MPC-Based Tracking and Autonomous Landing of Quadrotor on Moving Ground Vehicle

Cornelius Gruss, Benjamin Hsu, Yancheng Zhou

***Abstract*—Autonomous landing on a moving ground platform breaks the usual static-landing-zone assumption, yet is needed for scenarios from maritime recovery to last-mile delivery. We present a linear model predictive control (MPC) framework for tracking and landing a small quadrotor on a moving ground vehicle. The drone's translational motion is modelled as a double integrator augmented with a vertical-acceleration disturbance state to cancel steady-state errors, so each control step reduces to a convex quadratic program solved at $50\,\mathrm{Hz}$. A guidance layer forecasts the platform's trajectory and gates descent until the drone has closed the lateral gap. We deploy the controller on a Crazyflie 2.1 tracking an AgileX LIMO ground robot under OptiTrack feedback across stationary, straight-line, circular, and erratic teleoperated motion. Landing succeeds in all four cases. Lateral tracking error grows from $0.10\,\mathrm{m}$ to $0.95\,\mathrm{m}$ with motion complexity, while altitude error stays within $0.24\,\mathrm{m}$ to $0.29\,\mathrm{m}$.**

I. Introduction

Unmanned aerial vehicles (UAVs) have become a necessary tool in a wide range of applications and domains, from logistics to emergency response and military use. Even as these systems move from human-operated to fully autonomous, the UAV still must land eventually. It is usually assumed that the landing zone is static, but this assumption limits the environments in which autonomous UAVs can operate. In many real-world scenarios, especially maritime ones, the landing zone is not static. For instance, a reconnaissance drone returning to a naval ship [1] or a delivery drone intercepting a logistics vehicle [2] both involve a moving landing target. The mobile-landing-platform literature has grown rapidly over the past decade and is surveyed by Pranjić [3]. Compared to a static landing scenario, these cases require the UAV to track and predict the landing target's trajectory while simultaneously managing a controlled descent onto the landing surface. Our project aims to be a small-scale proof of concept that demonstrates this capability (Fig. 1).

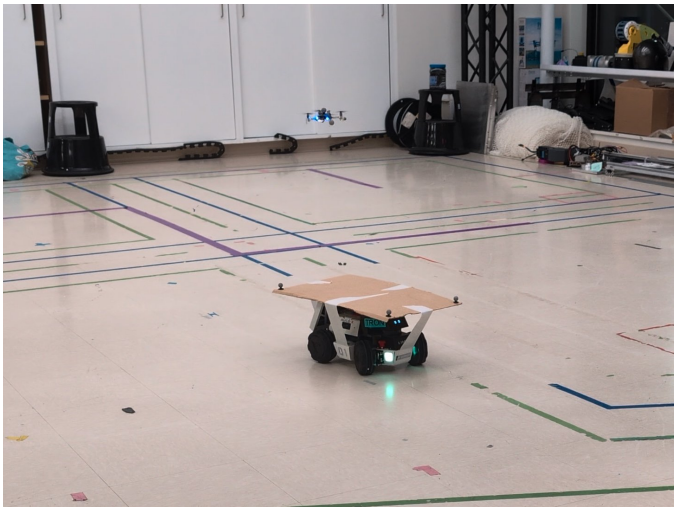

Fig. 1. Experimental setup: a Crazyflie 2.1 quadrotor tracking an AgileX LIMO ground robot carrying a cardboard landing pad, inside an OptiTrack motion-capture arena.

A. Related Work

The past decade has seen extensive work on both fixed and mobile landing platforms for UAVs. Pranjić [3] surveys this literature along the axes of platform motion, sensing modality, and control architecture. On the mobile-platform branch most relevant here, MPC has emerged as a dominant control approach because it can handle the coupled position, velocity, and actuation constraints of the landing problem within a single optimization.

Paris et al. [4] combine an MPC-based trajectory planner with a sliding-mode tracking controller to land a quadrotor on a moving platform under turbulent wind. Their planner does not require hovering over the platform before descent, which shortens total landing time at the cost of requiring simultaneous lateral tracking and vertical descent. Their experiments use a straight-line platform trajectory. Batool et al. [5] instead solve a nonlinear MPC augmented with a control-barrier function onboard the drone, producing collision-safe landing trajectories in the presence of an obstacle, again on a straight-line platform.

Our work is closest in spirit to Paris et al., in that a single MPC plans the entire tracking-and-descent trajectory. We differ in two ways. First, we adopt a *linear* double-integrator model with an additive vertical-acceleration disturbance state, which reduces each step to a convex QP solvable off-the-shelf, at the cost of requiring small-tilt operation. Second, our reference generator uses a constant-turn-rate, constant-velocity (CTRV) predictor so the MPC can preview circular platform trajectories, not only straight lines.

B. Problem Statement

We want a small quadrotor to autonomously track and land on a ground vehicle whose path is not known in advance. The drone must hold its horizontal position above the platform, descend in a controlled way, and stay within limits on thrust, tilt, and velocity. It must also remain robust as the platform's motion ranges from stationary to erratic.

The hardware is a Crazyflie 2.1 quadrotor, an AgileX LIMO differential-drive ground robot carrying a landing pad, and an

OptiTrack motion-capture system in an indoor arena. Within this setup, the project goals are to:

- formulate a real-time convex MPC that runs at $50\,\mathrm{Hz}$ on a ground computer and commands the Crazyflie through its stock attitude-and-thrust interface.
- generate reference trajectories that switch between hover, tracking, and descent while rejecting steady-state errors such as battery-voltage drift.
- achieve reliable touchdown across four platform-motion regimes (stationary, straight-line, circular, and erratic teleoperated motion) and report the resulting tracking errors.

Outdoor flight, onboard visual-inertial state estimation, and nonlinear MPC are out of scope.

C. Contributions

This paper makes the following contributions:

- A linear MPC for tracking and landing on a moving ground target, in which the standard offset-free MPC technique [6] (augmenting the model with a constant disturbance state, updated from the prediction residual at each tick) is applied to compensate for the dominant steady-state error in our hardware: battery-voltage drift, which monotonically reduces thrust per PWM unit as the Crazyflie discharges.
- A two-stage *guidance* layer for the MPC reference: a constant-turn-rate, constant-velocity (CTRV) predictor anticipates the target's xz trajectory during tracking, and a cone-gated landing reference forces the drone to close the lateral gap before descending.
- A hardware deployment on a Crazyflie 2.1 using OptiTrack feedback over MQTT (Message Queuing Telemetry Transport, a lightweight publish–subscribe protocol), with a boundary supervisor that disarms the motors on arena violation or pose-stream loss, and a separate yaw P controller that aligns the drone's heading with the platform's.
- An experimental evaluation across four platform-motion regimes: stationary, straight-line, circular, and teleoperated erratic lateral motion.

II. System Overview

An OptiTrack motion-capture system streams 6-DOF rigid-body pose estimates of both the quadrotor and the landing platform to a ground computer over an MQTT broker at $100\,\mathrm{Hz}$. A finite-difference velocity estimator reconstructs translational velocities from the position stream, yielding the full state $(p,\, v) \in \mathrm{R}^6$ for each tracked body. At $50\,\mathrm{Hz}$, the outer control loop runs in three stages. First, a *guidance* layer constructs the reference trajectory x_{ref} over the prediction horizon from the latest drone and platform poses, switching between hover, target-tracking (at $0.5\,\mathrm{m}$ above the platform), and descent depending on the operating mode. Second, an offset-free linear MPC solves a finite-horizon quadratic program to produce the desired world-frame acceleration $u = [a_x,\, a_y,\, a_z]^\top$. Third, an actuation layer converts that acceleration into the low-level roll/pitch/thrust setpoint expected by the Crazyflie firmware, using the drone's measured yaw to rotate lateral accelerations into the body frame. A separate proportional yaw controller runs alongside the position MPC, slaving the drone heading to the platform heading so that the small-angle linearization used to map acceleration into tilt angles remains valid. The resulting attitude setpoint is dispatched through a boundary supervisor that disarms the motors on arena violation or on $> 2\,\mathrm{s}$ of pose-stream loss, before reaching the drone. The full pipeline is summarised in Fig. 2.

The architecture reflects four deliberate design choices. First, the guidance layer is separated from the MPC core so that the controller is mode-agnostic: the MPC tracks whatever reference it is handed, and the entire flight-mode state machine (hover, tracking, descent) lives in the reference itself. This makes adding new modes a matter of writing a new reference generator, never a new controller. Second, yaw is decoupled from the position MPC because the small-angle approximation used to convert lateral acceleration into tilt requires the drone's heading to track the platform's; coupling yaw into the MPC would require either a nonlinear model or repeated re-linearization, neither of which the linear-QP formulation supports. Third, MQTT is chosen as the sensing backbone because it allows the drone-pose stream, the platform-pose stream, and the battery telemetry to be published independently and consumed by multiple processes (controller, dashboard, logger) without coupling them. Fourth, the boundary supervisor sits in front of the radio rather than inside the MPC so that it intercepts every command, including ones bypassing the MPC during takeoff and emergency abort.

A. Linearization of Quadrotor Dynamics

Although nonlinear MPC formulations for quadrotors are routinely solved in real time on modern hardware [7], a linear formulation suffices for the slow tracking and controlled-descent regime of this project (quantified below) and reduces the problem to a convex quadratic program solvable off-the-shelf. We further adopt the standard *cascaded* assumption [8]: the onboard attitude loop realises commanded tilts nearly instantaneously relative to the outer loop, so the outer loop sees only the commanded acceleration as its control authority. The translational dynamics then reduce to a 3D double integrator with an affine gravity term,

$$x[t+1] = A\,x[t] + B\,u[t] + g, \qquad u[t] = [a_x, a_y, a_z]^\top. \quad (1)$$

We adopt the motion-capture convention with $+Y$ pointing up, so that p_y is the altitude and a_y is the vertical acceleration. Gravity enters as the constant affine term g rather than as a nonlinear coupling, and the equilibrium input $u_{\mathrm{eq}} = [0,\, 9.81,\, 0]^\top$ corresponds to hover, i.e. the input cost $\|u - u_{\mathrm{eq}}\|_R^2$ is zero in steady state.

The stock Crazyflie firmware used in this work exposes an attitude-and-thrust interface (roll, pitch, yaw-rate, thrust) via the cflib library, rather than a direct acceleration setpoint. The commanded acceleration produced by the MPC is therefore converted to an attitude-and-thrust setpoint on the ground computer before being dispatched over the radio. The conversion

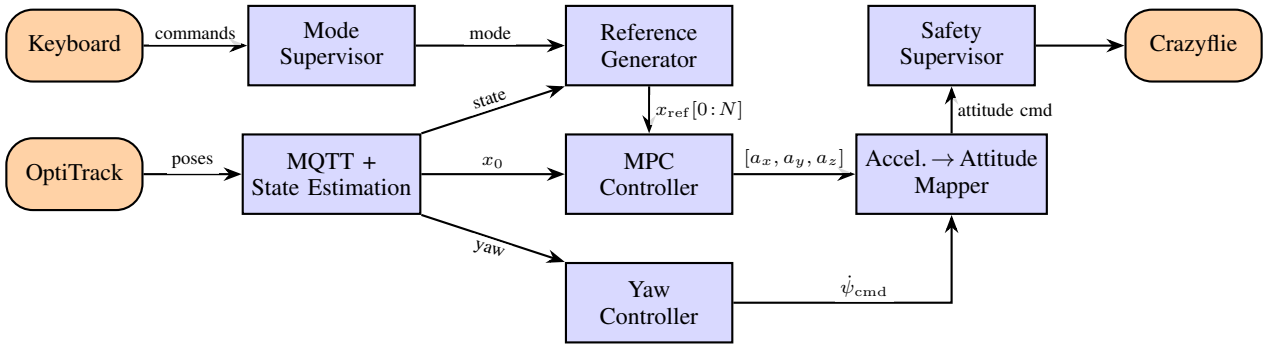

Fig. 2. Control system architecture. External hardware interfaces are orange. Control components are blue.

rests on the small-angle approximation. Denote pitch θ as the forward-tilt angle (positive when the thrust vector leans forward) and roll ϕ as the lateral-tilt angle (positive when the thrust vector leans in the $+z$ direction). For a quadrotor with mass-normalised thrust T/m in a yaw-zero body frame,

$$a \;\approx\; \begin{bmatrix} g\,\theta \\ T/m - g \\ g\,\phi \end{bmatrix} + \mathcal{O}(\theta^2,\, \phi^2), \tag{2}$$

which can be inverted to recover attitude and thrust setpoints from the MPC output,

$$\theta \;\approx\; \frac{a_x}{g}, \qquad \phi \;\approx\; \frac{a_z}{g}, \qquad \frac{T}{m} \;\approx\; g + a_y. \tag{3}$$

In the hardware implementation, the horizontal pair (a_x, a_z) is first rotated into the body frame using the drone's measured yaw before being mapped into pitch/roll. The decoupled yaw P-controller keeps the yaw error small so that the linearization around $\psi = 0$ remains accurate.

Validity of the linearization.: The small-angle approximation $\sin\alpha \approx \alpha$ incurs an error of order $\alpha^3/6$. With the lateral acceleration box $|a_{x,z}| \le a_{\max}$ clipped in software to a 15° maximum tilt, the acceleration-prediction error is bounded by $g\alpha^3/6 \approx 0.03\,\mathrm{m\,s^{-2}}$, or roughly $0.3\,\%$ of g, comfortably smaller than the dominant sources of steady-state mismatch (battery-voltage drift, thrust calibration, and downwash near the moving platform), which is why the offset-free disturbance state introduced in the next subsection targets those first-order effects rather than the linearization residual itself. The observed peak tilts during tracking experiments are well under the 15° bound.

B. Trajectory Planning: Model Predictive Control

The control problem is formulated as a finite-horizon optimal control problem shown in equation (4):

$$\begin{aligned} \min_{\{x[t]\},\,\{u[t]\}} \quad & \sum_{t=0}^{N-1} \Big[(x[t] - x_{\mathrm{ref}}[t])^\top Q (x[t] - x_{\mathrm{ref}}[t]) \\ & \qquad + (u[t] - u_{\mathrm{eq}})^\top R (u[t] - u_{\mathrm{eq}}) \Big] \\ & + (x[N] - x_{\mathrm{ref}}[N])^\top Q_f (x[N] - x_{\mathrm{ref}}[N]) \\ \text{s.t.} \quad & x[t+1] = Ax[t] + Bu[t] + g, \\ & x[0] = x_0^{\mathrm{aug}}, \\ & -a_{\max}\mathbf{1} \le u[t] \le a_{\max}\mathbf{1}, \\ & -v_{\max} \le v[t] \le v_{\max}. \end{aligned} \tag{4}$$

In this optimal control problem, the state vector $x[t] \in \mathrm{R}^7$ represents the augmented system configuration at time step t. It consists of the quadrotor positions and velocities in Cartesian coordinates together with an additional disturbance state:

$$x[t] = [p_x[t],\, v_x[t],\, p_y[t],\, v_y[t],\, p_z[t],\, v_z[t],\, d[t]]^\top.$$

A drop in battery voltage reduces the actual lift generated by a given thrust command, resulting in a lower measured vertical velocity than predicted by the model. The resulting prediction error is used to update the disturbance estimate, so battery-induced thrust loss is captured as a negative vertical disturbance in $d[t]$.

The control input $u[t] \in \mathrm{R}^3$ denotes the commanded accelerations along each axis:

$$u[t] = [a_x[t],\, a_y[t],\, a_z[t]]^\top.$$

The matrices A and B define the discrete-time dynamics of the augmented system, in which the translational motion in each spatial direction follows a double-integrator model. The disturbance state evolves according to $d[t+1] = d[t]$ and enters the vertical dynamics as an additive acceleration bias. The affine term g accounts for gravity.

The equilibrium input u_{eq} shifts the control cost so that zero deviation $(u[t] = u_{\mathrm{eq}})$ corresponds to the hover condition, so that hover incurs no control-effort penalty despite gravity being present in the dynamics.

The initial condition x_0^{aug} is constructed by augmenting the measured state with the disturbance estimate.

The reference trajectory $x_{\mathrm{ref}}[t]$ specifies the desired positions and velocities over the prediction horizon, with the disturbance

component set to zero. The objective function penalizes deviations from this reference using the weighting matrices Q and Q_f for stage and terminal costs, respectively, and penalizes control effort relative to u_{eq} via R. In place of an explicit terminal-set constraint, we use a terminal-state cost Q_f weighted roughly an order of magnitude larger than the stage cost Q (Table I) as an approximate infinite-horizon penalty. This avoids the infeasibility a terminal set would introduce at the step changes in x_{ref} during mode transitions (hover $\rightarrow$ tracking $\rightarrow$ descent). It is worth flagging that this choice does *not* provide a formal stability guarantee in the standard sense [9]: recursive feasibility and asymptotic stability would require a terminal set together with a terminal cost that is a Lyapunov function on that set. The larger terminal weight is a heuristic substitute that proved empirically robust over our four hardware experiments.

The disturbance state is not penalized in the cost function, allowing it to adapt freely to compensate for steady-state model mismatch. In practice, the disturbance is estimated externally at each time step and incorporated into the augmented state.

The scalar bounds $a_{\max}$ and $v_{\max}$ impose constraints on accelerations and velocities, ensuring physically feasible motion and the validity of the small-angle approximation, and N is the prediction horizon length.

The problem in (4) is implemented as a stage-wise formulation in CVXPY with state and input trajectories as decision variables. Because the dynamics are linear and the cost is quadratic, this is equivalent to the standard condensed QP obtained by eliminating the state trajectory in favour of the stacked control sequence,

$$z = \begin{bmatrix} u[0]^\top & u[1]^\top & \cdots & u[N-1]^\top \end{bmatrix}^\top, z \in \mathrm{R}^{3N} \quad (5)$$

By recursively applying the system dynamics, the predicted state trajectory can be expressed as an affine function of the initial state and control sequence:

$$\bar{x} = \bar{T} x_0^{\mathrm{aug}} + \bar{S} z + \bar{d}. \quad (6)$$

Substituting this expression into the cost function yields a quadratic function in z of the form

$$J(z) = \frac{1}{2} z^\top H z + f^\top z + \mathrm{const}, \quad (7)$$

where $H \succeq 0$ and f depend on the system matrices, weighting matrices, reference trajectory, and equilibrium input.

The input and velocity constraints can similarly be written as linear inequalities in z:

$$Gz \leq W + S x_0^{\mathrm{aug}} + r. \quad (8)$$

Thus, the MPC problem is equivalently expressed as the quadratic program

$$\begin{aligned} \min_z \quad & \frac{1}{2} z^\top H z + f^\top z \\ \text{s.t.} \quad & Gz \leq W + S x_0^{\mathrm{aug}} + r. \end{aligned} \quad (9)$$

This QP is solved at each time step, and only the first control input is applied, resulting in a receding horizon implementation.

C. Yaw Control

Yaw is controlled outside the MPC via a proportional law:

$$\dot{\psi}_{\mathrm{cmd}} = -k_p \, \mathrm{wrap}(\psi_{\mathrm{ref}} - \psi), \quad (10)$$

where $\mathrm{wrap}(\cdot)$ maps the yaw error to $[-\pi, \pi]$, so the quadrotor takes the shortest-angle rotation to align with the platform.

D. Reference Generation

The MPC controller is agnostic to *what* trajectory it tracks. The reference $x_{\mathrm{ref}}[k]$ over the prediction horizon is produced by a separate guidance layer that switches between three modes based on the pilot's command and the latest drone and platform poses.

Hover.: A static reference places the drone at a fixed setpoint with zero velocity for all $k \in \{0, \ldots, N\}$. This mode is used for takeoff, for manual repositioning, and as a safe fallback if the landing-platform pose stream is lost.

Tracking.: During tracking, the reference is a prediction of the platform's trajectory over the horizon, offset vertically by a fixed $0.5\,\mathrm{m}$ so that the drone hovers above the pad. Rather than using a constant-velocity (CV) extrapolation, which systematically overshoots on curved paths, we use a CTRV model in the xz-plane,

$$\dot{p}_{xz}^{\mathrm{ref}}(t) = R_y(\omega t)\, v_0, \qquad \omega = \dot{\psi}_{\mathrm{platform}}, \quad (11)$$

where ω is the platform's measured yaw-rate. Integrating this model analytically yields the closed-form prediction

$$p_{xz}^{\mathrm{ref}}(t) = p_{xz,0} + \frac{1}{\omega} \begin{bmatrix} \sin \omega t & 1 - \cos \omega t \\ -(1 - \cos \omega t) & \sin \omega t \end{bmatrix} v_{xz,0}, \quad (12)$$

which degenerates gracefully to the CV model when $|\omega| < 10^{-3}\,\mathrm{rad/s}$. This predicted trajectory is then loaded directly into the lateral components of $x_{\mathrm{ref}}[k]$ for the MPC. The choice of CTRV over CV lets the MPC preview the lateral acceleration required to stay centred over a LIMO executing a circular trajectory, rather than perpetually lagging it.

Landing.: During descent, the same CTRV prediction is used for the xz-reference, but the altitude component is *gated* by an approach-cone membership test. Let $\mathcal{C}(\alpha,\, r_0)$ denote the truncated cone with apex-centred at the pad, opening upward with half-angle α and a base disk of radius r_0,

$$\begin{aligned} \mathcal{C}(\alpha, r_0) = \big\{\, p \in \mathrm{R}^3 \;:\; & \|(p_x, p_z) - (p_x^{\mathrm{pad}}, p_z^{\mathrm{pad}})\| \\ & \leq r_0 + (p_y - p_y^{\mathrm{pad}}) \tan \alpha \,\big\}. \end{aligned} \quad (13)$$

At each control tick, the altitude reference for the horizon is computed as

$$p_{y,\mathrm{ref}}[k] = \begin{cases} p_y^{\mathrm{ramp}}[k] & \text{if } p^{\mathrm{drone}} \in \mathcal{C}, \\ p_y^{\mathrm{drone}} & \text{otherwise}, \end{cases} \quad (14)$$

where $p_y^{\mathrm{ramp}}[k] = \max\big(p_y^{\mathrm{drone}} - v_d\, k\Delta t,\; p_y^{\mathrm{pad}}\big)$ is the descending ramp, with descent rate $v_d = 0.3\,\mathrm{m\,s^{-1}}$ and cone parameters $\alpha = 20^\circ$, $r_0 = 5\,\mathrm{cm}$. The descent rate is chosen empirically as a compromise between three constraints: it must be slow enough that the disturbance estimator has time to

absorb residual model mismatch as the drone enters ground-effect downwash near the pad, slow enough that any lateral tracking error remaining at touchdown does not translate into a hard impact, and fast enough that the entire descent fits within the few-second window before the battery-driven thrust margin degrades meaningfully; $0.3\,\mathrm{m\,s^{-1}}$ satisfies all three across our four motion regimes. The cone parameters are likewise empirical: $\alpha = 20^\circ$ is wide enough that small lateral oscillations during descent do not repeatedly trigger the freeze, and $r_0 = 5\,\mathrm{cm}$ matches the radius of the landing pad. The consequence of this construction is that whenever the drone drifts laterally outside the cone (as it will during the transient on a circular path or under an erratic target), the reference altitude freezes, so the optimiser sees zero altitude error and spends its control authority closing the lateral gap. Only once the drone re-enters the cone does the vertical ramp resume. Motor cutoff is triggered one step outside the MPC, when the drone's altitude falls within $5\,\mathrm{cm}$ of the pad and the drone is inside the cone.

The full set of MPC, disturbance-estimator, and yaw-controller parameter values used in both simulation and hardware experiments is summarised in Table I.

TABLE I. MPC and auxiliary-controller parameters used in simulation and hardware.

Parameter	Value
Control timestep Δt	$0.02\,\mathrm{s}$ ($50\,\mathrm{Hz}$)
Prediction horizon N	50 steps ($1.0\,\mathrm{s}$)
Stage state weights Q	$\mathrm{diag}(10,\ 1,\ 10,\ 1,\ 10,\ 1)$
Terminal state weights Q_f	$\mathrm{diag}(200,\ 20,\ 100,\ 10,\ 100,\ 10)$
Input weights R	$\mathrm{diag}(1,\ 1,\ 1)$
Equilibrium input u_{eq}	$[0,\ 9.81,\ 0]^\top\ \mathrm{m\,s^{-2}}$
Acceleration bound $a_{\max}$	$15\,\mathrm{m\,s^{-2}}$ (per axis)
Velocity bound $v_{\max}$	$2\,\mathrm{m\,s^{-1}}$ (per axis)
Disturbance filter gain α	0.2
Yaw P-gain k_p	2.0
Yaw-rate saturation	$60^\circ\,\mathrm{s^{-1}}$
QP solver	CVXPY / OSQP

III. Experimental Results

A. Simulations

Before flying on hardware, we tested the MPC formulation in the Crazyflow simulator (Fig. 3) using the same core controller, with only a coordinate-frame remapping (Crazyflow's Z-up vs. the MPC's Y-up) and an acceleration-to-attitude conversion as environment-specific code. A PyVista GUI with a draggable target let us exercise hover, tracking, and descent in closed loop. These runs validated the MPC and reference-generation logic and gave us a baseline tuning of Q, R, and horizon length before hardware deployment.

B. Hardware

Hardware experiments were run in the RASTIC indoor arena at Boston University. The three main pieces of hardware, shown in Fig. 4, are a Crazyflie 2.1 quadrotor, an OptiTrack motion-capture system running Motive, and an AgileX LIMO ground robot. The Crazyflie flies on stock firmware, commanded from a PC over a Crazyradio 2.0 dongle using cflib. The LIMO carries a rooftop landing pad attached with OptiTrack markers and is driven along user-defined trajectories via a UDP motion-command server.

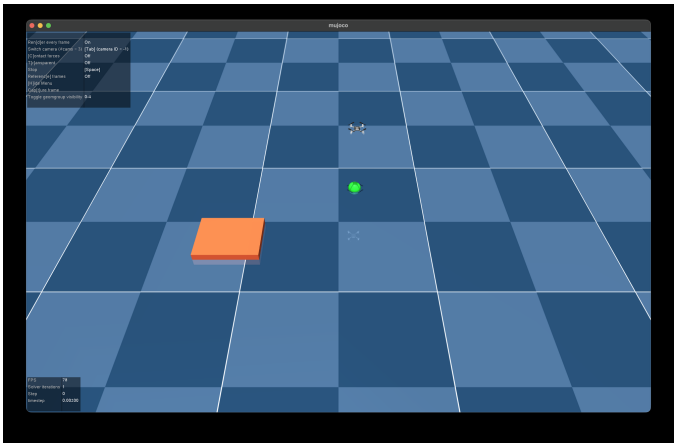

Fig. 3. Crazyflow simulation environment (MuJoCo-rendered) used for pre-hardware MPC tuning. The Crazyflie hovers above a landing pad (orange) while the user-draggable target (green sphere) defines the tracking reference fed to the guidance layer.

Pose data from Motive is published to an MQTT broker on the arena's local network, from which the PC control process ingests drone and platform streams. At $50\,\mathrm{Hz}$ it solves the MPC and dispatches the resulting roll/pitch/yaw-rate/thrust setpoint back to the Crazyflie over cflib.

Every flight writes a per-tick CSV log (drone state, MPC reference, commanded acceleration, attitude setpoint, solver status, battery voltage) plus event JSON logs for takeoffs, mode changes, and touchdowns. These logs are the source of the results below.

Four experiments were run, all sharing the same takeoff–hover–tracking–descent sequence and differing only in the LIMO's motion profile.

C. Stationary Landing Pad

1) Setup: The LIMO stayed still throughout the flight. This baseline removes any platform-motion prediction error and isolates the tracking-and-landing controller.

2) Results: Fig. 5 shows a typical stationary landing. Takeoff brings the drone from the floor to the $1\,\mathrm{m}$ hover height in about two seconds, with small lateral transients of around $0.29\,\mathrm{m}$ in x and $0.20\,\mathrm{m}$ in z that damp out within a few seconds. After that the drone holds position with oscillations under $0.05\,\mathrm{m}$ and follows the $-0.3\,\mathrm{m\,s^{-1}}$ descent ramp almost exactly. The resulting RMS errors ($0.10\,\mathrm{m}$, $0.24\,\mathrm{m}$, $0.08\,\mathrm{m}$, see Table II) are the smallest of any experiment, as expected when the pad does not move.

D. Straight-Line Trajectory

1) Setup: The LIMO drove along a straight line at constant speed. This is the simplest moving-platform case and tests

(a). Crazyflie 2.1 quadrotor.

(b). OptiTrack motion-capture system.

(c). LIMObot with a landing pad mounted

Fig. 4. The three main pieces of hardware used in the experiments.

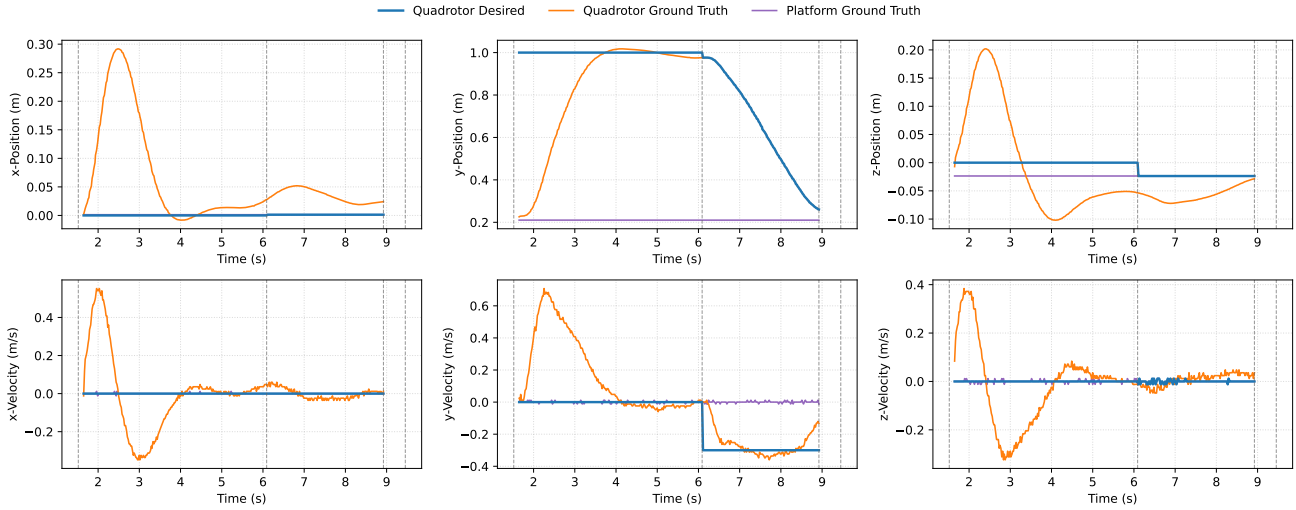

Fig. 5. Position and velocity tracking for the stationary landing experiment. Vertical dashed lines mark takeoff, entry to tracking, start of descent, and touchdown.

tracking of a constant-velocity target under simultaneous descent.

2) Results: Fig. 6 shows the straight-line case. The platform starts about $2.6\,\mathrm{m}$ behind and $1.1\,\mathrm{m}$ to the side of the drone, then accelerates to roughly $0.5\,\mathrm{m\,s^{-1}}$ in $+x$. At tracking entry the reference jumps onto the pad, so the drone closes the gap in one burst with x-velocity near $-1.1\,\mathrm{m\,s^{-1}}$ and z-velocity near $1.0\,\mathrm{m\,s^{-1}}$, catching up by $t \approx 9\,\mathrm{s}$ and then matching the platform's speed. The vertical step from $1.0\,\mathrm{m}$ to $0.7\,\mathrm{m}$ and the descent ramp to $0.25\,\mathrm{m}$ are both tracked cleanly. The RMS errors rise to $0.65\,\mathrm{m}$ in x and $0.37\,\mathrm{m}$ in z, driven mostly by the initial chase rather than steady-state error.

E. Circular Trajectory

1) Setup: The LIMO followed a smooth circular trajectory at constant angular velocity. The continuously changing heading and non-zero lateral acceleration are more demanding on both the position and yaw controllers than the straight-line case.

2) Results: Fig. 7 shows the circular case. Because the LIMO now drives around a circle, its x- and z-positions curve smoothly instead of stepping or ramping. The drone catches up with a $-1.5\,\mathrm{m\,s^{-1}}$ x-velocity swing at tracking entry, then locks onto the motion with a visible phase lag in z but matching amplitude. Altitude RMS is essentially unchanged ($0.28\,\mathrm{m}$) and lateral RMS grows only slightly ($0.68\,\mathrm{m}$ in x), but yaw RMS jumps to $15.4°$ because the drone's heading has to rotate continuously with the platform.

F. Erratic Lateral Motion

1) Setup: The LIMO was teleoperated to move left and right unpredictably, producing sudden velocity and heading changes that a constant-velocity predictor cannot anticipate. This stresses robustness when the assumed target dynamics are violated.

2) Results: Fig. 8 shows the erratic case, the hardest of the four. The platform drifts forward at roughly $0.5\,\mathrm{m\,s^{-1}}$ while the teleoperator swings it back and forth in z, and the drone follows the zig-zag with similar amplitude but a fraction of a second of delay, since the predictor cannot anticipate the reversals. The tracking-entry is also the largest, with x-velocity briefly reaching $-1.7\,\mathrm{m\,s^{-1}}$. The RMS errors peak here ($0.95\,\mathrm{m}$ in x, $25.6°$ in yaw, see Table II), but the closed loop stays stable and the drone still lands on the pad.

G. Summary

Table II collects the tracking-error RMS across all four experiments. Lateral (x) position error grows steadily with the complexity of the platform motion, from $0.100\,\mathrm{m}$ in the stationary case to $0.951\,\mathrm{m}$ in the erratic case. Altitude (y) error

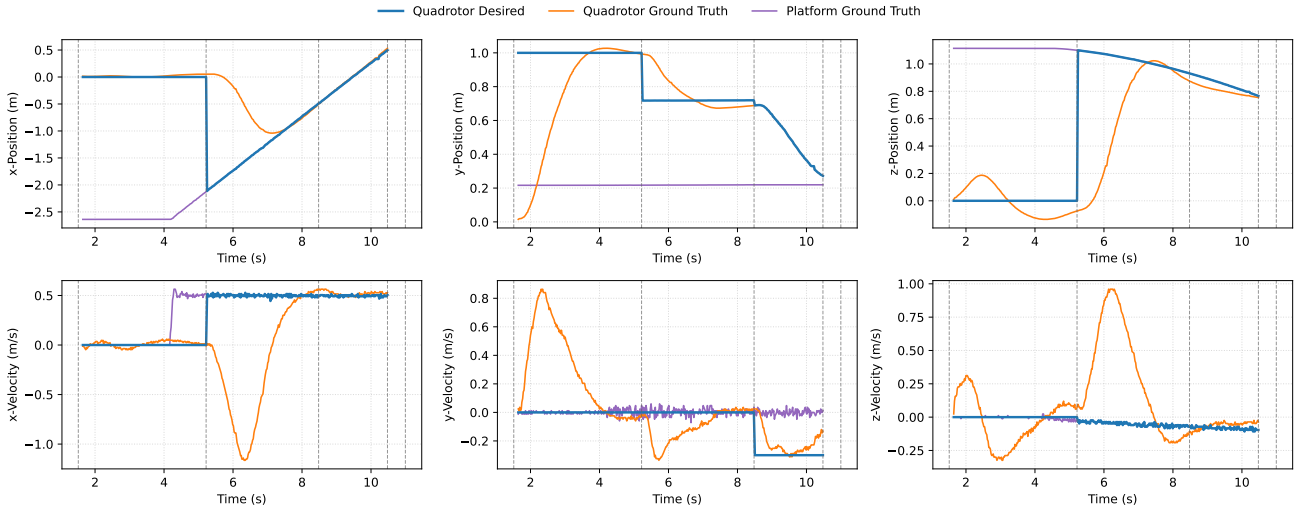

Fig. 6. As Fig. 5, but for the straight-line trajectory.

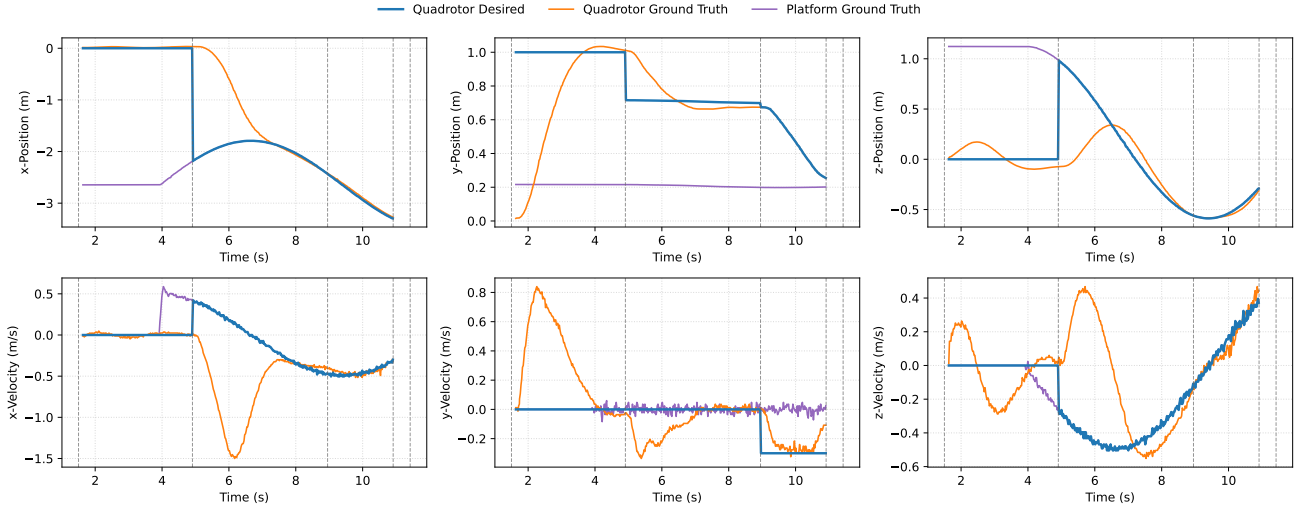

Fig. 7. As Fig. 5, but for the circular trajectory.

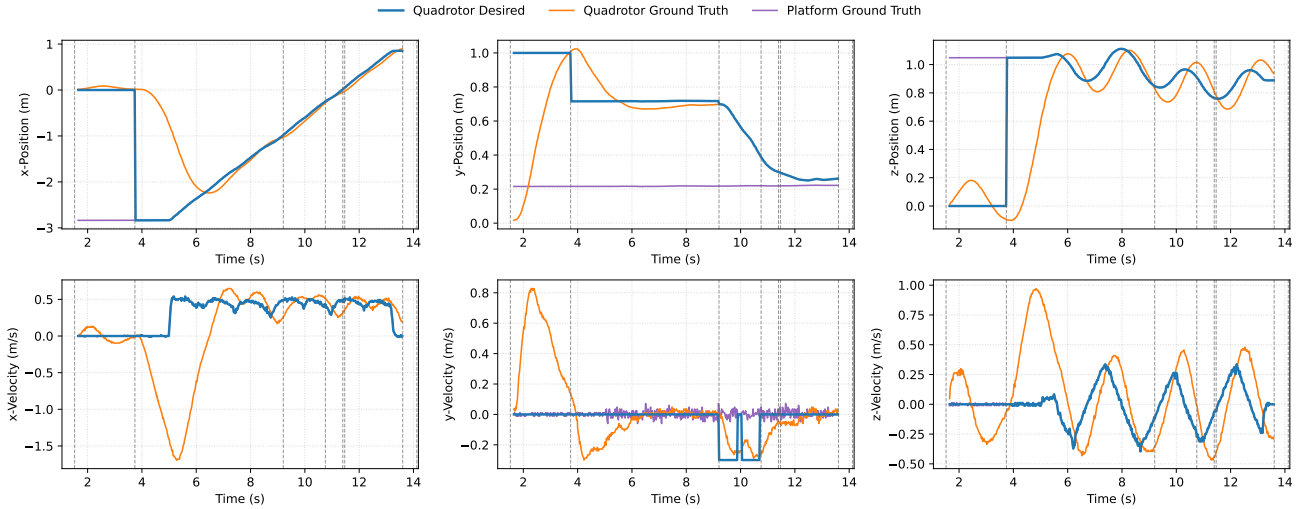

Fig. 8. As Fig. 5, but for the erratic lateral motion.

stays in a narrow band ($0.24\,\mathrm{m}$ to $0.29\,\mathrm{m}$) across scenarios, which is in line with the offset-free vertical disturbance estimator and the cone-gated descent logic holding altitude until the lateral gap is closed. Yaw error is highest in the circular and erratic cases, where the platform heading keeps changing throughout the experiment.

IV. Conclusion

This paper developed a linear MPC with a landing reference to allow a quadrotor to track and land on a moving ground platform, and demonstrated its effectiveness across four hardware experiments of increasing complexity: stationary, straight-line, circular, and erratic lateral platform motion. Future work includes blending the reference trajectories between hover, tracking, and descent modes so that the reference no longer steps at each mode change, scaling the approach up to a 5-inch quadrotor landing on the back of a pickup truck for outdoor experiments, and replacing the OptiTrack feedback with visual-inertial odometry (VIO) for onboard-only state estimation.

References

[1] S. Abujoub, J. McPhee, C. Westin, and R. A. Irani, "Unmanned aerial vehicle landing on maritime vessels using signal prediction of the ship motion," in *OCEANS 2018 MTS/IEEE Charleston*. IEEE, 2018, pp. 1–9.

[2] Y. Scherr, B. Neumann Saavedra, M. Hewitt, and D. Mattfeld, "Service network design with mixed autonomous fleets," *Transportation Research Part E: Logistics and Transportation Review*, vol. 124, pp. 40–55, 2019.

[3] A. Pranjić, "A decade of UAV docking stations: A brief overview of mobile and fixed landing platforms," *Drones*, vol. 6, no. 1, p. 17, 2022.

[4] A. Paris, B. T. Lopez, and J. P. How, "Dynamic landing of an autonomous quadrotor on a moving platform in turbulent wind conditions," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9577–9583.

[5] A. Batool, F. Batool, R. A. Khan, M. A. Mustafa, A. Fedoseev, and D. Tsetserukou, "NMPC-Lander: Nonlinear MPC with barrier function for UAV landing on a mobile platform," arXiv preprint arXiv:2505.03931, 2025.

[6] G. Pannocchia and J. B. Rawlings, "Disturbance models for offset-free model-predictive control," *AIChE Journal*, vol. 49, no. 2, pp. 426–437, 2003.

[7] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "Pampc: Perception-aware model predictive control for quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8.

[8] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[9] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

TABLE II. Tracking-error RMS across the four hardware experiments, computed from the per-tick CSV logs. Position errors are in the OptiTrack world frame (x forward, y up, z lateral). Velocities are finite-differenced from the position stream. Yaw error is wrapped to $[-\pi, \pi]$ before RMS.

Experiment	x (m)	y (m)	z (m)	v_x (m/s)	v_y (m/s)	v_z (m/s)	yaw ($^\circ$)
Stationary	0.100	0.243	0.080	0.167	0.237	0.136	7.59
Straight-line	0.651	0.287	0.367	0.537	0.281	0.337	3.10
Circular (CCW)	0.679	0.281	0.285	0.525	0.274	0.315	15.44
Erratic	0.951	0.253	0.335	0.618	0.249	0.345	25.64