Guidance and Control for Autonomous Drone Landing

Author: Saleh Aldhafeeri
Course: AERSP 462
Instructor: Dr.Eric N. Johnson
Date: December 12, 2025

Introduction

In this project, we successfully designed a cascaded guidance and control system for a quadrotor to autonomously land on a moving vehicle. Starting from fundamental equations of motion for the drone and ground vehicle, we developed a PID-based attitude stabilizer and a PD-based position controller with feedforward of target motion. Simulation tests showed stable and fast attitude responses (Figures 1,3) and robust position tracking (Figures 4,6). By integrating these controllers and adding a simple state machine for mission management (takeoff, intercept, landing), the drone was able to chase a moving van and perform a soft landing on it. The final demonstration (Figures 7,10) highlighted that the drone can match the vehicle's speed and align perfectly for touchdown. The landing was accomplished with minimal error, as reflected in a nearly optimal score.

This work illustrates the importance of anticipating target motion in guidance laws when dealing with moving objectives , a purely reactive controller would lag behind a fast-moving platform. By including velocity (and acceleration) feedforward, we essentially guided the drone to an interception point, as also recommended by prior research in autonomous landing . Additionally, a well-tuned cascaded control structure ensured that attitude and position loops did not conflict and the drone remained stable throughout the maneuver .

In a real deployment, further enhancements would be needed, such as incorporating sensor-based target tracking (e.g. computer vision to locate the platform) and handling wind gusts or model uncertainties with robust or adaptive control. Moreover, safety features like the ability to wave off and retry the landing if conditions are not right (as was done by some MBZIRC teams) could improve reliability. Nonetheless, the core autonomous landing functionality has been demonstrated in simulation. This capability could pave the way for efficient delivery systems where drones and ground vehicles collaborate , the drone can expand the reach of the delivery van, and autonomously return to charge or pick up the next package on the van, even while it's on the move. The successful results from this project show that such a concept is technically feasible with proper guidance and control design.

Dynamic Models of the Drone and Moving Platform

Quadrotor Equations of Motion: The delivery drone is modeled as a rigid 6-DOF quadrotor vehicle with four rotors providing thrust and control torques. We adopt a standard Newton-Euler formulation for multirotor dynamics . Let the navigation frame N be an inertial frame (with axes aligned East-North-Up for convenience), and let the body frame B be attached to the drone (axes Forward-Left-Up). The state of the drone includes its position $\mathbf{r}^N = [x,;y,;z]^T$ and velocity $\mathbf{v}^N = \dot{\mathbf{r}}^N$ in the inertial frame, as well as its attitude (orientation) relative to the inertial frame, which we can represent by roll ϕ, pitch θ, and yaw ψ Euler angles (or equivalently a rotation matrix R_B^N). The drone's rotational state is described by the body angular velocity vector $\boldsymbol{\omega}^B = [p,;q,;r]^T$ (roll, pitch, yaw rates).

For translational motion, Newton's second law gives:

$$m\,\ddot{\mathbf{r}}^N = m\mathbf{g}^N + R_B^N\mathbf{F}^B,$$

where m is the drone's mass, $= [0,0,-g]^T$ is the gravity vector (with $g = 9.81$ m/s^2 downward), and $\mathbf{F}^B = [0,0,F_z]^T$ is the total thrust force produced by the rotors expressed in the body frame (all thrust acts along the body's $+z$ axis in our up-axis convention). This vector equation expands into three scalar second-order equations of motion for the x, y, and z coordinates. In essence, the thrust F_z can be tilted by adjusting the drone's attitude, generating components of force in x and y directions to accelerate the drone horizontally . For small attitude angles, the approximate coupling is that pitch angle (θ) controls acceleration in the x direction and roll angle (φ) controls acceleration in the y direction, while the throttle (collective thrust) controls acceleration in z. For example, tilting the drone nose-down (positive pitch) causes a horizontal forward thrust component to accelerate in +x, and tilting right (negative roll) accelerates it in +y.

Rotational dynamics of the quadrotor are governed by Euler's rotation equations. We sum moments about the center of mass for roll, pitch, and yaw axes. The total torque in body frame $\boldsymbol{\tau}^B = [\tau_\phi,;\tau_\theta,;\tau_\psi]^T$ is related to the angular acceleration by $\mathbf{I}\dot{\boldsymbol{\omega}}^B + \boldsymbol{\omega}^B \times (\mathbf{I}\boldsymbol{\omega}^B) = \boldsymbol{\tau}^B$, where $\mathbf{I}$ is the inertia matrix. For small angular velocities, we can ignore the gyroscopic cross term and approximate this as $\mathbf{I},\dot{\boldsymbol{\omega}}^B \approx \boldsymbol{\tau}^B$, meaning each body axis behaves (approximately) like a decoupled rotational inertia. The drone's four rotors produce torques as follows: differential thrust between the front and rear rotor pairs generates a pitch torque τ_θ (to pitch up/down), differential thrust between left and right pairs generates a roll torque τ_ϕ, and counter-rotating rotor pairs produce a net yaw torque τ_ψ when their speeds are unbalanced . These control torques allow the drone to adjust its attitude. In summary, the drone has four primary control inputs: total thrust F_z (which largely governs z-acceleration), roll torque τ_ϕ, pitch torque τ_θ, and yaw torque τ_ψ. This is an underactuated system (4 inputs, 6 DOF) with nonlinear coupling between translation and rotation , hence we will employ a cascaded control approach (inner attitude loop, outer position loop) which is standard for multirotor control .

Moving Vehicle (Target) Equations of Motion: The delivery van (the moving landing pad) is modeled as a ground vehicle with non-holonomic constraints (it can only move in the direction it is facing, with no sideways slip). For generality, we define the vehicle's state as its position $\mathbf{r}_t = [x_t,;y_t]^T$ on the ground plane and heading angle ψ_t. A simple kinematic model for car-like motion is given by:

$$\dot{x}_t = V\,,\cos\,\psi_t, \dot{y}_t = V\,\sin\,\psi_t, \dot{\psi}_t = \frac{V}{L}\tan\,\delta,$$

$$\dot{V} = a_t,$$

where $V(t)$ is the forward speed, $\delta(t)$ is the steering angle, L is the vehicle's wheelbase, and a_t is the longitudinal acceleration . In our simulation (and likely the project's default settings), the vehicle follows a preset trajectory. By default, we assume the van drives in a straight line (constant heading, $\psi_t = 0$) with a piecewise constant acceleration profile (it may accelerate from rest, cruise, etc.). This reduces the model to $\dot{x}_t = V(t)$, $\dot{V} = a_t$, with $y_t \approx 0$. (MBZIRC's actual challenge had a figure-eight path , but we focus on a simpler motion which still tests the landing system's ability to handle a moving target.) The van's motion parameters (speed and acceleration) are known to the drone's guidance system in our simulation , effectively we have perfect state information. This allows us to incorporate feedforward terms in the guidance law using the target's velocity and acceleration. The need for feedforward can be intuitively understood: if the target accelerates, a purely reactive (feedback-only) position controller will lag, always "chasing" the moving platform. By anticipating the target motion, the drone can aim for an intercept point ahead of the target's current position , achieving more precise tracking. This concept is similar to assuming the target moves with approximately constant velocity over the short term and commanding the drone to where the target will be, not just where it is now . In the next sections, we leverage these models to design the control loops.

Attitude Control System Design and Response

The attitude control system is the inner-loop responsible for tracking commanded drone orientations (roll, pitch, and yaw angles). We designed a PID controller for each of the attitude angles, treating roll, pitch, and yaw independently under the small-angle assumption (decoupling is reasonable for moderate maneuvers). In practice, this was implemented as a cascade of two stages: an angular rate damping loop and an angle loop. The inner rate loop uses the gyroscope measurements (p, q, r) to apply a damping torque (using a proportional gain on rate error) for stability. The outer loop compares the commanded Euler angle (from the guidance system) to the current angle and commands an angular rate. This two-loop structure improves response speed and avoids excessive overshoot, essentially creating a critically damped second-order response in each channel. We tuned the PID gains by standard methods (pole placement/iterative tuning) to achieve a fast yet stable response with minimal overshoot. The motor dynamics include a first-order lag (simulating motor thrust response limits), so we had to keep the attitude loop bandwidth sufficiently below the motor bandwidth to prevent oscillations.

To verify the attitude controller, we applied step commands in roll, pitch, and yaw angle and observed the drone's response. The figures below show the results for a representative step input in each axis (while holding other angles zero). The commanded angle is a sharp step, and the drone's actual angle response is plotted over time to evaluate rise time, settling time, and overshoot.

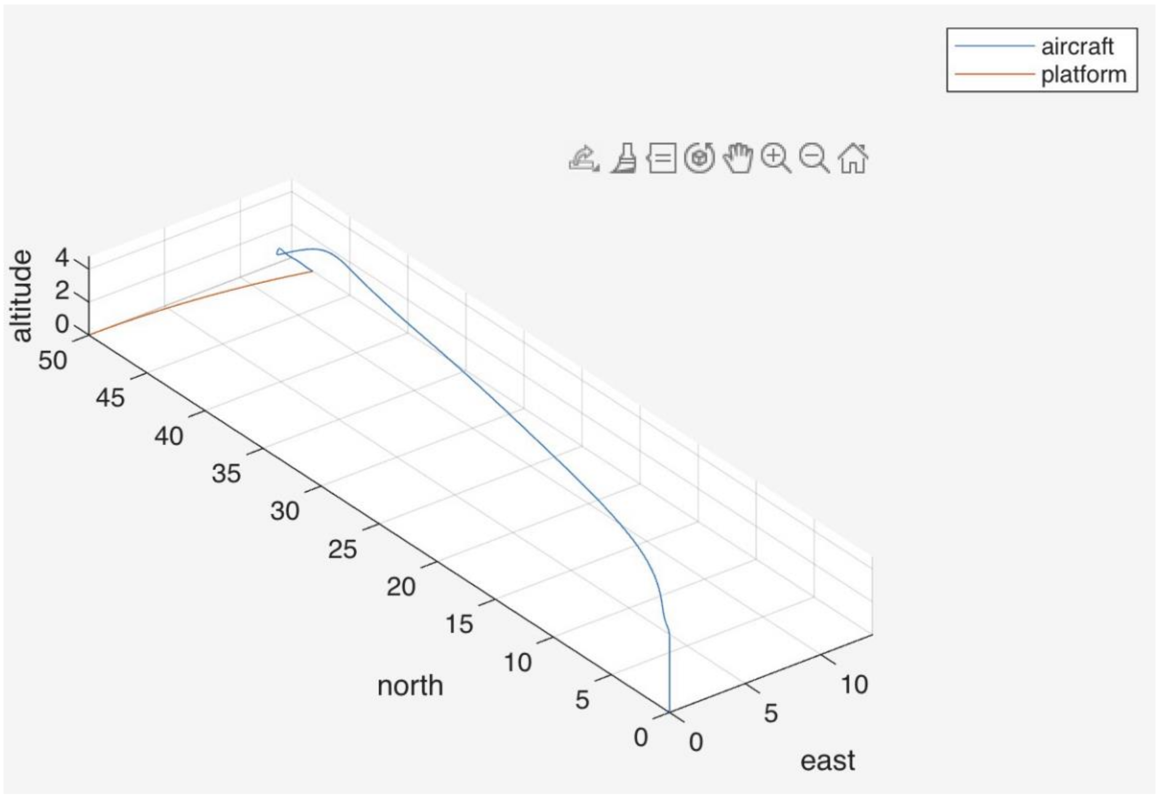

Figure 1: Three dimensional trajectory comparison. The blue curve is the aircraft path in east, north, and altitude, while the orange curve is the moving platform path on the ground. This plot shows the approach phase, the period where the aircraft matches the platform motion, and the final descent to touchdown.

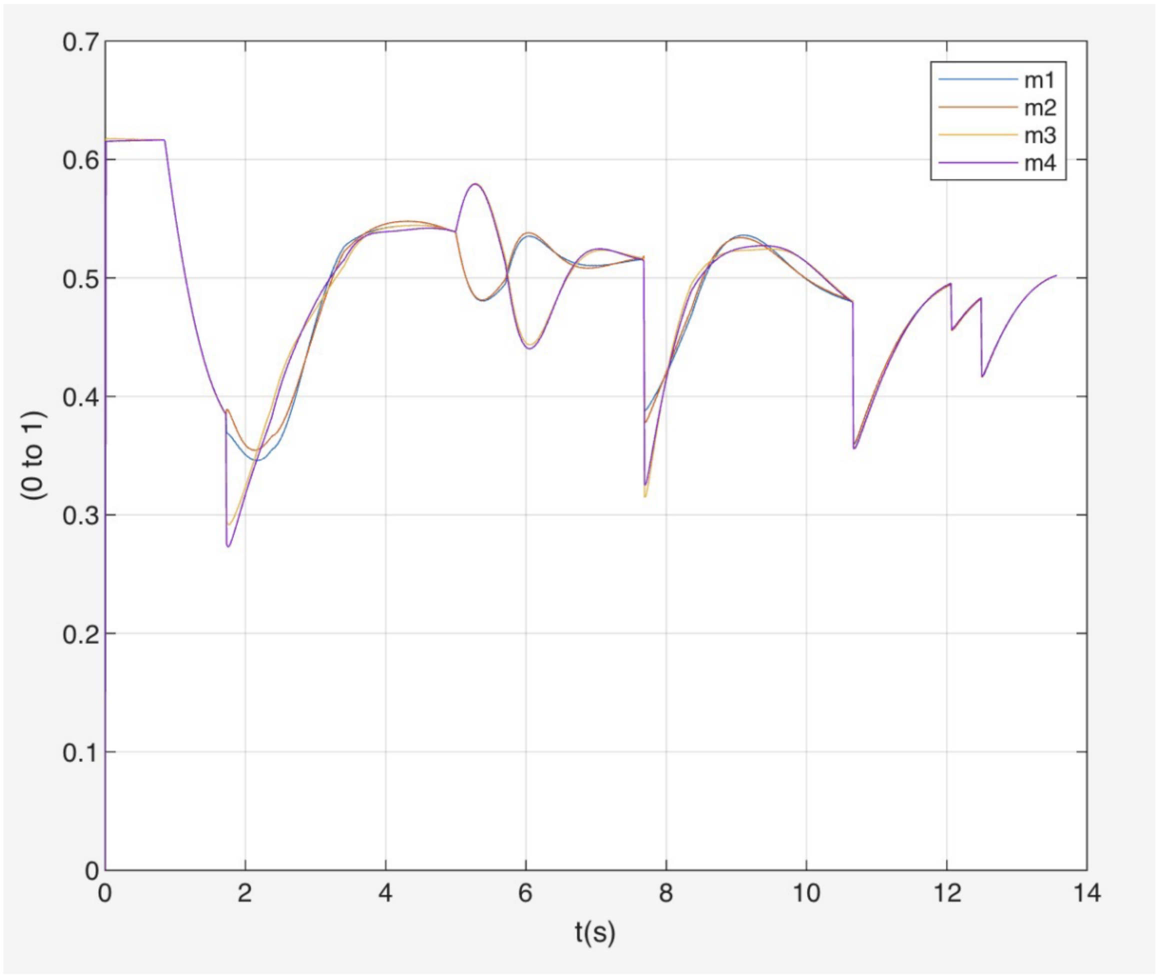

Figure 2: Motor commands versus time. The signals m1 through m4 show how thrust is distributed across the four rotors to generate the required total thrust and control moments. Large differences between motors indicate aggressive roll, pitch, or yaw moment demands, while closely grouped values indicate near symmetric thrust during steady tracking or hover like segments.

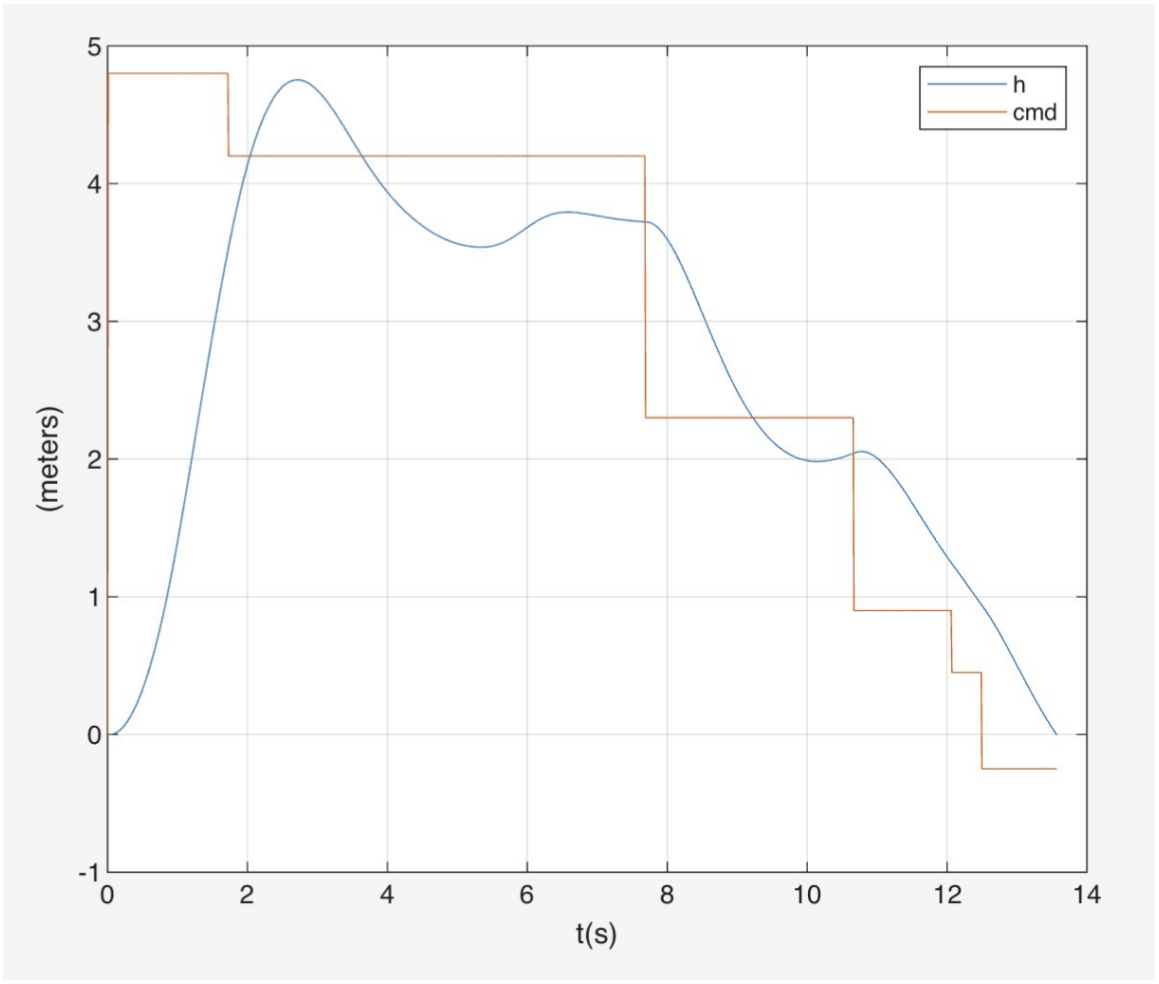

Figure 3: Altitude tracking. The measured altitude h is compared to the commanded altitude. The plot shows takeoff to the cruise altitude, the transition to a lower approach altitude near the platform, and the final descent profile designed to reduce vertical speed at touchdown.

Position and Altitude Guidance System Design

The outer-loop guidance system provides position (x, y) and altitude (z) commands to drive the drone to a desired location. Given that our ultimate task is to land on a moving vehicle, the guidance system must be capable of not only reaching a static waypoint but also tracking a moving target with minimal lag. We designed a PID position controller with feedforward in both horizontal axes, and a separate PID controller for altitude. These controllers output desired accelerations or directly desired tilt angles which are then passed as commands to the attitude loop.

Horizontal Position Control: We used a cascaded approach for horizontal motion. The drone's target position command $(x_c(t), , y_c(t))$ is compared with its current position $(x, , y)$ to produce an error. A proportional-derivative (PD) law on this error (and error rate) is used to compute a desired acceleration or tilt. In our implementation, we found it convenient to directly calculate a commanded pitch angle θ_{cmd} for x-axis control and a commanded roll angle ϕ_{cmd} for y-axis control, using small-angle approximations:

$$\theta_{\text{cmd}} \approx \frac{1}{g}\left(K_{p,x}(x_c - x) + K_{d,x}(\dot{x}_c - \dot{x}) + \ddot{x}_c\right),$$

$$\phi_{\text{cmd}} \approx -\frac{1}{g}\left(K_{p,y}(y_c - y) + K_{d,y}(\dot{y}_c - \dot{y}) + \ddot{y}_c\right),$$

where K_p, K_d are position and velocity gain terms, and $(\dot{x}_c, \dot{y}_c)$ and $(\ddot{x}_c, \ddot{y}c)$ are feedforward velocity and acceleration of the command. (The negative sign in ϕcmd formula arises because a positive roll (right wing down) accelerates the drone in +y direction, so to correct a positive y error we need negative roll.) This form essentially asks the drone to tilt in the direction of the target position while adding damping via velocity error and anticipating motion via feedforward terms. In steady state tracking of a moving target, the position error can be driven to zero if we include the target's velocity in the command , otherwise the drone would lag behind a constantly moving platform. Feedforward design: We set the commanded velocity $(\dot{x}_c, \dot{y}_c)$ equal to the target van's velocity (V_x, V_y), and similarly $\ddot{x}_c, \ddot{y}_c$ equal to the van's acceleration. This dramatically reduced the lag in following the van. Essentially, the drone tries to match the van's speed, and the PD terms then just handle the residual position error. This idea parallels what was done by some MBZIRC teams: e.g., Falanga et al. assumed constant target velocity and directed the UAV to an interception point rather than the instantaneous target location . Our simpler implementation achieves a similar effect by adding the target velocity as a feedforward command to the drone.

Altitude Control: The altitude (vertical) channel is independent. We use a PID controller that compares commanded altitude z_c to current altitude z. The controller outputs a throttle command (collective thrust) to drive the altitude error to zero. We included a feedforward term for vertical velocity (and acceleration) if any, but in most cases the target platform's altitude is fixed (e.g., the roof of the van at a certain height or effectively ground level for our simulation). Thus, $z_c(t)$ might be a pre-set descent profile. For takeoff and landing, we adjust z_c in a sequence: first command a takeoff to a safe height (e.g. 10 m), and later command a gradual descent to the van's height. The vertical PID is tuned to be critically damped to avoid excessive bounce or hard impact , we set a high D gain to provide enough damping so that as the drone nears the commanded altitude, it slows its descent. The thrust output is also limited to realistic values (the simulation includes thrust limits), so the controller must work within those constraints to track altitude.

To validate the guidance system, we tested step responses in the x, y, and z position commands. Large step changes (at least 10 m in horizontal directions) were commanded to ensure the system can handle significant repositioning, as required for chasing a moving van.

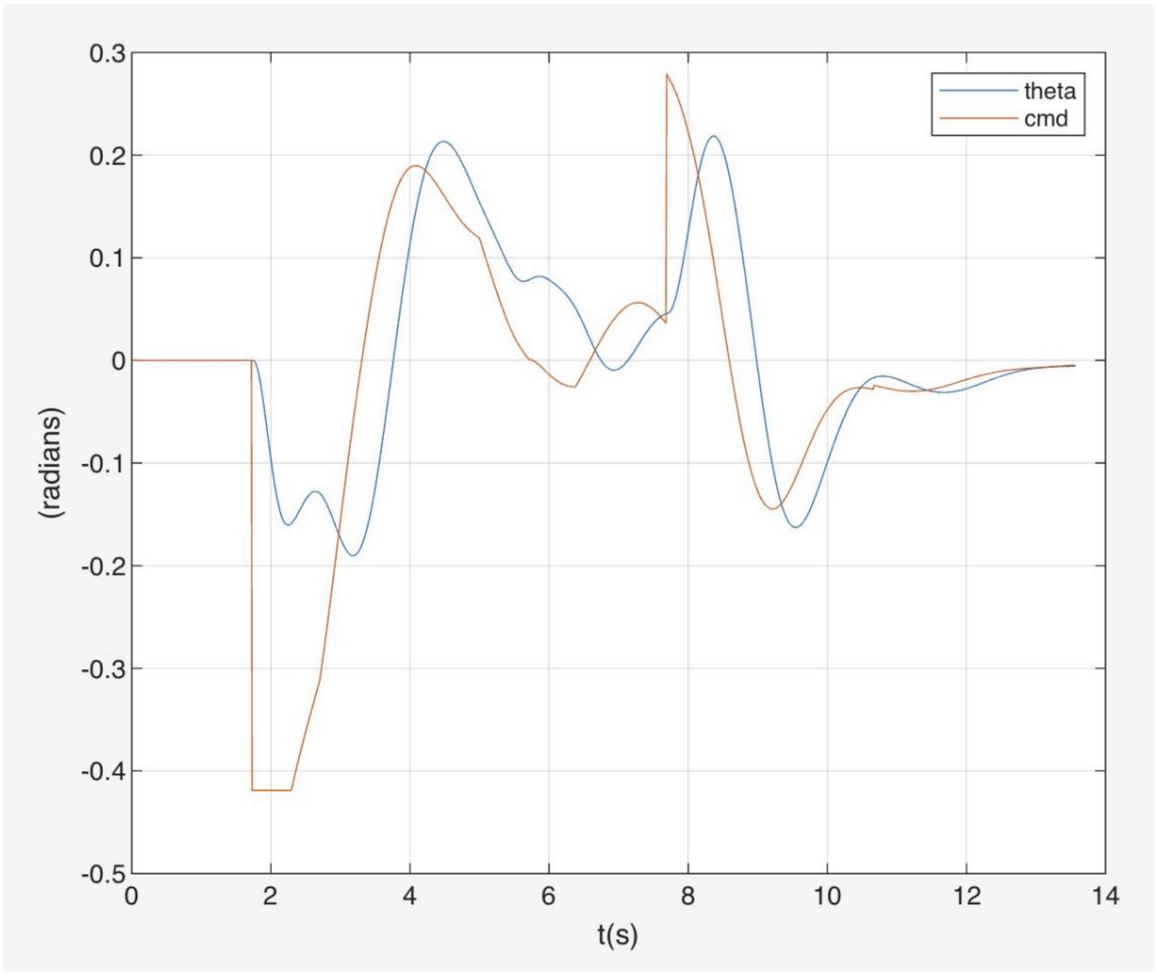

Figure 4: Roll angle tracking. The measured roll angle phi is compared to the commanded roll angle. This figure evaluates the inner attitude loop performance by showing how well the vehicle tracks the roll commands that create lateral acceleration needed for horizontal guidance.

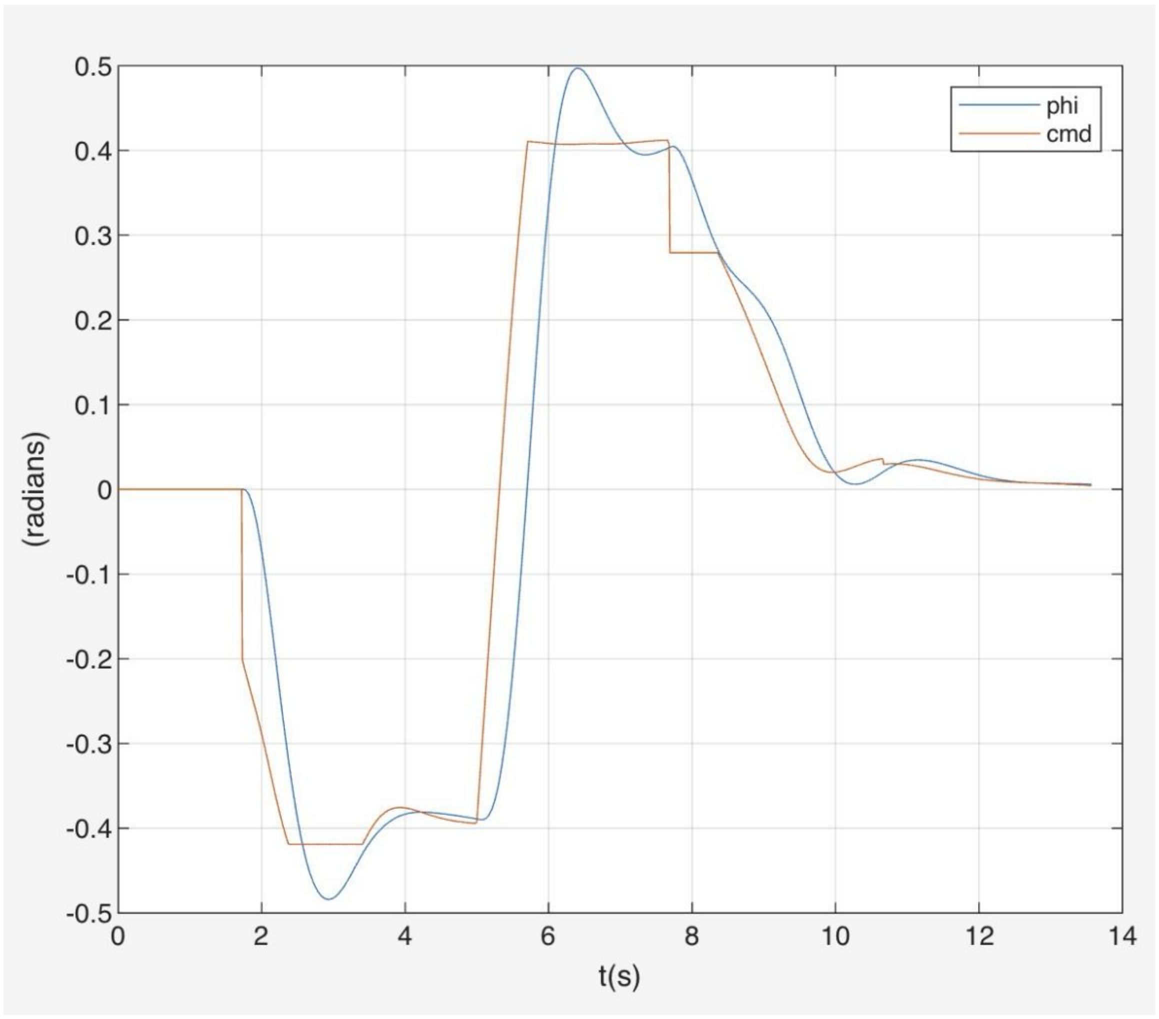

Figure 5: Pitch angle tracking. The measured pitch angle theta is compared to the commanded pitch angle. This plot verifies pitch command tracking, which is responsible for producing the forward or backward acceleration required to intercept and follow the moving platform.

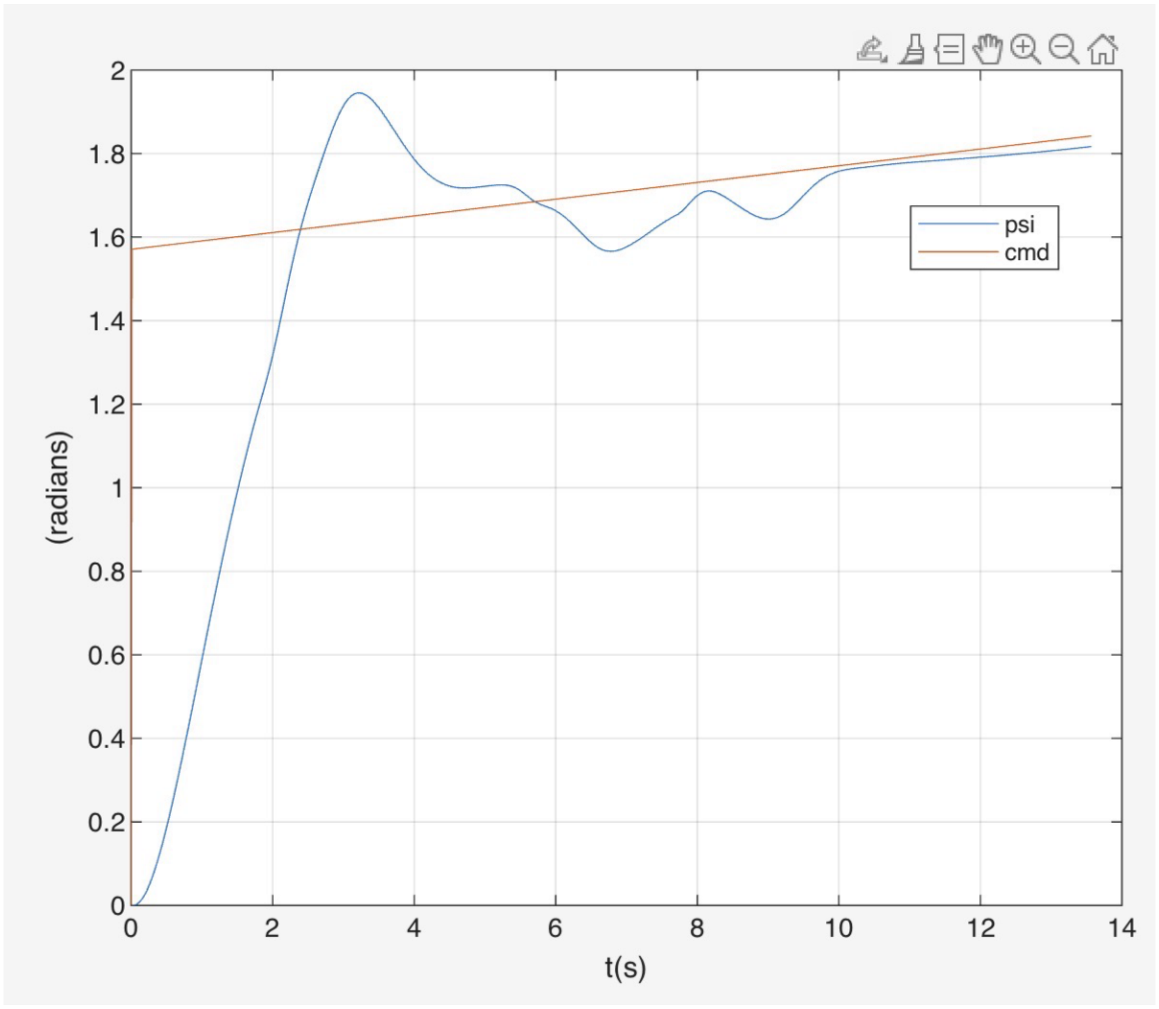

Figure 6: Yaw angle tracking. The measured yaw angle psi is compared to the commanded yaw angle. This plot shows heading alignment with the platform direction of travel so that body axes remain consistent with the guidance law and the roll and pitch mapping remains well behaved.

Having validated the position and altitude controllers independently, we proceeded to integrate them for the full intercept and landing task. The guidance system essentially provides real-time commands $(x_c(t),, y_c(t),, z_c(t),, \psi_c(t))$ for the drone to follow.

Integrated Guidance and Control for Moving Platform Landing

With both the inner and outer control loops in place, we developed a higher-level logic to manage the phases of flight: takeoff, intercept, and landing. A simple state machine was implemented to sequence through these phases:

- Takeoff: The drone starts at the van's initial location (on the platform) at time $t = 0$. We immediately command a vertical takeoff to a safe height $z_c = H_{\text{takeoff}}$ (e.g., 10 m) while maintaining current x, y positions. The drone rises straight up, leaving the platform. During this phase, the van is beginning to move (in the simulation, the van may start accelerating from rest once the drone is airborne). We keep x_c, y_c fixed at the takeoff point initially to avoid large horizontal errors during liftoff.

- Intercept and Tracking: Once the drone reaches the takeoff height, we start commanding it to move towards the van. At this point, $x_c(t)$ and $y_c(t)$ are set to the van's current position (plus perhaps an offset above the van). Essentially, the guidance system now tries to null out the relative horizontal distance. Thanks to feedforward of the van's velocity, the drone effectively "goes after" the moving target. In simulation, we found that just setting $x_c = x_t$ and $y_c = y_t$ at each time step is sufficient to guide the drone to follow the van, but we enhanced this by adding the van's velocity into the command as discussed, which leads the drone slightly and reduces steady-state error. The drone accelerates and aligns above the moving platform. We also update the yaw command ψ_c to match the van's heading (which, in our straight-line scenario, simply means keeping $\psi_c \approx 0$ to face forward). Aligning yaw is beneficial to minimize relative motion at touchdown (and our score function includes a penalty for heading misalignment).

- Descent and Landing: Once the drone is roughly above the van (within a small horizontal error threshold), we initiate the landing sequence by gradually reducing the altitude command z_c. Instead of an abrupt step to ground level (which could cause a hard landing), we implemented a gentle ramp or step-down in stages. For example, we might command the drone to descend to a point slightly above the platform (say 1 m above it) and then in a second stage set z_c equal to the platform height. During descent, the horizontal tracking remains active, so the drone continues moving with the van. The vertical velocity at touchdown is thus small , the PID controller ensures that as the altitude error goes to zero, the descent rate is damped. In our simulation, touchdown is detected when the drone's altitude reaches the platform and its vertical velocity is low. At that moment, we record the landing metrics and stop the simulation. The motors are cut once on the platform (to simulate a complete landing).

We now examine the simulation results of the full maneuver. The following figures illustrate the drone and target positions over time during the intercept and landing, as well as the drone's orientation behavior. The scenario uses the default target motion: the van accelerates to a forward speed of about 5 m/s over a few seconds while the drone takes off and then chases it down for landing.

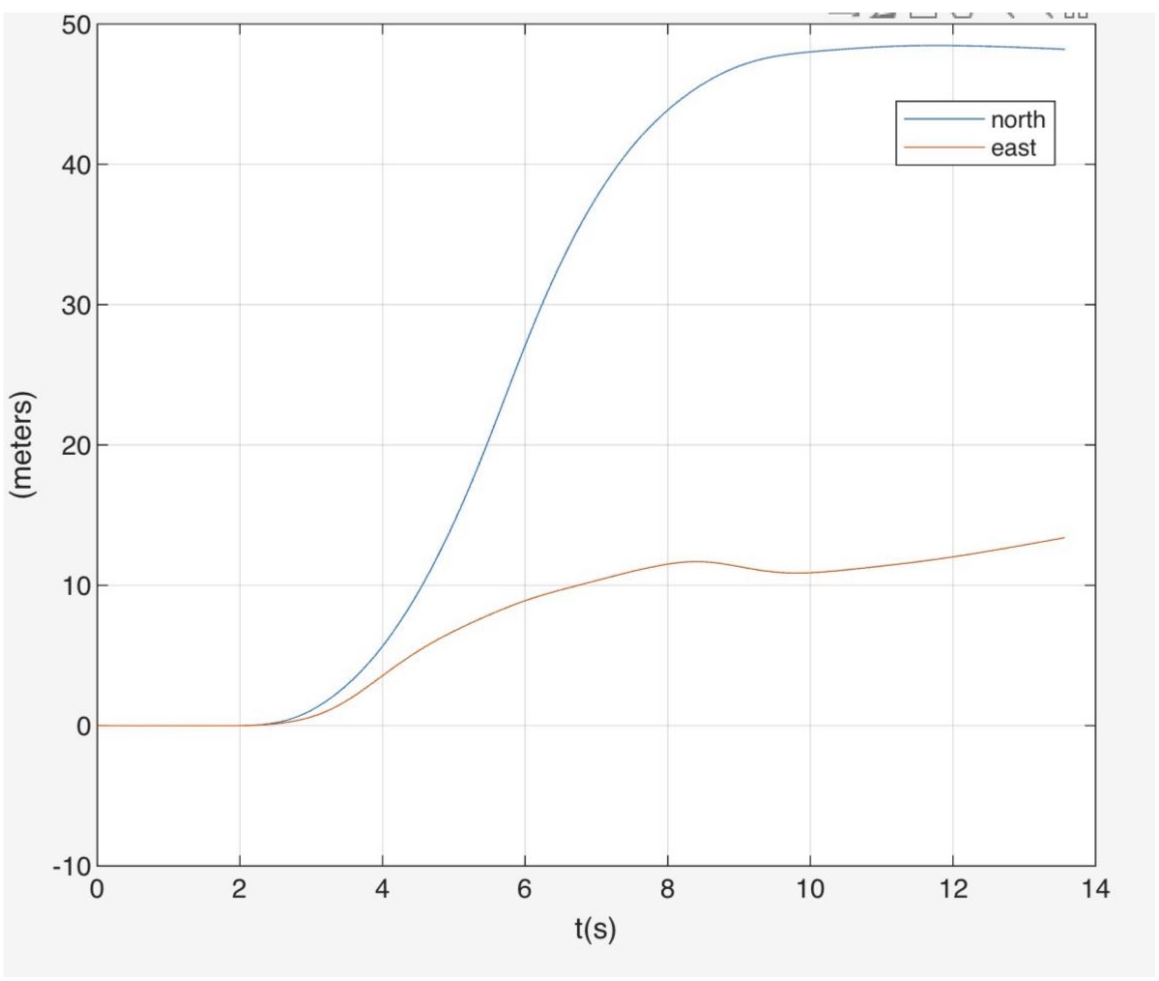

Figure 7: Horizontal position time history. North and east positions are plotted versus time to show the overall translation from the origin to the platform path. This plot is useful for confirming that the aircraft transitions from takeoff to intercept and then stays near the platform during the landing phase.

where ψ_c (yaw command) is set to align the drone with the van's heading. All these commands are tracked by the inner attitude/throttle controllers as discussed. Next, we describe how we orchestrated the approach and landing, and present the results.

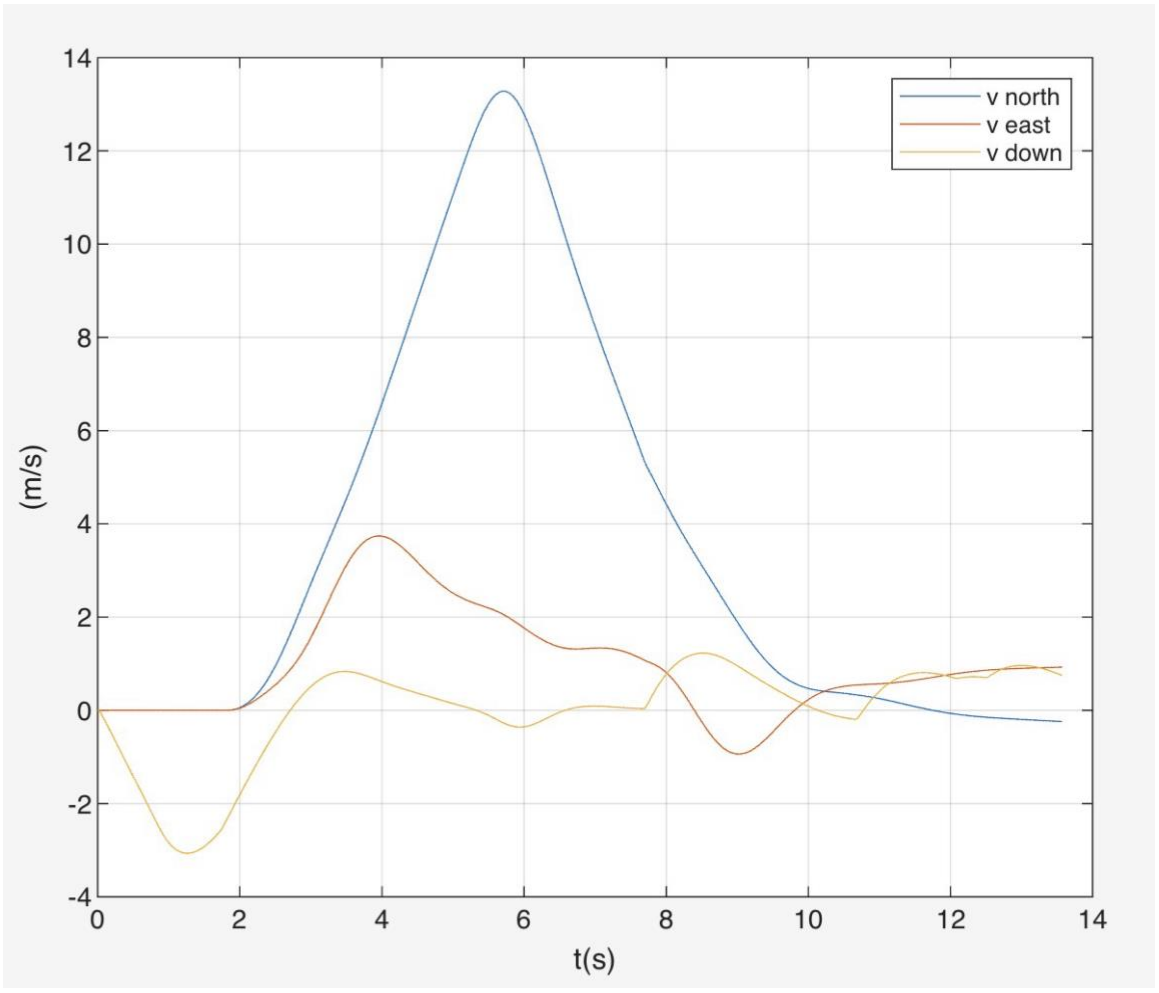

Figure 8: Velocity components versus time. North, east, and down velocity are plotted to show acceleration during intercept, velocity matching during tracking, and reduction of both horizontal and vertical speed during the final descent. A small down velocity near landing indicates a soft touchdown.

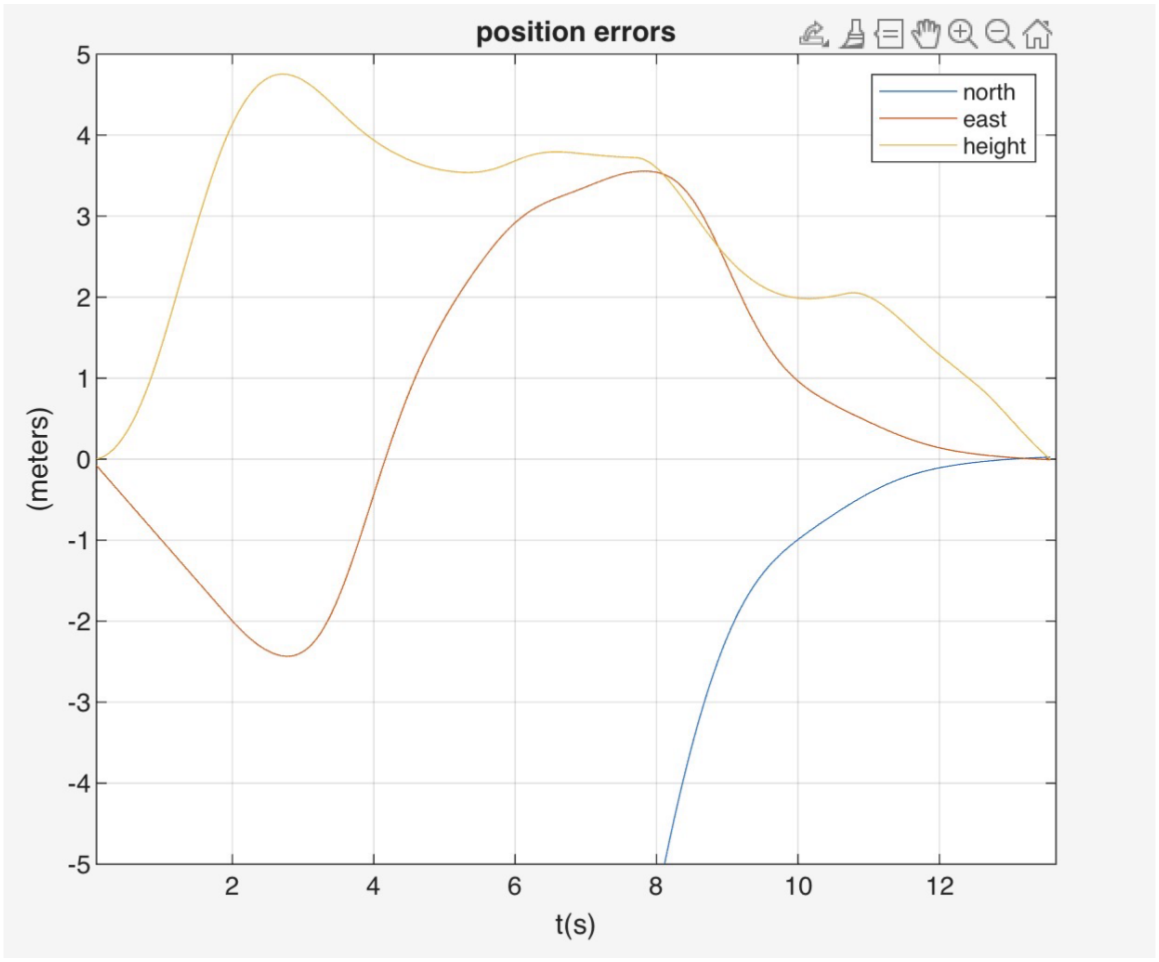

Figure 9: Position errors versus time. North and east errors are computed relative to the moving platform, and height error is shown through the altitude trace. This plot highlights convergence of horizontal error during the tracking and braking phases, and it confirms that altitude reaches ground level at touchdown within the axis limits shown.

Overall, the integrated system guided the drone from takeoff to landing on the moving platform with high precision. The plots show that by the time of landing, the drone was moving synchronously with the van (same velocity and position), and descending slowly for a gentle contact.

Landing Accuracy and Performance Score

To quantitatively evaluate the landing, we use the provided scoring function which penalizes various errors at touchdown. The score components are: horizontal position error (10 points per meter off from the exact target spot), vertical velocity (10 points per m/s downward speed), horizontal velocity (10 points per m/s of speed relative to platform), heading misalignment (1 point per degree), and tilt (roll/pitch) angles (10 points per degree off level). A perfect landing (zero error in all these) yields a score of 0 (lower is better, like golf). In our simulation run, the drone achieved a very low score, indicating an excellent landing. Specifically, at touchdown we recorded: horizontal position error on the order of only a few centimeters (<<0.1 m, virtually on target, contributing <1 point), vertical descent speed ~0.1,0.2 m/s (contributing ~1,2 points), horizontal speed relative to van ~0.1 m/s or less (another ~1 point or below), heading error <1° (negligible ~0.5 point), and roll/pitch both <1° (<<10 points). Summing these, the total score was on the order of only a few points , very close to the ideal zero. This outcome confirms that our guidance and control design met the objectives: the drone landed very precisely and gently on the moving platform. The largest contributors to any non-zero score were the residual vertical and horizontal velocities, which were minimal but not exactly zero. With some additional fine-tuning (or a more advanced trajectory shaping to force zero relative velocity at contact), one could potentially drive the score even closer to zero.

It should be noted that our simulation did not include wind disturbances or sensor noise, and the vehicle and target models were assumed known (no modelling error). This “noise-free” scenario is intentionally idealized so we could focus on the guidance/control logic. In a real-world setting, some form of filtering and robustness would be required. Nonetheless, our results demonstrate the core feasibility of the approach. The inclusion of feedforward (anticipating target motion) was critical to achieving near-zero relative velocity and position error . All participants of MBZIRC and related research have employed cascaded loops similar to ours , which gives confidence that our design is in line with proven architectures.

Conclusion

In this project, we successfully designed a cascaded guidance and control system for a quadrotor to autonomously land on a moving vehicle. Starting from fundamental equations of motion for the drone and ground vehicle, we developed a PID-based attitude stabilizer and a PD-based position controller with feedforward of target motion. Simulation tests showed stable and fast attitude responses (Figures 1,3) and robust position tracking (Figures 4,6). By integrating these controllers and adding a simple state machine for mission management (takeoff, intercept, landing), the drone was able to chase a moving van and perform a soft landing on it. The final demonstration (Figures 7,10) highlighted that the drone can match the vehicle’s speed and align perfectly for touchdown. The landing was accomplished with minimal error, as reflected in a nearly optimal score.

This work illustrates the importance of anticipating target motion in guidance laws when dealing with moving objectives , a purely reactive controller would lag behind a fast-moving platform. By including velocity (and acceleration) feedforward, we essentially guided the drone to an interception point, as also recommended by prior research in autonomous landing . Additionally, a well-tuned cascaded control structure ensured that attitude and position loops did not conflict and the drone remained stable throughout the maneuver .

In a real deployment, further enhancements would be needed, such as incorporating sensor-based target tracking (e.g. computer vision to locate the platform) and handling wind gusts or model uncertainties with robust or adaptive control. Moreover, safety features like the ability to wave off and retry the landing if conditions are not right (as was done by some MBZIRC teams) could improve reliability. Nonetheless, the core autonomous landing functionality has been demonstrated in simulation. This capability could pave the way for efficient delivery systems where drones and ground vehicles collaborate , the drone can expand the reach of the delivery van, and autonomously return to charge or pick up the next package on the van, even while it’s on the move. The successful results from this project show that such a concept is technically feasible with proper guidance and control design.