Precision Home Landing for Bee-Nav Robot Navigation

Dequan Ou*, Quentin Missinne, Christophe De Wagter, and Guido C.H.E. de Croon
Micro Air Vehicle Laboratory, Control and Operations, Delft University of Technology, The Netherlands

Abstract

Small flying robots are an attractive platform for tasks such as greenhouse monitoring, warehouse inventory, and last-mile delivery. However, for any of these applications to be sustainable, the robot must be able to return autonomously to a recharging pad without relying on GPS or external infrastructure. The recently introduced Bee-Nav navigation strategy, inspired by honeybee learning flights, brings a small quadrotor back to within 0.5 m of its takeoff point after hundreds of metres of outbound flight, but it lacks the centimetre-level precision a charging pad requires. Here we extend Bee-Nav with a SIFT- and memory-based visual-servoing approach: a single downward SIFT view of the landing zone, captured during the same takeoff hover, is stored on board and later drives a closed-loop image-based visual-servoing descent once Bee-Nav returns the drone to the home area. Across multiple flights, the custom quadrotor lands with a mean position error of 16 ± 10 cm—a $\sim$60 % reduction in mean error and a $\sim$44 % reduction in spread relative to a camera-free output-flip detection baseline. Storing a single reference image is therefore enough to turn coarse visual homing into centimetre-precise landing, closing the loop on fully autonomous, infrastructure-free return-and-recharge cycles.

1 Introduction

Small flying robots are an increasingly attractive platform for applications such as warehouse inventory tracking and last-mile delivery [2]. For any such mission to be sustainable and easy to deploy, the robot must be able to return to a docking station for autonomous recharging without relying on external positioning infrastructure such as GPS or fiducial markers [3, 4, 5, 6]. Yet the very property that makes these robots attractive—their low payload and limited on-board compute—rules out the high-precision, map-based navigation methods that larger drones use. Micro air vehicles (MAVs) carry severely limited payloads, leaving no room for LiDAR or GPU-capable companion computers and forcing the traditional SLAM pipelines that scale with map size [7] off the table.

*Email address: d.ou@tudelft.nl

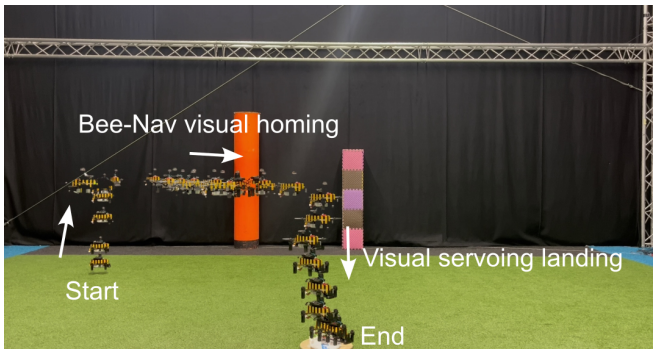

Figure 1: Time-lapse of a complete autonomous homing-and-landing flight in the CyberZoo. The drone takes off (*Start*) at a random location with random heading inside the *learned homing area* (LHA), executes Bee-Nav visual homing [1] to return to the vicinity of its takeoff point, and then performs the proposed visual-servoing-based precision landing onto a small pad (*End*).

Honeybees show that this problem is solvable with impressively minimal resources: they navigate kilometres from their hive on neural hardware of approximately one million neurons [8]. They achieve this by combining two complementary mechanisms: *path integration*, which estimates the position of the bee with respect to the hive by accumulating directions and distances travelled, and *view memory*, which maps remembered visual snapshots of the surroundings to nearby places of interest [9, 10]. In long-distance flights the two interplay: path integration provides the navigation signal while the bee is far from the nest, and view memory takes over once it is close enough to recognise familiar landmarks. The view memory itself is built during one or more short *learning flights* immediately after takeoff: while the path-integration system is still accurate, the bee deliberately samples views around the nest, and the path-integration estimate for each sample serves as the label that associates the view with its position relative to home [8, 1].

Translating this two-stage strategy onto a small drone, Bee-Nav [1] demonstrated that a 3.4-42 kB neural network trained from a single learning flight is enough to navigate back to within 0.5 m of the takeoff point from up to 110 m away in 100 % of test flights, and from up to 600 m away in the majority of outdoor flights, all using only an omnidirectional camera, an optical-flow sensor and a Raspberry Pi 4. However, 0.5 m is not enough to actually land on a recharging pad: charging contacts are typically a few centimetres

wide, and the surrounding clutter (cables, neighbouring pads, infrastructure) leaves no margin for a half-metre lateral error.

In robotics, this last-half-metre gap is bridged by what is usually called *precision landing based on prior knowledge*: engineered targets such as ArUco markers or H-shaped helipads with conventional pose estimation [3, 4], classical handcrafted feature matching against a reference image, or more recently learned descriptors. We discuss these approaches in Sec. 2.3.

What is novel here is the integration of precision landing into Bee-Nav itself: by sharing its hardware and reusing its learning flight as the deployment-time reference-capture step, the combined system becomes fully autonomous from take-off to touchdown—something existing precision-landing methods, which assume a separately instrumented landing zone, do not provide on their own.

We propose a precision-landing system that extends Bee-Nav to a fully autonomous takeoff and landing pipeline. The system captures a single SIFT view of the landing zone during Bee-Nav's takeoff hover (*i.e.* during the learning flight), and then, once Bee-Nav's visual-homing phase—which runs inside the *learned homing area* (LHA), a circle of radius roughly 5–10 m centred on home [1]—has brought the drone close to its takeoff point, the landing pipeline triggers a closed-loop visual-servoing descent that drives the drone onto the centroid of the SIFT cluster seen at takeoff. The whole pipeline uses one extra downward-facing webcam and runs on the same Raspberry Pi 4 that already runs Bee-Nav.

The contributions of this work are:

1. A complete autonomous pipeline for a central-place-foraging MAV that travels out to a remote location and returns to its takeoff point with a precision sufficient for autonomous recharging, by augmenting Bee-Nav [1] with a visual-servoing precision-landing phase.

2. A SIFT-based landing-zone representation that is robust to post-takeoff hover drift, running in real time on a Raspberry Pi 4 alongside the navigation pipeline.

3. A significant reduction in landing error over a camera-free baseline that detects home arrival from overshoot-induced flips in Bee-Nav's home-direction output (*output-flip detection*), achieving centimetre-level landing accuracy without GPS, fiducial markers, or external infrastructure.

The rest of the paper is organised as follows: Sec. 2 reviews related work; Sec. 3 details the hardware, the Bee-Nav navigation strategy, and the capture-detect-servo landing pipeline; and Sec. 4 evaluates landing precision against the camera-free baseline, before Sec. 5 concludes the findings and future work.

2 RELATED WORK

2.1 Bee-Nav insect-inspired navigation

Honeybees navigate by interplaying two mechanisms: path integration drives the trajectory when the bee is far from the nest, and view memory takes over once it is close enough to recognise familiar landmarks [9, 10]. The view memory is itself built during dedicated *learning flights* immediately after takeoff, in which the path-integration estimate labels each visual snapshot with its position relative to home [8, 1]. Bee-Nav [1] ports this strategy onto a small quadrotor by training a compact convolutional homing network from learning-flight data, achieving a $\sim$0.5 m residual error to the takeoff point on a Raspberry Pi 4. We refer the reader to [1] for the full method and to Sec. 3.2 for the four-phase summary needed to follow the rest of this paper.

A half-metre residual is, however, not enough to actually land on a recharging pad. Our work closes this gap: we keep Bee-Nav's hardware and its entire navigation pipeline, and add a precision-landing phase on top.

2.2 Autonomous drone landing with prior knowledge

Most precision-landing systems instrument the pad with a fiducial pattern—H-shaped helipads [4, 3], AprilTag [5] or ArUco [6]—and reach centimetre-level pose estimation in real time on embedded hardware; the cost is operational, since every deployment site must be tagged. A second line of work, which our system belongs to, instead captures one or more reference images of the pad at deployment time and matches each live frame against them using handcrafted scale-invariant descriptors such as SIFT [11]. Reported demonstrations include natural-landmark landing [12], fully onboard monocular SIFT pipelines [13], and SVO-based elevation mapping on textured natural surfaces [14]. The failure modes of this regime are scale-variance degradation at very low altitude and sensitivity to motion blur. Recent learned descriptors (SuperPoint [15], SuperGlue [16], LightGlue [17]) and end-to-end DRL landing policies [18] improve robustness at the cost of compute that is hard to afford on Raspberry-Pi-class companions. We adopt SIFT because it gives a deterministic error signal and runs comfortably onboard, and we control its failure modes by capturing the reference during Bee-Nav's existing learning flight (Sec. 3.4) and tightly coupling it to an IBVS descent (Sec. 3.7).

2.3 Precision landing with visual servoing

Visual servoing is the standard methodology for closing the control loop in the terminal phase of autonomous UAV landing, in which visual feedback is converted directly into closed-loop velocity or attitude commands [19]. The literature divides into two canonical architectures. *Position-based visual servoing* (PBVS) first reconstructs the 3D relative pose of the target and then generates Cartesian velocity commands, yielding clean, near optimal trajectories in the workspace but inheriting a strong dependence on accurate camera calibra-

tion, a known target model, and a reliable depth estimate, with no guarantee that the target remains within the field of view [3, 4]. *Image-based visual servoing* (IBVS) instead defines the error directly on the image plane and maps it to control through the interaction (image-Jacobian) matrix; this makes it robust to calibration error and keeps the target in view by construction, at the price of coupling translational and rotational motion and requiring an estimate of the feature depth [20, 21].

To avoid the depth- and calibration-sensitivity of PBVS and the coupled depth-dependent dynamics of IBVS, a family of hybrid, or 2.5D, approaches decouples the rotational and translational loops by operating partly in the cartesian workspace and partly in the image plane [22, 23]. Such schemes come at a cost, requiring a continuous homography or essential-matrix decomposition at every frame, which is comparatively expensive and yields several geometrically valid pose solutions that must be disambiguated through heuristics. For Size, Weight, and Power (SWaP)-constrained platforms such as the Raspberry-Pi-based system used here, this overhead is significantly undesirable, and pure IBVS remains the most appealing option; it offers a deterministic pixel-to-command mapping and minimal perception-to-actuation latency, with no decomposition ambiguity to resolve in flight.

A remaining difficulty, independent of the servoing architecture, is the definition of a stable geometrically meaningful target to servo towards as the apparent scale of the scene changes throughout the descent. Servoing to whichever local features happen to be detected ties the control objective to arbitrary texture and drifts as features appear and disappear. A more robust strategy is to match each live frame against a reference image of the landing zone and to project a single canonical target point, fixed once on the reference, into the current frame through the estimated homography, thereby decoupling the control objective from the specific features visible at any given instant [24, 12]. We adopt exactly this strategy: a pure IBVS descent commands a body-frame velocity towards a homography-projected target point that is fixed once during the learning-flight hover. This results in a deterministic, low-latency landing controller that fits within the SWaP budget already imposed by the navigation pipeline.

3 Methods

The proposed system combines the Bee-Nav navigation strategy with a precision-landing sub-system. The precision-landing pipeline is fully driven by a single downward-facing webcam and the on-board flight controller's state estimate (vehicle position fused from optical flow and a downward LiDAR). It is divided into two primary steps: landing trigger and visual servoing.

3.1 Hardware system

The platform is a custom-built quadrotor (Fig. 2) running PX4 on a Holybro PixHawk 6c mini flight controller.

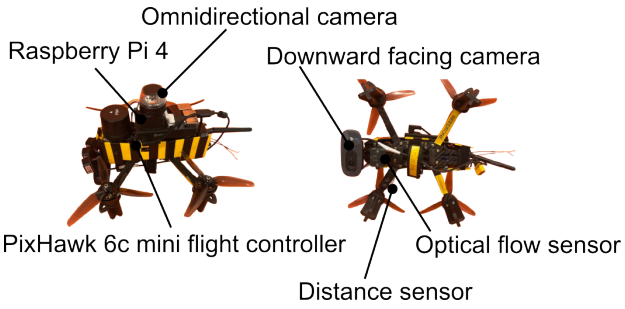

Figure 2: The custom quadrotor platform BeeDrone used in this work, shown in isometric and bottom views. The upward-pointing omnidirectional Pi-Camera v2 feeds the Bee-Nav visual-learning and homing network, while the downward-facing webcam is the sole sensor used by the precision-landing pipeline. The flight controller runs PX4; a Raspberry Pi 4 runs the vision pipeline. A downward optical-flow sensor and a distance sensor are fused by the autopilot to provide velocity and position estimates along all three axes.

The on-board companion computer is a Raspberry Pi 4 connected to the flight controller through a serial link using the micro-XRCE-DDS bridge. The Pi runs the vision pipeline on top of Raspberry Pi OS using OpenCV and a lightweight Flask HTTP layer that decouples the cadence of the vision algorithms from the ROS 2 state machine. The drone is equipped with:

- An upward-pointing fish-eye Pi-Camera v2 that feeds the Bee-Nav visual-homing network, which maps a single panoramic frame to a body-frame home vector.
- A downward-facing Logitech C270 USB webcam, the only sensor used for the precision-landing pipeline described below.
- An optical-flow sensor (PMW3901) and a downward LiDAR rangefinder (TFmini-S), fused by PX4's EKF2 to provide velocity and position estimates along all three axes.

3.2 Bee-Nav visual learning and homing

Our system extends Bee-Nav [1] with two additional phases (home capture and precision landing) that share its hardware. Bee-Nav itself consists of four phases that we refer to in the rest of this section; full details are in [1]. During a *learning flight*, the drone records pairs of (omnidirectional image, path-integration information) while flying a structured pattern within 5–10 m of the takeoff point, and a small CNN is trained to map each omnidirectional image to its associated home vector; the region within which the network can be used to reach home is called the *learned homing area* (LHA). Afterwards, when performing a task, the drone flies an *outbound* task trajectory while path integration tracks the home

direction. Once the task is completed, it follows the integrated home vector during an *inbound* flight (not ending at true home because of drift), and then enters a *visual-homing* phase inside the LHA in which the CNN is run at each step until the drone reaches home. The precision-landing pipeline inserts immediately after this visual-homing phase. In the original Bee-Nav paper, the operator triggered landing manually once the drone came within 0.5 m of the home pad; here, as a camera-free baseline, we additionally exploit a behavioural property of the same network's outputs at convergence (Sec. 3.3).

3.3 Output-flip detection (camera-free baseline)

Bee-Nav's homing network points the drone reliably toward its takeoff point throughout the return: far from home, the predicted home vector is smooth and converges, step by step, toward the takeoff point. Inside the last ~ 0.5 m, however, the drone's residual velocity carries it past home before the controller can fully decelerate, and it overshoots. Once on the other side of home, the home direction now points back the way the drone just came, so the drone reverses; on the next step it overshoots again, and so on. Repeated overshoots make the home-vector output *flip* direction every few frames as the drone dances back and forth across home—the expected behaviour of any well-tuned homing controller at convergence, not a failure of the network.

This back-and-forth pattern can itself be used as a landing trigger that requires no downward camera, no SIFT, and no extra hardware on top of Bee-Nav. We track the most recent five home-direction outputs and call a successive pair a *flip* when their angles differ by more than $90°$. When at least three of the last four pairs flip, the trigger fires: the drone enters the open-loop velocity descent of Sec. 3.6 directly and lands *in place*, accepting whatever residual error remains in the Bee-Nav inbound trajectory. We refer to this as the *output-flip detection* baseline, against which both SIFT-only and SIFT+IBVS variants are compared in Sec. 4.

3.4 Home capture at takeoff

We capture the SIFT view of the landing pad during the first hover of Bee-Nav's learning flight: at this moment the drone is hovering at takeoff altitude over (or close to) its takeoff point, and the downward webcam has a clean, sharp top-down view of whatever surface the drone will later be asked to land on. The onboard vision daemon extracts SIFT keypoints and descriptors from this frame and caches the raw image, the features, and a single 2-D target point $\mathbf{u}^\star = (u^\star, v^\star)$ in image coordinates that marks the location in the reference frame to which the drone will later be servoed.

The choice of $\mathbf{u}^\star$ is non-trivial. A naive choice would be the image centre; however, because the drone is typically not perfectly above the landing pad when the reference is captured (lateral drift during takeoff), the image centre frequently falls onto the edge or the surface next to the pad. Landing on this point therefore reproduces the takeoff drift instead of the actual pad location. Instead, we set $\mathbf{u}^\star$ to the centroid of the densest cluster of SIFT keypoints, which on a textured pad reliably falls on the pad surface itself (the pad has more high-response SIFT features than the background). The cluster centroid is computed by iterative response-weighted mean-shift:

$$\mathbf{c}_{k+1} = \frac{\sum_{i \in \mathcal{N}(\mathbf{c}_k)} r_i \, \mathbf{p}_i}{\sum_{i \in \mathcal{N}(\mathbf{c}_k)} r_i}, \tag{1}$$

where $\mathbf{p}_i$ and r_i are the position and response of keypoint i, $\mathcal{N}(\mathbf{c}_k)$ is the set of keypoints within bandwidth ρ of the current centroid, and ρ is set to 20% of the shorter image dimension. The iteration is initialised at the response-weighted mean of all keypoints and terminates either when the centroid moves less than 0.5 px between steps or after a 15-iteration cap.

3.5 SIFT-based pad detection during return

During the visual homing phase, the same vision daemon loads the cached SIFT reference at startup and runs a background detector at 5 Hz against the latest downward webcam frame. Each cycle runs SIFT detect-and-compute on the live frame, matches the live descriptors to the cached reference descriptors with a FLANN $k = 2$ search, keeps matches whose nearest/second-nearest distance ratio is below 0.75 (Lowe ratio), and fits a homography H by RANSAC. A detection is accepted only if at least 10 good matches and 10 RANSAC inliers survive with an inlier ratio above 0.4, and if the target $\hat{\mathbf{u}} = H\mathbf{u}^\star$ warped through H falls within the inner 80% of the live frame. The accepted target pixel is back-projected through the pinhole model and the current above-ground altitude h to a body-frame error $\Delta x_b = -(\hat{v} - c_y)h/f_y$, $\Delta y_b = +(\hat{u} - c_x)h/f_x$, and converted to polar (α, d). The pinhole intrinsics are derived from the C270 specification (60° diagonal FOV at 640×480) without per-unit calibration, since a small bias on d is absorbed by the closed-loop nature of the descent (Sec. 3.7). The detector runs at the chosen 5 Hz cadence with headroom for the rest of the onboard pipeline.

Each successful match is forwarded to the landing controller, which caches the most recent body-frame error along with a timestamp. This pushed cache is consumed by the landing state machine described next.

3.6 Landing state machine

The landing sub-system is a small state machine driven entirely by external set-points sent to the autopilot, so its own internal landing modes are never invoked (Fig. 4). There are two possible entry events and two operating states. An accepted SIFT detection enters the *IBVS* state, in which a body-frame XY velocity command is computed from the latest SIFT update (Sec. 3.7) and a constant descent rate is commanded vertically. The IBVS state holds until either the autopilot reports ground contact, in which case the drone disarms, or the SIFT inlier count drops below a minimum, in

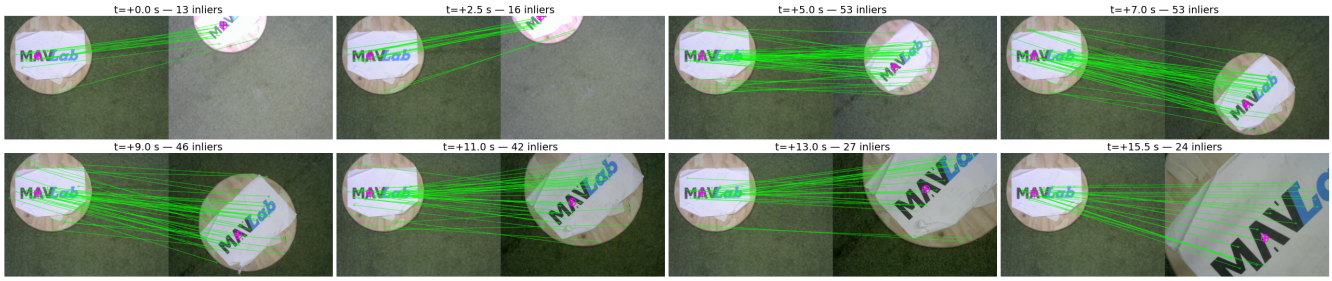

Figure 3: Onboard SIFT feature matching during the visual-servoing descent of a representative SIFT+IBVS flight, with eight frames sampled at ∼2 s intervals across the IBVS state. Each panel shows the takeoff-time landing reference (*left half*) and the live downward-camera frame (*right half*), with green lines connecting RANSAC-inlier keypoint correspondences and a magenta cross marking the pad-cluster centroid (the landing target) in each image.

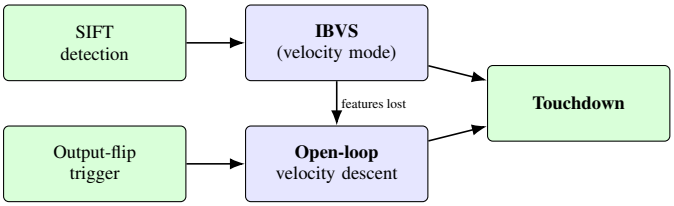

Figure 4: Landing state machine. Two entry events: an accepted SIFT detection (top-left) goes straight into velocity-mode IBVS; the camera-free output-flip detection trigger (bottom-left) goes straight into an open-loop velocity descent. The IBVS state falls through to the open-loop descent if SIFT features are lost. Both descent states terminate when PX4 reports touchdown.

which case the controller falls through to an *open-loop velocity descent*: planar position is held at the current (x, y) and the same constant downward velocity is commanded until touchdown. The output-flip detection trigger (Sec. 3.3) enters the same open-loop velocity descent directly without ever passing through the IBVS state.

3.7 *Image-based visual servoing*

The IBVS state implements a proportional image-based visual servo at the velocity set-point. Given the most recent SIFT push (Sec. 3.5)—body-frame range d and bearing α, equivalently the back-projected error $(\Delta x_b, \Delta y_b) = (d\cos\alpha, d\sin\alpha)$—the controller commands a planar speed proportional to d, rotated into the world frame by the current yaw ψ, and a constant descent rate:

$$\begin{aligned} v_{\text{des}} &= \text{clip}\big(K_v\, d,\ -v_{\max},\ +v_{\max}\big), \\ v_x^w &= v_{\text{des}}\ \cos(\psi+\alpha), \\ v_y^w &= v_{\text{des}}\ \sin(\psi+\alpha), \\ v_z &= v_d, \end{aligned} \tag{2}$$

which is published as a velocity set-point to the autopilot, whose own attitude and thrust loops track it. Entry into the IBVS state is gated by two conditions: (i) the IBVS mode must be enabled in the controller configuration; and (ii) the latest SIFT detection must be fresher than $\tau_{\text{lost}} = 300$ ms. Failure of either gate falls through to the open-loop velocity descent instead.

4 RESULTS

We evaluate three landing conditions back-to-back in the same indoor arena (CyberZoo motion-capture cage, 10×10 m): (i) the camera-free *output-flip detection* baseline of Sec. 3.3, which lands the drone in place as soon as Bee-Nav's home-direction output starts flipping from overshoot at convergence; (ii) *SIFT only*, in which a SIFT detection triggers a landing but IBVS is disabled, so the controller falls through to an open-loop velocity descent with no closed-loop XY correction; and (iii) *SIFT + IBVS*, the full system. Our focus is what SIFT+IBVS adds over Bee-Nav's own outputs; benchmarks against fiducial or learned-descriptor methods are left to future work. Eight flights per condition correspond to eight equally-spaced starting positions and directions covering the full 360° inside the LHA. All flights used the same pad and lighting; texture and illumination robustness are noted as limitations. Ground-truth trajectories were recorded from an OptiTrack motion-capture system, and only the mocap stream was used for analysis.

The final landing position relative to the target is shown in Fig. 5; the target is defined as the touchdown of a single reference SIFT+IBVS flight (whose SIFT view was acquired during the same learning flight) so that the metric is anchored to the actual visual target rather than to an arbitrary mocap-frame origin. The camera-free output-flip detection baseline lands all 8 trials but with a mean error of 0.40 ± 0.18 m and a worst case of 0.69 m—useful as a fallback when no downward camera is available, but well outside the radius a typical charging pad can tolerate. Adding the SIFT-based trigger reduces this to 0.25 ± 0.11 m, and adding velocity-mode IBVS brings it further down to 0.16 ± 0.10 m. The 0.42 m worst case of the SIFT-only condition (visible in the lower-

Table 1: Per-flight landing metrics, mean ± one standard deviation across $n = 8$ flights per condition. Bold row highlights the headline precision metric. †For output-flip detection the time-to-land row is computed over the 7 of 8 trials in which the drone actually entered the 0.5 m radius before touchdown; the eighth landed at 0.60 m without ever crossing the radius.

Metric	Output-flip	SIFT only	SIFT+IBVS
Landing error [m]	$\mathbf{0.40 \pm 0.18}$	$\mathbf{0.25 \pm 0.11}$	$\mathbf{0.16 \pm 0.10}$
Time to land from 0.5 m [s]	$14.8 \pm 5.3^{\dagger}$	14.4 ± 5.9	21.9 ± 17.4
Descent path length [m]	0.85 ± 0.01	0.89 ± 0.02	0.98 ± 0.10
Descent tortuosity [-]	0.002 ± 0.001	0.037 ± 0.017	0.126 ± 0.088
Mean horizontal speed [m/s]	0.051 ± 0.027	0.037 ± 0.008	0.066 ± 0.022

left of its panel) is absent from the IBVS condition, whose worst landing is 0.30 m. An unpaired Welch's t-test confirms that SIFT+IBVS significantly reduces landing error over the output-flip baseline ($p = 0.008$, Cohen's $d = 1.62$); SIFT-only shows a similarly large effect vs. output-flip ($d = 1.01$, $p = 0.07$), and the further SIFT+IBVS-over-SIFT-only improvement is a large trend not yet significant at $n = 8$ ($d = 0.85$, $p = 0.11$).

Fig. 6 shows the visual-homing and landing trajectories. Outside the 0.5 m radius (light gray), the eight launch directions all converge on the target; inside the radius (per-condition colour), landing behaviour diverges. Output-flip detection lands wherever the trigger fires, often outside the dashed circle (one trial landed at 0.60 m without ever entering). The SIFT-driven conditions reliably enter the radius and bias the touchdown toward the target, with SIFT+IBVS producing the densest cluster.

As a complementary measure to landing-position accuracy, we report the *time-to-land from* 0.5 *m*—the elapsed time from the first sample at which the drone enters the 0.5 m radius until touchdown. This is an end-to-end view of how long each method spends in the landing zone before committing to land, and is summarised together with the landing error in Table 1.

Time-to-land is comparable between output-flip detection and SIFT-only (∼14 s on average): both commit to a descent as soon as their respective trigger fires, with no further requirement on the SIFT detection thereafter. SIFT+IBVS, in contrast, averages ∼ 22 s with much larger run-to-run variance: the median is similar at ∼ 14 s, but two of the eight trials took 45 and 54 s as the IBVS loop spent additional time waiting for the SIFT match to stabilise. The mechanism is the entry condition into the IBVS state (Sec. 3.7); failure of that condition falls through to an open-loop velocity descent, which gives up the closed-loop XY correction. To preserve precision, the system therefore effectively waits for the drone to come closer to home, where the pad spans a larger image area and matches are denser and more frequent, before and during the closed-loop descent.

5 DISCUSSION AND CONCLUSIONS

Bee-Nav alone closes a multi-hundred-metre round-trip to within 0.5 m of the takeoff point [1]; the proposed extension takes that residual to inside one charging-pad radius, making the autonomous foraging-and-recharging cycle viable on a Raspberry-Pi-class platform. The camera-free output-flip baseline lands roughly 2.5× less accurately and is best regarded as a fallback for when no downward camera is available. The closed-loop IBVS pays a real efficiency cost (a longer, more variable approach inside the landing zone) for a substantial precision gain — the expected trade-off of a closed-loop visual servo, and favourable for a fixed charging pad.

Some limitations are worth flagging:

- *Low-altitude blind zone*: below ∼ 0.3 m the pad fills the frame and the surrounding texture that anchors the homography leaves view, so residual error accumulates in the last stretch of the descent.
- *Detector cadence*: the 5 Hz SIFT loop is adequate for a static pad but would not track a moving platform or hold up under strong wind disturbance; a lighter-weight detector is needed for those regimes.
- *Detector and back-projection assumptions*: a half-visible pad, a strongly textured ground (e.g. grass), or roll and pitch during descent all shift the SIFT-cluster target off the pad; if any of these escalates to the pad leaving the camera's view entirely, or to the background out-competing the pad in keypoints, the pipeline simply cannot land precisely.
- *Scope of the evaluation*: $n = 8$ per condition gives directional coverage but limited statistical power (the SIFT+IBVS vs. SIFT-only comparison is a large-effect trend rather than a firm p-value); outdoor grass results are pending, and robustness to illumination changes, shadows, and partial pad occlusion was not evaluated.

Future work follows from these limitations. (i) A single visual system shared between Bee-Nav homing and pad detection (e.g. an omnidirectional camera with sufficient vertical FOV) would reduce the sensor count, though a panoramic reference is geometrically harder to match than a rectilinear one. (ii) Replacing SIFT with a learned local-feature pipeline (SuperPoint [15]+SuperGlue [16]/LightGlue [17]) or a lighter-weight traditional extractor would attack the motion-blur and scale-variance failure modes, and would let sub-Pi-4 platforms (STM32, Coral) — on which Bee-Nav already runs — carry the landing pipeline too. (iii) Additional low-altitude reference templates captured during takeoff (a "multi-bank" SIFT) would close the low-altitude blind zone.

In summary, we have extended Bee-Nav with a SIFT-based visual-servoing precision-landing phase that reaches centimetre-level touchdown without GPS, fiducial markers,

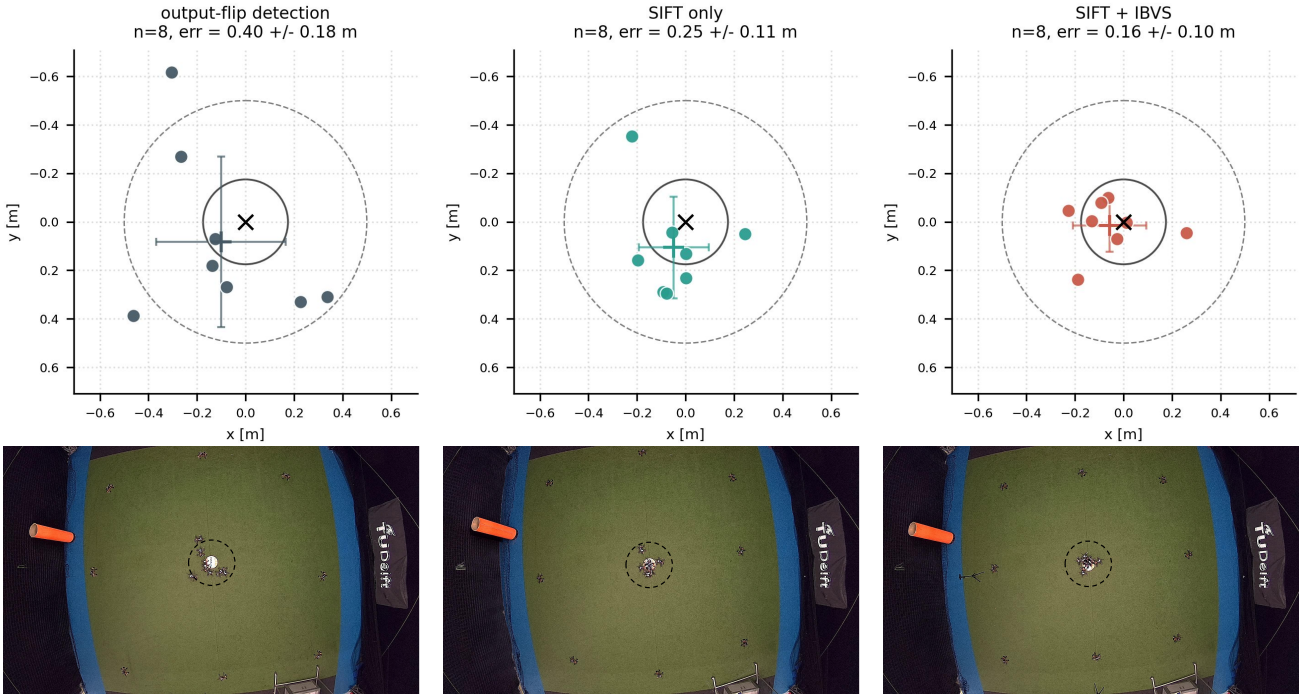

Figure 5: Touchdown precision across eight flights per condition (left: output-flip detection; middle: SIFT only; right: SIFT+IBVS). *Top row:* touchdown points expressed in the OptiTrack mocap frame relative to the visual target (black cross; defined as the touchdown of a single reference SIFT+IBVS flight). The solid inner circle (0.175 m radius) matches the size of the actual landing pad; the dashed outer circle (0.5 m radius) marks the homing-zone boundary used as the landing trigger. The coloured cross and error bars give the per-condition centroid and 1σ ellipse. Mean landing error: output-flip detection 0.40 ± 0.18 m, SIFT only 0.25 ± 0.11 m, SIFT+IBVS 0.16 ± 0.10 m. *Bottom row:* top-view time-lapse of the same eight flights inside the CyberZoo, showing the drone from launch at the edges of the cage to touchdown near the central pad; the dashed circle marks the 0.5 m homing-zone radius.

or environment-side instrumentation, reusing the same hardware and learning flight Bee-Nav already performs. This makes a fully autonomous central-place-foraging round-trip practical on limited onboard compute.

Acknowledgements

This project was financed by the Dutch Research Council (NWO) under grant number 20663 of the VICI personal grant programme. We thank Taran J. Singh for his assistance with the experiments.

References

[1] Dequan Ou, Jesse J Hagenaars, Maciej R Jankowski, Michiel VM Firlefyn, Christophe De Wagter, Florian T Muijres, Jacqueline Degen, and Guido CHE de Croon. Efficient robot navigation inspired by honeybee learning flights. *Nature*, 653(8116):1039–1046, 2026.

[2] Marius Beul, David Droeschel, Matthias Nieuwenhuisen, Jan Quenzel, Sebastian Houben, and Sven Behnke. Fast autonomous flight in warehouses for inventory applications. *IEEE Robotics and Automation Letters*, 3(4):3121–3128, 2018.

[3] C. Patruno, M. Nitti, A. Petitti, E. Stella, and T. D'Orazio. A Vision-Based Approach for Unmanned Aerial Vehicle Landing. *Journal of Intelligent & Robotic Systems*, 95(2):645–664, August 2019.

[4] S. Saripalli, J.F. Montgomery, and G.S. Sukhatme. Visually guided landing of an unmanned aerial vehicle. *IEEE Transactions on Robotics and Automation*, 19(3):371–380, June 2003.

[5] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011.

[6] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

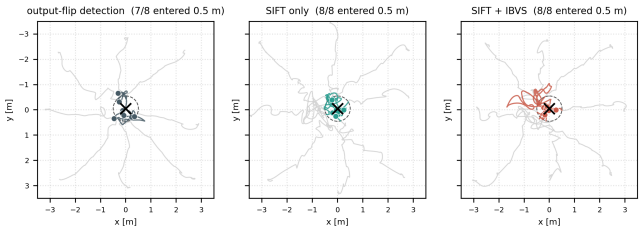

Figure 6: XY trajectories of the visual-homing and landing phases for the eight flights per condition. Outside the 0.5 m landing-zone radius around the target (dashed circle), trajectories are drawn in light gray; inside the radius, all flights are drawn in the per-condition colour, pooling the in-zone behaviour visually. Black cross: target. Filled dot at the end of each trajectory: touchdown.

[7] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[8] Jacqueline Degen, Andreas Kirbach, Lutz Reiter, Konstantin Lehmann, Philipp Norton, Mona Storms, Miriam Koblofsky, Sarah Winter, Petya B. Georgieva, Hai Nguyen, Hayfe Chamkhi, Uwe Greggers, and Randolf Menzel. Exploratory behaviour of honeybees during orientation flights. *Animal Behaviour*, 102:45–57, 2015.

[9] Thomas Stone, Barbara Webb, Andrea Adden, Nicolai Ben Weddig, Anna Honkanen, Rachel Templin, William Wcislo, Luca Scimeca, Eric Warrant, and Stanley Heinze. An anatomically constrained model for path integration in the bee brain. *Current Biology*, 27(20):3069–3085, 2017.

[10] BA Cartwright and TS Collett. Landmark maps for honeybees. *Biological cybernetics*, 57(1):85–93, 1987.

[11] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[12] Andrea Cesetti, Emanuele Frontoni, Adriano Mancini, Primo Zingaretti, and Sauro Longhi. A vision-based guidance system for UAV navigation and safe landing using natural landmarks. *Journal of Intelligent & Robotic Systems*, 57(1):233–257, 2010.

[13] Shaowu Yang, Sebastian A. Scherer, and Andreas Zell. An onboard monocular vision system for autonomous takeoff, hovering and landing of a micro aerial vehicle. *Journal of Intelligent & Robotic Systems*, 69(1):499–515, 2013.

[14] Christian Forster, Matthias Faessler, Federico Fontana, Manuel Werlberger, and Davide Scaramuzza. Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 111–118, 2015.

[15] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperPoint: Self-supervised interest point detection and description. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 337–349, 2018.

[16] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning feature matching with graph neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4937–4946, 2020.

[17] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. LightGlue: Local feature matching at light speed. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17581–17592, 2023.

[18] Alejandro Rodriguez-Ramos, Carlos Sampedro, Hriday Bavle, Paloma de la Puente, and Pascual Campoy. A deep reinforcement learning strategy for UAV autonomous landing on a moving platform. *Journal of Intelligent & Robotic Systems*, 93(1):351–366, 2019.

[19] François Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, 2006.

[20] Daewon Lee, Tyler Ryan, and H Jin Kim. Autonomous landing of a vtol uav on a moving platform using image-based visual servoing. In *2012 IEEE International Conference on Robotics and Automation*, pages 971–976. IEEE, 2012.

[21] Pedro Serra, Rita Cunha, Tarek Hamel, David Cabecinhas, and Carlos Silvestre. Landing of a quadrotor on a moving target using dynamic image-based visual servo control. *IEEE Transactions on Robotics*, 32(6):1524–1535, 2016.

[22] Ezio Malis, Francois Chaumette, and Sylvie Boudet. 2 1/2 d visual servoing. *IEEE Transactions on Robotics and Automation*, 15(2):238–250, 1999.

[23] Francois Chaumette and Seth Hutchinson. Visual servo control. ii. advanced approaches [tutorial]. *IEEE Robotics & Automation Magazine*, 14(1):109–118, 2007.

[24] Selim Benhimane and Ezio Malis. Homography-based 2d visual tracking and servoing. *The International Journal of Robotics Research*, 26(7):661–676, 2007.