ArUco marker on top of the main plate of the Husky. Additionally, we used velocity commands to control the wheels, allowing the Husky to move in the desired direction at the specified speed.

We have set the speed limit of the ground vehicle to 1 m/s, which is the maximum velocity of the real ground robot.

4.3 World Model

Once we had fully defined the models for both the ground vehicle and the drone, we needed to establish the world environment for the simulation. To achieve this, we used the "empty world," which is the simplest configuration featuring just the physics plugin and nothing else, since we were interested in just proving the effectiveness of the algorithm.

Table 1 delineates various characteristics of the robots employed in this study.

	Holybro X500	Husky A200
type	UAV	UGV
dimensions (mm)	500 × 500 × 160	990 × 670 × 390
weight (kg)	2	50
max speed (m/s)	15-20	1

TABLE 1: The specific characteristics of the Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) utilized in the simulation.

4.4 Landing Control using PID control

With all the components appropriately configured, it became essential to optimize the Proportional-Integral-Derivative (PID) controller to guarantee that the drone could achieve its desired position and execute a precise landing. The controller takes as inputs the current distances between the drone's center of mass and the tag's center along the x and y axes, denoted as $\delta_x(t)$ and $\delta_y(t)$, which correspond to the north and east components of the North-East-Down (NED) coordinate system. It subsequently generates a reference velocity in the following form:

$$v_{i,PID}(t+1) = P \cdot \delta_i(t) + I \cdot \sum_{j=0}^{t} \delta_i(j) + D \cdot (\delta_i(t) - \delta_i(t-1))$$

In this context, P, I, and D denote the tunable parameters that dictate the system's response behavior. The variable i can represent either the x or y axis. Notably, we employ the identical set of parameters for regulating the velocities along both the x and y axes, thereby ensuring a uniform approach to control within the two-dimensional framework. Additionally, the velocity input is constrained within the range of [-3 m/s, 3 m/s] to mitigate the occurrence of excessive oscillations. This limitation is implemented to ensure system stability and prevent undesirable dynamic responses.

In the static scenario, the unmanned ground vehicle (UGV) remains stationary while the drone is dispatched. After the drone moves away, it acquires the UGV's local coordinates and returns to its original position. Once the drone detects the tracking tag, the PID control is activated. For the tuning process, we began by applying the method presented by Wescott et al. [26]. The objective is to have the drone reach the center of the tag as quickly as possible while minimizing oscillations around it.

In the dynamic scenario, the unmanned aerial vehicle embarks on its flight shortly prior to the commencement of movement by the unmanned ground vehicle. Following takeoff, the UAV is deployed for a limited duration before the UGV initiates transmission of its current positional data. As the UAV identifies the UGV, the estimation algorithm for positioning activates, determining the UGV's velocity. The proportional-integral-derivative (PID) tuning parameters employed previously in the static scenario are applied once more. Upon achieving alignment between the UAV's velocity and positional data with those of the UGV, the landing phase is started.

5 Results

The testing conducted within the simulation environment yielded several noteworthy factors warranting comprehensive analysis. Initially, we selected an altitude of 7 meters above the

ArUco marker, as the tag detection algorithm exhibited significant challenges in accurately identifying the position of the tag, particularly during instances when the ground vehicle was in motion. Furthermore, we established a threshold value of 0.1 meters for centering the drone, which proved to be sufficiently precise considering that the tag measures 0.4 meters in size. In addressing the velocity error associated with the estimation algorithm, we determined an optimal value of 0.05. This choice was made to ensure a highly reliable estimation while simultaneously maintaining computational efficiency and preventing excessive algorithmic latency.

In this study, we conducted a series of tests to evaluate the algorithm's performance under varying velocities of the Unmanned Ground Vehicle (UGV). The simulation was designed with the ground vehicle spawned at the coordinates (0,0) within the North-East-Down (NED) reference frame, while the drone commences its operation directly above it. The simulated scenarios consisted of initially sending the drone to the position (-10 , 20) m while the UGV's velocity was systematically varied at intervals of 0 m/s, 0.2 m/s, 0.4 m/s, 0.6 m/s, and 0.8 m/s, 1m/s along the x-axis. Fig. 4 shows the behavior of the estimation algorithm for different velocities of the UGV, it can be seen that the algorithm is very fast at estimate the velocity since in all of the cases it was able to estimate it within 4 seconds.

Fig. 5 presents the position of the ground vehicle and of the drone during a complete mission when the velocity of the UGV is set to 0.4 m/s. In the figure, the landing phase is highlighted with different colors representing the different phases that compose it. It can be seen that the drone takes around 25 seconds to land on the UGV from when the command is given. Among the four parts, the estimation takes the highest amount of time of around 12 seconds. The search time is of course related to the initial position of both the drone and the ground vehicle and therefore is highly variable.

In the process of tuning the control gains, we focused on two primary performance metrics: settling time and oscillation amplitude. Settling time was defined as the duration during which the drone's relative position to the target, in both the x and y axes, remained within a specified threshold of $\delta_t = 0.1\,\text{m}$. This interval was measured following the transition from position control to velocity control commands. Additionally, the amplitude of oscillations was evaluated by determining the maximum overshoot produced by the control algorithm after this transition.The tuned hyperparameters are set to $P = 0.45$, $I = 0.05$, and $D = 0.5$.

6 Discussion

As anticipated, we encountered minimal difficulties while fine-tuning the PID controller in the static scenario—specifically, when the ground vehicle remained stationary. By estimating the velocity of the ground vehicle and then imposing that as a base reference for the drone, we were able to obtain the same results as the static scenarios also while the ground vehicle was moving.

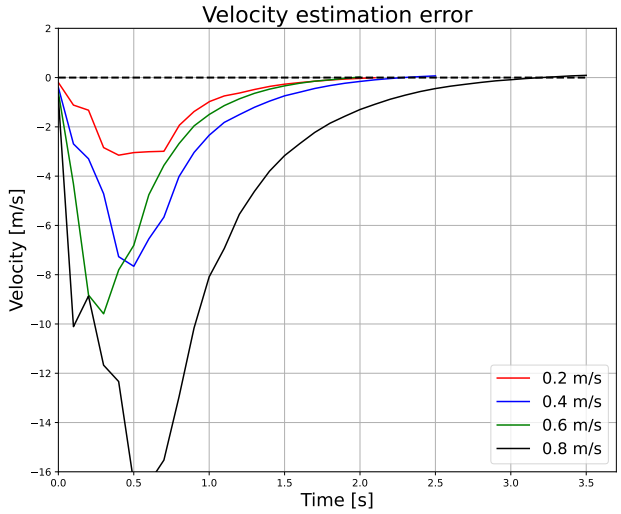

FIGURE 4: The graph illustrates the velocities estimated by the least square algorithm for different UGV ground velocities.

The findings concerning the landing algorithm indicate that both the estimation and the PID control algorithms yielded highly favorable results, demonstrating the capability to execute landing maneuvers with remarkable speed and efficacy. However, certain limitations were identified, particularly in scenarios where the drone approaches the target at a minimal distance. In such instances, there exists a brief interval during which the drone operates without visual feedback, relying solely on the estimated velocity for guidance. Despite these challenges, the drone consistently achieved targeted landings, albeit occasionally deviating slightly from the center. This phenomenon is largely attributable to the constraints imposed by the size of the visual tag, which must remain detectable from elevated altitudes.

The high-level control architecture utilized in ROS2 has demonstrated considerable efficacy in managing inter-robot communication and enhancing coordination during task execution. By developing an action server that delineates the requisite commands, this new algorithm can be seamlessly incorporated into the existing architecture.

7 Conclusions

This paper introduces a novel algorithm designed for the precision landing of aerial drones atop mobile ground vehicles. The proposed algorithm has undergone extensive simulations, demonstrating optimal performances in landing operations. A key feature of this approach is the implementation of a linear regression filter for estimating the velocity of the ground vehi-

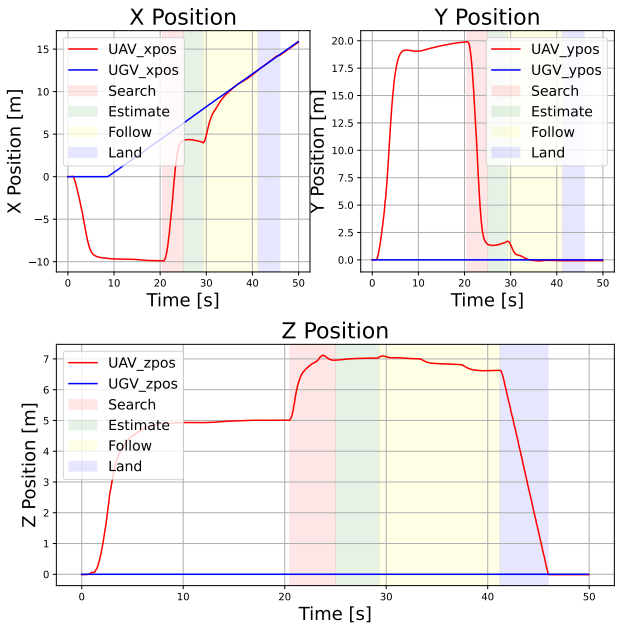

FIGURE 5: The graphs illustrate in red the position of the drone and in blue the position of the ground vehicle with respect to time for a mission where the ground vehicle moves along the x-axis at a velocity of 0.4 m/s. The four phases of the landing are highlighted: Search (red), Estimate (green), Follow (yellow), Land(blue)

cle. This design choice contributes to the algorithm's overall efficiency, allowing the dynamic landing scenario to be treated similarly to a static one through the application of a straightforward PID velocity controller. The results underscore the algorithm's efficacy and potential for real-world applications in drone navigation and autonomous landing systems.

8 Future Works

The findings delineated in this paper are currently limited to a simulation framework; thus, a logical and essential next step would entail the implementation and empirical assessment of the proposed control strategy in a real-world outdoor environment. This transition is critical, as it encompasses a range of additional complexities, including networking and communication challenges that necessitate careful consideration and resolution. Moreover, there exists the potential for enhancements to be directed toward the velocity control algorithm. Specifically, the traditional PID controller could be supplanted with a more advanced model-based control approach. Additionally, modifications aimed at refining the velocity estimation component of the algorithm could be pursued, expanding its applicability beyond scenarios characterized by constant velocity.

ACKNOWLEDGMENT

This work was supported by ARO Contract Number W911NF2420018

REFERENCES

[1] Sudhakar, S., Vijayakumar, V., Kumar, C. S., Priya, V., Ravi, L., and Subramaniyaswamy, V., 2020. "Unmanned aerial vehicle (uav) based forest fire detection and monitoring for reducing false alarms in forest-fires". Computer Communications, **149**, pp. 1–16.

[2] Mondal, M. S., Ramasamy, S., Humann, J. D., Dotterweich, J. M., Reddinger, J.-P. F., Childers, M. A., and Bhounsule, P., 2024. "A robust uav-ugv collaborative framework for persistent surveillance in disaster management applications". In 2024 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 1239–1246.

[3] Mondal, M. S., Ramasamy, S., and Bhounsule, P., 2025. "Deep reinforcement learning enabled persistent surveillance with energy-aware uav-ugv systems for disaster management applications". arXiv preprint arXiv:2502.02666.

[4] Ramasamy, S., Reddinger, J.-P. F., Dotterweich, J. M., Childers, M. A., and Bhounsule, P. A., 2022. "Coordinated route planning of multiple fuel-constrained unmanned aerial systems with recharging on an unmanned ground vehicle for mission coverage". Journal of Intelligent & Robotic Systems, **106**(1), p. 30.

[5] Nelson, B., Du Preez, J., van Niekerk, T., Phillips, R., and Stopforth, R., 2020. "Autonomous landing of a multirotor aircraft on a docking station". In 2020 International SAUPEC/RobMech/PRASA Conference, IEEE, pp. 1–6.

[6] Janousek, J., and Marcon, P., 2018. "Precision landing options in unmanned aerial vehicles". In 2018 International Interdisciplinary PhD Workshop (IIPhDW), IEEE, pp. 58–60.

[7] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., and Marín-Jiménez, M. J., 2014. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In Pattern Recognition, pp. 228–237.

[8] Olson, E., 2011. "Apriltag: A robust and flexible visual fiducial system". In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3400–3407.

[9] Kim, D., Bong, J. H., and Jeong, S., 2024. "Enhancing pose estimation using multiple graphical markers with spatial and temporal outlier detection". Applied Sciences, **14**(22), p. 10225.

[10] Rao, Y., Ma, S., Xing, J., Zhang, H., and Ma, X., 2020.

"Real time vision-based autonomous precision landing system for uav airborne processor". In 2020 Chinese Automation Congress (CAC), IEEE, pp. 532–537.

[11] Badakis, G., Koutsoubelias, M., and Lalis, S., 2021. "Robust precision landing for autonomous drones combining vision-based and infrared sensors". In 2021 IEEE Sensors Applications Symposium (SAS), IEEE, pp. 1–6.

[12] Gupta, P. M., Pairet, É., Nascimento, T., and Saska, M., 2022. "Landing a uav in harsh winds and turbulent open waters". IEEE Robotics and Automation Letters, **8**(2), pp. 744–751.

[13] Wenzel, K. E., Masselli, A., and Zell, A., 2011. "Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle". Journal of intelligent & robotic systems, **61**, pp. 221–238.

[14] Mondal, M. S., Ramasamy, S., Humann, J. D., Reddinger, J.-P. F., Dotterweich, J. M., Childers, M. A., and Bhounsule, P., 2023. "Optimizing fuel-constrained uav-ugv routes for large scale coverage: Bilevel planning in heterogeneous multi-agent systems". In 2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS), pp. 114–120.

[15] Priambodo, A., Arifin, F., Nasuha, A., Winursito, A., et al., 2022. "A vision and gps based system for autonomous precision vertical landing of uav quadcopter". In Journal of Physics: Conference Series, Vol. 2406, IOP Publishing, p. 012004.

[16] Rigogliuso, L., 2022. "Robust autonomous landing of a uav on a high-speed platform". PhD thesis, Politecnico di Torino.

[17] Polvara, R., Sharma, S., Wan, J., Manning, A., and Sutton, R., 2017. "Towards autonomous landing on a moving vessel through fiducial markers". In 2017 European Conference on Mobile Robots (ECMR), IEEE, pp. 1–6.

[18] Koenig, N., and Howard, A., 2004. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp. 2149–2154.

[19] Herisse, B., Hamel, T., Mahony, R., and Russotto, F.-X., 2010. "The landing problem of a vtol unmanned aerial vehicle on a moving platform using optical flow". In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp. 1600–1605.

[20] Antenucci, A., Mazzaro, S., Fiorilla, A., Messina, L., Massa, A., and Matta, W., 2020. "A ros based automatic control implementation for precision landing on slow moving platforms using a cooperative fleet of rotary-wing uavs". In 2020 5th International Conference on Robotics and Automation Engineering (ICRAE), IEEE, pp. 139–144.

[21] Uddin, J., Wadud, M. F., Ashrafi, R., Alam, M. G. R., and Rhaman, M. K., 2023. "Landing with confidence: the role of digital twin in uav precision landing". In 2023 10th International Conference on Recent Advances in Air and Space Technologies (RAST), IEEE, pp. 1–6.

[22] Meier, L., Honegger, D., and Pollefeys, M., 2015. "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms". IEEE International Conference on Robotics and Automation (ICRA) - Open Source Robotics.

[23] Team, A. D., 2025. Ardupilot: Open source autopilot. Accessed: 2025-03-22.

[24] Humann, J., Ortega, S. C., Glenn, J., Folsom, J. L., Ramasamy, S., Mondal, M. S., Bhounsule, P., Reddinger, J.-P., and Dotterweich, J., 2024. "Data model and simulation for persistent mission planning with energy-sharing autonomous ground and air vehicles". In 2024 Winter Simulation Conference (WSC), pp. 2070–2081.

[25] Electronics, A., 2025. Tracktor beam. Accessed: 2025-03-22.

[26] Wescott, T., 2000. "Pid without a phd". Embedded Systems Programming, **13**(11), pp. 1–7.