POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Toward Context-aware LLM-driven UAVs

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Christian Mariano

Advisor: Prof. Luca Mottola

Academic year: 2025-2026

1. Introduction

Unmanned Aerial Vehicles (UAVs) are progressively evolving from manually piloted platforms into autonomous systems capable of interacting with users through high-level instructions. This transformation introduces new challenges in Human-Drone Interaction (HDI), where usability, adaptability, and context-awareness become central design requirements. Rather than framing interaction as a sequence of isolated commands, recent research increasingly understands HDI as a situated and collaborative process emerging from shared perception and mutual adaptation. In this perspective, interaction unfolds within continuous perception-action loops in which both users and systems co-evolve over time [6].

The evolution of HDI paradigms, illustrated in Figure 1, reflects a gradual transition from manual control toward adaptive and cognitively driven interaction models. Early approaches relied on direct teleoperation, where users explicitly controlled drone motion through continuous input, requiring sustained attention and domain expertise. Shared autonomy subsequently introduced partial automation, combining human guidance with onboard stabilization, obstacle avoidance, or goal-directed assistance, thereby reducing user burden while retaining human oversight.

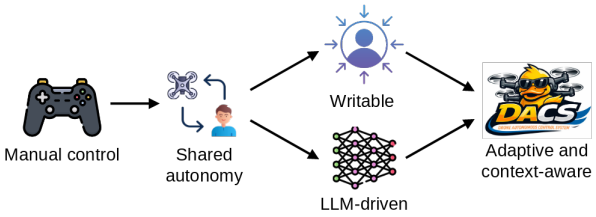

Figure 1: Evolution of Human-Drone Interaction paradigms.

More recent paradigms further abstract interaction from motion control to behavior specification. Writable interaction approaches [1] enable users to actively shape drone behavior by composing new executable actions during use, promoting flexibility and personalization. In parallel, LLM-driven systems [4] emphasize autonomous reasoning, allowing drones to interpret natural-language goals and generate execution plans dynamically. Although these directions advance HDI along complementary trajectories, their integration remains largely unexplored, revealing a gap between user-driven authorship and reasoning-based autonomy.

Recent advances in Large Language Models (LLMs) have enabled drones to interpret natural-language instructions and generate structured execution plans. In Figure 1, this paradigm is represented by the *LLM-driven* in-

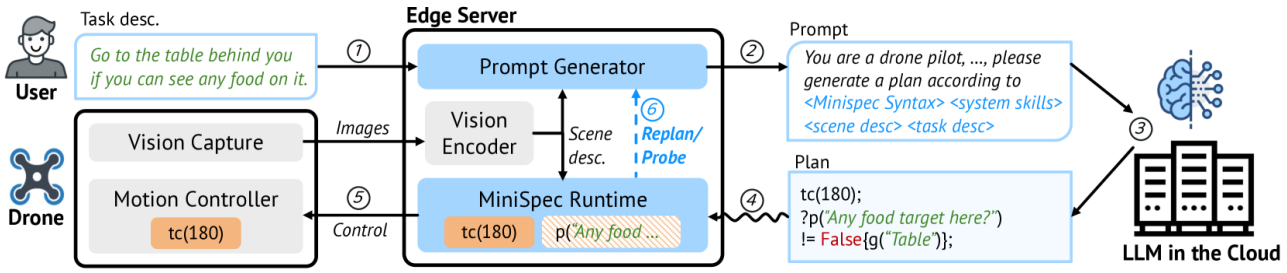

Figure 2: Overview of `TypeFly`.

teraction model. A concrete example of this approach is `TypeFly` [3], which demonstrate how an LLM can function as a cognitive planning engine that translates user goals into executable programs grounded in perception. By framing interaction as a conversational planning process, these systems reduce user effort and support flexible task specification. However, reasoning often remains episodic and stateless: perception is treated as transient input, and the system does not retain long-term contextual knowledge or user preferences, limiting adaptation across tasks and over time.

To address this limitation, we introduce the `Drone Adaptive Context-aware System (DACS)`, an HDI framework that enables drones to reason over their surroundings, remember past interactions, and progressively adjust their behavior to individual users. Rather than treating each request as an isolated command, `DACS` supports interaction that evolves over time, combining natural-language understanding with contextual awareness and personalized adaptation. In a comparative user study, this approach results in higher perceived autonomy and greater confidence in exploration behavior compared to the baseline `TypeFly`, with users reporting that the system progressively adjusts its behavior according to their individual needs and preferences.

2. Design and Implementation

We first outline the architectural foundation of the system by summarizing `TypeFly` as the LLM-driven baseline, and then present `DACS` as its adaptive and context-aware extension.

2.1. TypeFly: LLM-Driven Drone Interaction

Before introducing DACS, we summarize `TypeFly` [3], the LLM-driven framework that provides the architectural baseline for this work. `TypeFly` connects natural-language task specification to robotic execution through LLM-driven planning, edge-based perception, and a domain-specific execution language.

As shown in Figure 2, a user issues a task in natural language, while the drone streams visual data to an edge server. A vision module converts images into a list of detected objects, which, together with the user task and available robot skills, are assembled into a structured prompt sent to a cloud-hosted LLM.

The LLM generates a plan in `MiniSpec`, a lightweight and intentionally minimal language designed for concise programs. Its constrained structure promotes predictable and safe behavior.

Skills are organized hierarchically: low-level skills expose primitive sensing and motion operations, including a dedicated `query` skill that allows a plan to invoke the LLM at runtime when additional reasoning is required. High-level skills encapsulate reusable behaviors such as object search. These abstractions guide the LLM toward reliable execution.

2.2. DACS

Building on the `TypeFly` framework introduced in Section 2.1, `DACS` extends LLM-driven human-drone interaction with mechanisms for persistent adaptation and context-aware reasoning.

To illustrate the design choices, we refer throughout this section to a simple yet representative task: *"find an apple"*. While this request can already be executed in `TypeFly`, it exposes a key limitation: search behavior follows

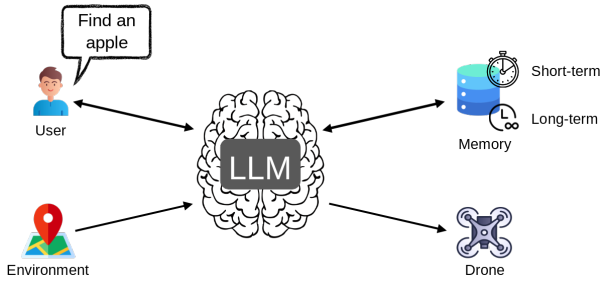

Figure 3: Conceptual architecture of DACS.

a fixed local scanning routine, where the drone repeatedly rotates and checks visibility from its current position, independently of past observations, user preferences, or broader environmental context. This static execution pattern motivates the adaptive and context-aware extensions introduced in DACS.

Figure 3 presents the conceptual architecture of DACS. At the center of the system lies an LLM-based reasoning core that integrates user input, environmental knowledge, and user-specified preferences to produce structured MiniSpec programs. Unlike the baseline, planning is informed by accumulated context rather than limited to isolated, reactive execution.

From an implementation perspective, this reasoning core is realized through a distributed architecture based on specialized *LLM instances*. Inspired by modular cognitive architectures for robotics that decompose generative reasoning into role-specific modules [5], each instance is assigned a narrowly scoped reasoning role defined by a fixed system prompt, a dynamically generated user prompt, and dedicated configuration parameters that regulate model behavior, including response verbosity and reasoning effort. The `Plan LLM` instance acts as the central orchestrator, generating MiniSpec programs that may invoke auxiliary reasoning modules.

This modular design enables the separation of adaptation, contextual reasoning, and execution-level decisions. From a deployment perspective, sensing and actuation remain on the drone, whereas adaptive reasoning and contextual modeling are handled by software components running on the edge server and invoking cloud-hosted LLM services.

2.2.1 User Adaptation

In contrast to `TypeFly`, where each request is handled largely in isolation, a primary objective of DACS is to enable interaction that evolves through repeated use. Rather than re-executing the same strategy for similar tasks, the system accumulates knowledge about prior outcomes and user preferences, progressively shaping its reasoning and adapting its behavior over time.

High-level skill creation represents a key point of departure from the `TypeFly` design. In `TypeFly`, high-level skills are statically defined by the programmer and included in the planning prompt to bias the LLM toward reliable execution patterns. While effective for guiding reasoning, these abstractions remain fixed and cannot evolve through interaction.

In DACS, instead, high-level skills are dynamically defined by the LLM itself whenever it identifies a recurring or structured behavior that would be beneficial to encapsulate. Rather than regenerating the same sequence of low-level actions for similar tasks, the planner can decide to package that behavior into a reusable MiniSpec abstraction.

In the illustrative task *"find an apple"*, for instance, an exploration strategy may be abstracted into a skill such as `search_object(object)`. Once created, this skill not only reduces planning latency by avoiding full plan regeneration, but also biases future reasoning toward validated behaviors learned from prior interactions.

This progressive evolution is supported by **long-term memory**, which stores persistent knowledge across interactions, including generated skills and accumulated behavioral refinements. These elements act as stable priors that influence planning beyond a single task.

Adaptation further extends through **user feedback**. Feedback provided after execution is stored in long-term memory and may modify existing skills or influence future planning decisions. For example, instructions such as *"Next time you find an apple, tell me where you are"* are interpreted as behavioral refinements that shape future behavior. Over time, interaction converges toward personalized routines aligned with the user's preferences.

2.2.2 Context Awareness

Alongside user adaptation, DACS introduces context awareness as a core design principle. The system integrates multimodal perception, environmental memory, and spatial constraints directly into the reasoning workflow, enabling planning decisions that reflect both semantic understanding and geometric structure.

Visual environment awareness extends reasoning beyond the symbolic object lists adopted in TypeFly. While TypeFly relies primarily on transient scene descriptions to trigger predefined search behaviors, it does not explicitly reason about the surrounding environment to decide where the drone should navigate in order to maximize progress toward fulfilling the user task.

In DACS, selected LLM instances can instead reason over raw images captured by the drone, enabling navigation decisions grounded in environmental context rather than purely reactive execution patterns. Visual cues such as spatial layout, object groupings, or surface context can inform exploration even when the target object is not immediately detected. In the *"find an apple"* scenario, contextual elements such as a fruit bowl or kitchen surface guide exploration toward semantically promising regions, shifting execution from reactive scanning toward context-driven navigation.

Environmental memory complements this capability by maintaining a persistent representation of previously explored regions and observed objects, a feature absent in TypeFly, where each task is processed without retaining knowledge of prior exploration. In DACS, this memory is implemented as a structured *context graph* that incrementally records explored regions, detected objects, and their spatial associations.

When a new task is issued, the planner can exploit this accumulated knowledge to prioritize semantically relevant areas instead of restarting exploration blindly. For instance, in the *"find an apple"* scenario, if an apple was previously detected in a specific region, the system can directly navigate toward that region, reducing unnecessary exploration and improving efficiency across repeated use.

Short-term memory further differentiates the two systems at the execution level. In TypeFly, short-term memory is limited to retaining the generated plan during execution, without recording how that plan unfolds at runtime.

In DACS, short-term memory captures execution traces generated during the current task, including explored viewpoints, failed attempts, and intermediate outcomes. This information is used by the LLM when generating subsequent planning steps if the previous iteration did not fulfill the user request, allowing reasoning to build upon recent execution history rather than restarting from scratch. For instance, if a region has already been explored without locating the target object, the next planning step avoids revisiting that area.

In contrast to long-term memory, which stores persistent knowledge such as validated skills and behavioral refinements, short-term memory is designed to retain only transient, task-specific information. Execution traces often reflect situational conditions and temporary decisions that are not meaningful beyond the current episode. By discarding these traces once the task is completed, the system prevents irrelevant runtime details from being accumulated in long-term memory, preserving the integrity of persistent knowledge.

Finally, spatial reasoning is further supported through **flyzones**, which allow users to define operational boundaries using natural language. These descriptions are translated into polygonal regions that constrain navigation and ensure that generated plans remain within explicitly defined safe operating areas. By restricting motion to user-approved regions, flyzones prevent unintended displacements, reduce the risk of collisions, and increase user trust in autonomous behavior.

3. Prototype Platform

To validate the proposed design, DACS was implemented on a distributed aerial robotics platform that separates real-time flight execution from high-level reasoning.

The aerial platform is based on a DJI Tello quadrotor, which provides RGB video streaming and executes motion primitives generated by the planning system. To enable accurate indoor positioning, a Bitcraze Crazyflie 2.1 equipped with a Lighthouse positioning deck is

rigidly mounted on the drone and used exclusively as a localization sensor.
All perception, planning, and memory management components run on a Linux-based edge workstation. The edge server aggregates visual input and localization data, constructs prompts for the reasoning modules, interprets MiniSpec plans, and coordinates execution. Language model inference is performed remotely using `GPT-5` accessed through API calls, allowing computationally intensive reasoning to be decoupled from real-time flight control.

4. Evaluation

We focus on whether adaptive and context-aware mechanisms improve interaction quality, reasoning behavior, and perceived autonomy compared to the baseline system `TypeFly`. The evaluation adopts a between-subject [2] user study with ten participants divided into two independent groups, each interacting with only one system to avoid transfer effects.
Participants exhibit heterogeneous backgrounds. Most have a technical or engineering profile, while each group includes at least one non-technical user. Direct drone experience is generally limited, with most participants reporting little or no prior piloting experience. In contrast, familiarity with AI and natural-language interfaces is moderate to high across all users.

4.1. Study Design

Participants interact with the system through a chat-based web interface across a sequence of open-ended scenarios, including object search, exploration when the target is not immediately visible, and unconstrained interaction. Tasks are intentionally unscripted to observe spontaneous behavior, learning strategies, and feature discoverability. Both systems share the same hardware platform and interaction modality, allowing observed differences to be attributed primarily to the adaptive mechanisms introduced in `DACS`.
Evaluation combines three complementary perspectives: (i) user-reported questionnaires assessing perceived usability and autonomy, (ii) qualitative analysis of interaction logs, and (iii) quantitative metrics extracted from system execution traces.

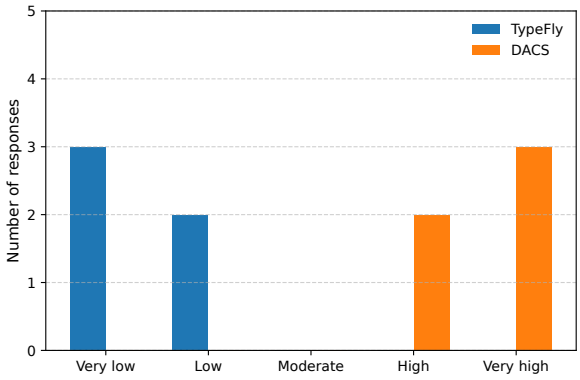

Figure 4: User-reported usefulness of feedback-induced behavioral change.

4.2. Results

User Adaptation. User adaptation emerges primarily through persistent feedback mechanisms, whose effects are illustrated in Figure 4. In the baseline `TypeFly`, user corrections are interpreted as new task requests rather than as persistent behavioral refinements. As a consequence, the system attempts to execute the feedback immediately within the current interaction, without retaining it for subsequent tasks. Although this can produce a temporary change in behavior, users rarely perceive these adjustments as useful over time, which explains the lower usefulness ratings shown in in Figure 4.
In contrast, responses associated with `DACS` concentrate in the higher range, indicating that feedback is perceived as producing stable and meaningful adaptation. Instead of executing corrections instantly, the `DACS` treats feedback as a persistent instruction that is stored and reintegrated into future planning processes. Interaction logs confirm that feedback is recognized as a refinement of interaction policy rather than as a new command, and is automatically reused when relevant conditions arise. This mechanism allows users to progressively shape interaction through natural-language refinements, transforming individual corrections into long-term behavioral changes.
Context Awareness. As illustrated in Figure 5, a central difference between `DACS` and the baseline `TypeFly` lies in the ability to reason over the visual surroundings of the drone using the captured images. Rather than relying solely on the objects currently detected in the scene by YOLO, `DACS` reasons about the visual

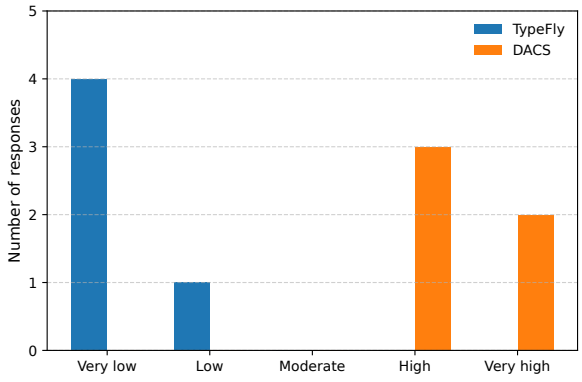

Figure 5: User-reported confidence in exploration behavior when the target is not immediately visible.

surroundings captured by the camera to decide where exploration should continue. During evaluation, correlated objects such as bananas and apples were placed in the environment: when a fruit was already visible, the planner prioritized nearby areas, inferring that semantically related objects were likely to be located there. In contrast, `TypeFly` lacks visual contextual reasoning; when the target is not detected, the drone typically continues rotating in place until manual human intervention stops the execution, making exploration ineffective.

Contextual reasoning is further reinforced through an environment graph that accumulates spatial knowledge over time. Planning traces show that the system can reuse previously observed information, for instance explicitly noting that apples were previously associated with `region_0` in the context graph and suggesting movement toward that region as a first step. This reuse of accumulated context transforms exploration from reactive scanning into experience-driven navigation, which aligns with the trends visible in Figure 5.

Latency and trade-offs. Adaptive reasoning introduces higher planning latency compared to `TypeFly`, particularly for requests involving memory integration, directional reasoning, or spatial constraints. However, user-reported disturbance due to waiting time remains lower overall for `DACS`. Participants primarily associate disruptive delays with ineffective execution behavior in the baseline rather than with planning time, suggesting that increased reasoning effort is acceptable when it results in more purposeful navigation.

This trade-off becomes clearer when considering the effect of high-level skill reuse. Tasks generated entirely from low-level primitives require on average around 41 s of planning time, whereas equivalent tasks executed through previously stored skills require roughly 20 s. Although the first execution introduces additional reasoning overhead due to skill synthesis, subsequent executions benefit from shorter and more stable planning phases, showing how adaptive abstractions progressively reduce reasoning effort over time.

5. Conclusion

We introduce `DACS` as an adaptive and context-aware HDI framework that extends LLM-driven systems beyond stateless task execution.

The evaluation shows that adaptive mechanisms enable users to shape system behavior over time through feedback and reusable skills. Overall, we demonstrate that combining LLM-based reasoning with persistent environmental and user knowledge shifts HDI toward a more adaptive, context-aware, and collaborative paradigm.

6. Bibliography

References

[1] Francesco Caserta. Towards drones as writable surfaces. 2023.

[2] G. Charness et al. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior Organization*, 2012.

[3] G. Chen et al. Typefly: Flying drones with large language model. 2023.

[4] S. Javaid et al. Large language models for uavs: Current state and pathways to the future. *OJVT*, 2024.

[5] A. Lykov et al. Cognitiveos: Large multimodal model based system to endow any type of robot with generative ai. In *ICRA*, 2025.

[6] M. Sondoqah et al. Shaping and being shaped by drones: Programming in perception-action loops. In *DIS*, 2024.