OPTIMISING UAV DYNAMICS: USER-CENTRIC LARGE LANGUAGE MODEL INTEGRATION FOR DYNAMIC ADAPTATION

Zeaus Koh Jin Rui[1], Zhang Ning An Serene[2], Lim En-lye Perrie[3], Lim Gang Le[3], Wong Rui Ming Jeremy[3],
[1] Hwa Chong Institution, 661 Bukit Timah Road, Singapore 269734
[2] Raffles Girls' School, 2 Braddell Rise, Singapore 318871
[3] Defence Science and Technology Agency, 1 Depot Rd, Singapore 109679

Abstract

The expanding drone market demands advanced programming for drones in dynamic environments. In areas with jammed radio signals, remote-controlled and pre-coded drones fail, especially in unpredictable scenarios like disaster response. This paper introduces a novel multi-agent adaptive Retrieval Augmented Generation (RAG) system, incorporating Large Language Models (LLMs) for enhanced functionality. It enables operators to use natural language commands, allowing drones to quickly adapt to varying situations.

Our approach integrates an adaptive RAG model, error correction modules, an inference engine, and a simulation component. It learns from Github repositories, with LLM agents correcting coding errors. Designed for compatibility with Airsim and Dronekit, the system is modular and secure for defense applications, easily integrating various LLMs.

Empirical tests with GPT-4-1106 demonstrate our system's superiority, achieving 44%, 73%, and 60% success rates in basic, operational, and advanced tasks, respectively, significantly outperforming GPT-4-1106. This marks a 15-fold improvement in task success, paving the way for practical drone use. Future efforts will focus on real drone testing, user interface enhancement, and increasing code processing speed to meet evolving drone operational needs.

Introduction and Background

In recent years, the exponential growth of drone technology has revolutionised industries worldwide, by keeping humans away from dull, dirty and dangerous tasks, evolving from a niche application to a key enabler of many industries. The mission effectiveness of teleoperated drones today hinges on pilot skill or the ability of the drone to execute a limited range of automated tasks (or machine states) along human-designated waypoints [1]. This limits the deployment of autonomous drones to benign missions, and precludes usage in dynamic, contested environments like disaster zones or conflict areas. In the realm of drone-based search and rescue or security missions, open challenges persist in deploying drones that are able to adapt to dynamically challenging environments where swift, unpredictable situations with minimal or no intervention [2]. However, no reliable model for coding drones' behaviours to accomplish novel critical tasks in these environments without prior training exists, hence driving a strong impetus for this project through exploratory development at the nexus of robotic control and advances in Large Language Models (LLMs).

The concept of LLMs has only recently begun to spur wide-ranging applications. However, such technology is primarily a generalist and "hallucinates" in specialised fields [3], is not up-to-date, and even those providing near up-to-date data like Bard AI still suffer from relevance and reliability issues [4] due to data imbalances surrounding specialised, reliable information online being outnumbered by erroneous responses from netizens. In our work, we address these shortcomings identified through an adaptive Retrieval Augmented Generation (RAG), a novel context-specific approach via self-training among other curated changes. This aimed to solve our problem by optimising adaptability, accuracy, speed, and user-friendliness all while being storage-efficient for field deployment.

We hypothesise that our multi-agent adaptive RAG model will outperform existing LLMs like ChatGPT in responding reliably to dynamic scenarios with simple natural language prompts, thereby addressing the need for a more precise and context-relevant solution. The code for our project can be found in this repository[1]. To generate a proof of concept and empirically benchmark our system, we use a direct comparison between our model and another powerful LLM, GPT-4-1106.

[1] https://github.com/zeauskoh/llm-drone-pipeline

Methodology

Algorithm Development:

Our system consists of 1 main and 4 assisting segments. The adaptive Retrieval Augmented Generation (RAG) system, 2 error correcting modules, an inference module and a simulation segment.

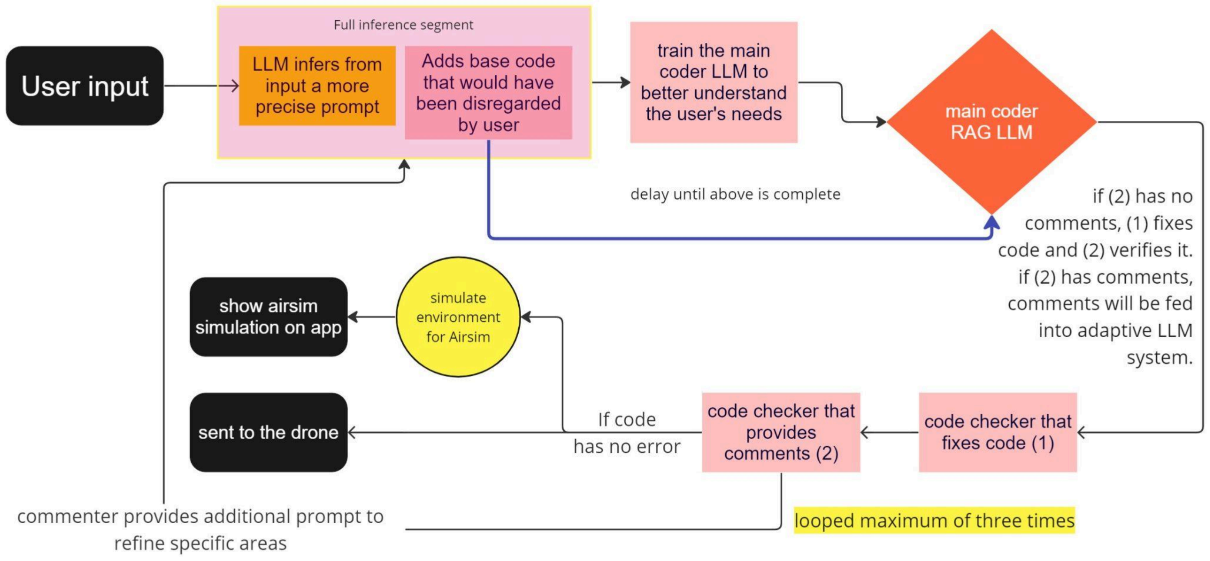
User input
Full inference segment
LLM infers from input a more precise prompt
Adds base code that would have been disregarded by user
train the main coder LLM to better understand the user's needs
main coder RAG LLM
delay until above is complete
if (2) has no comments, (1) fixes code and (2) verifies it. if (2) has comments, comments will be fed into adaptive LLM system.
show airsim simulation on app
simulate environment for Airsim
sent to the drone
If code has no error
code checker that provides comments (2)
code checker that fixes code (1)
looped maximum of three times
commenter provides additional prompt to refine specific areas

Fig 1: Algorithm flow chart

It is important to note that our system is modular in nature and can be swapped with a module for use in an intranet for more secure matters. For example, the use of SerpAPI for querying Google is mentioned as testing on the internet with Google is similar to an intranet in the logical pathway and allows for these research results to remain unclassified. The base LLM used and tested for this system is GPT-4-1106. Similarly, this is also modular in nature and can easily be swapped with any other LLM available including local LLMs if data sensitivity is a concern. Our test, having a baseline comparison with the LLM it originates from (GPT-4-1106), allows us to prove the inherent improvement of our system based on a percentage success rate comparison.

Including "Common Sense Code":

To enhance user-friendliness and tackle the high specificity in instructions required in LLM-produced code [5], our system employs a two-part solution. First, an LLM agent interprets user input into a more precise prompt for the main coding LLM. Second, we integrate hidden prompts for common drone operations like crash avoidance, ensuring essential functions are included even if omitted by the user. This is enabled by the positioning of the hidden prompt in question to give a disproportionate weightage favouring the input.

Adaptive RAG System:

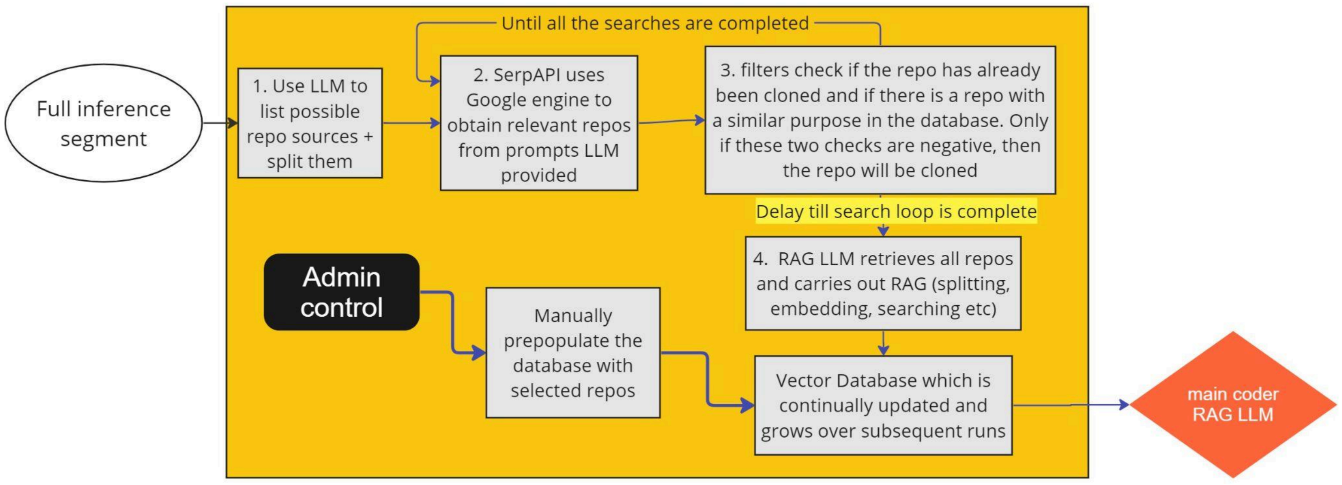
Full inference segment
1. Use LLM to list possible repo sources + split them
Until all the searches are completed
2. SerpAPI uses Google engine to obtain relevant repos from prompts LLM provided
3. filters check if the repo has already been cloned and if there is a repo with a similar purpose in the database. Only if these two checks are negative, then the repo will be cloned
Delay till search loop is complete
4. RAG LLM retrieves all repos and carries out RAG (splitting, embedding, searching etc)
Admin control
Manually prepopulate the database with selected repos
Vector Database which is continually updated and grows over subsequent runs
main coder RAG LLM

Fig 2: Adaptive RAG model training flowchart

Our adaptive RAG system (Fig 2) is tailored to address the relevance issue in traditional RAG databases [6]. If a database is prepopulated with an extensive set of files, the abundance of irrelevant files results in a reduced response accuracy [7]; if the data set is prepopulated with only a specially curated set of files, its adaptability becomes limited with its performance dropping in anything outside its designed scope [8]. Our model captures the strength of RAG whilst mitigating its limitations: it starts with an initial database which we intentionally prepopulated to cover basic drone manoeuvres and capabilities, then actively downloads relevant GitHub repositories based on further user queries to grow its database.

First, an LLM agent takes the processed user input and outputs a string of repositories relevant to the query, which is then split into individual queries sent to Google searching API, SerpAPI [8]. The algorithm then obtains GitHub repositories in the top 10 search results and downloads them in one file. Several filters are applied, the first checking for duplication in cloned repositories to eliminate storage issues, and the second checking if a repository with a similar function, such as infrared sensor integration, exists in the database. Through a cosine similarity test comparing the ReadME of the repository with the descriptions of those in our database, the repository will only be cloned upon passing both filters.

From here, the model works like a traditional RAG model [8], using OpenAI's text embedding model for vector database embedding. AutoLLM [9] was piloted as our RAG model as we can easily optimise the model's parameters for our coding use case.

Error Correction:

Error correction is critical for code generated by LLMs. We mitigate errors through a two-tiered approach: one LLM agent corrects syntax errors, while another comments on logical issues, ensuring the output is both syntactically sound and logically consistent. If the first agent outputs an accurate code, it is used as the final code. Otherwise, the second agent (commenter) will provide comments in the form of an appended prompt fed back into the RAG to refine the script. This is designed to reduce logical discontinuities within the code.

Simulation:

For simulation compatibility, our system works with both Airsim and Dronekit codes, featuring a converter for seamless integration. Each specialised LLM agent within our network is fine-tuned through prompt engineering, utilising unique system messages and hidden prompts to guide operations effectively.

Testing Framework:

We focused primarily on the AirSim simulator for our tests which is widely recognised for its drone trust tests and realistic environments [10].

However, as LLM integration with unmanned aerial vehicles (UAV) is an emerging domain, there are no widely accepted benchmarking frameworks to evaluate such a pipeline. To generate a proof of concept, we developed a testing framework to demonstrate its reliability and evaluate its task accuracy, closely mirroring real UAV mission requirements. This testing framework is categorised into three stages: first, a set of nine tests of basic UAV capabilities (covers basic manoeuvres like determining local North, takeoff and landing); second, a set of 11 tests involving operational requirements within contested environments (which requires computation and path planning such as flying in an equilateral triangle formation); and third, a set of five advanced tests which involved technical requirements for challenging operations (specifically obstacle detection and avoidance). The first and second stages are aligned with the Singapore Unmanned Aircraft Pilot License practical test requirements [11]. For the third stage, we manually set up blocks directly in the drone's flight path which required obstacle detection and avoidance.

During testing, we benchmarked our system against GPT-4-1106, a leading LLM exhibiting near-general intelligence, surpassing prior models in difficult tasks and showcasing near-human-level performance [12]. As GPT-4-1106 is a part of our pipeline, this comparison is essential in testing our hypothesis that an adaptive RAG system and multiple integrated agents bring significant improvement in performance. All tests were designed to be adaptive, meaning our system was not pre-trained or fine-tuned. To ensure a fair evaluation of solely programming logic, we corrected syntax errors in the codes generated by both our pipeline and the benchmark, GPT-4-1106.

For each test, we crafted natural language commands that specified parameters like altitude and speed of the drone. We repeated each test three times for both set-ups. Results are classified as "Successfully Complete", "Partially Complete", or "Fail", and results are graphically analysed for a clear comparison. The list of test objectives and accompanying natural language commands can be found in the Annex.

Results

In this section, we compare results between tests on our system and on GPT-4-1106. For each set of results, we converted them into percentages on a stacked bar chart, with each category representing Complete Success, Partial Success and Fail. Complete Success cases refer to instances when the drone in AirSim finishes the assigned mission correctly. Partial Success is defined as instances when the drone in AirSim executes part of the assigned mission but deviates or stops. Failure cases refer to instances when the drone in AirSim either does not take off, or takes off but does not execute any part of the assigned mission.

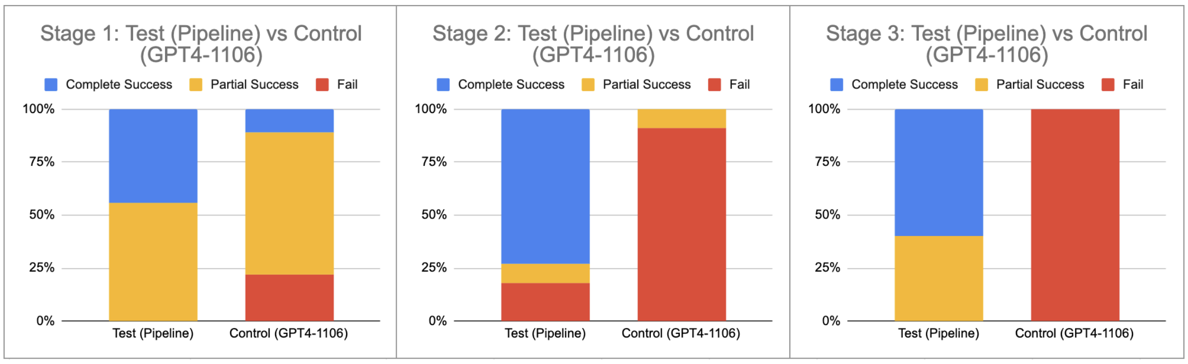

Fig 3: Column charts showing levels of success by percentage; separated by stages

Fig 3 represents the results by Stages of our testing. Stage 1, or basic capabilities, consisted of 9 tests run thrice each, separately for test and control. As shown, across 27 tests of basic capabilities, our pipeline partially succeeds approximately 56% of the time and completely succeeds 44% of the time. In comparison, GPT-4-1106 fails 22% of the time, partially succeeds 67% of the time and completely succeeds 11% of the time.

Stage 2, or operational requirements, consisted of 11 tests run thrice each, separately for test and control. Across 33 tests, our pipeline fails approximately 18% of the time, partially succeeds 9% of the time, and completely succeeds 73% of the time. In contrast, GPT-4-1106 fails 91% of the time and only partially succeeds 9% of the time.

Stage 3, advanced technical requirements, consisted of 5 tests run thrice each, separately for test and control. As depicted, across 15 tests, our system partially succeeds 40% of the time and completely succeeds 60% of the time. In contrast, the drone following the code generated by GPT-4-1106 failed in completing the mission for all 15 runs.

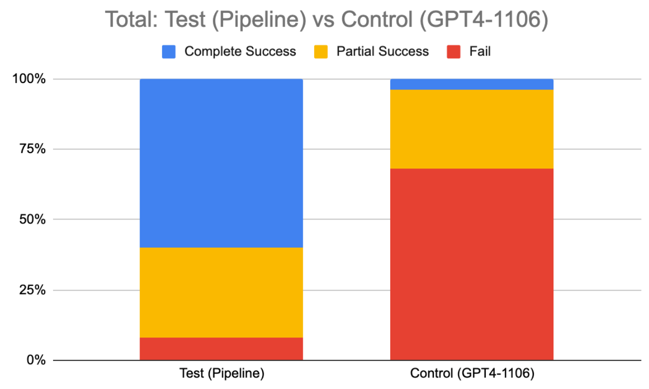

Fig 4: Level of mission success as a proportion of tests; overall

Fig 4 displays the total successes, partial successes and failures across all Stages of testing. It helps us visualise the overall performance differences between our pipeline and GPT-4-1106. Notably, our pipeline completely succeeded 60% of the time, whereas GPT-4-1106 only completely succeeded 4% of the time. Our pipeline partially succeeded 32% of the time, while GPT-4-1106 partially succeeded 28% of the time.

Discussion of Results and Implications

Our testing framework categorised UAV requirements which are success metrics for drone operations into three stages: basic manoeuvrability, operational requirements and advanced technical capabilities. Across these three stages, our pipeline significantly outperformed GPT-4-1106. In terms of complete success, our system is 15 times as effective as GPT-4-1106. This is a strong indicator of the accuracy of the code our pipeline generated, even with respect to a powerful LLM like GPT-4, proving our hypothesis that the RAG system and multiagent setup are highly effective.

In Stage 1, our pipeline had more partial successes, such as flying in L-shape and U-shape formations, than complete successes, where the drone did not fully complete either formation and deviated off course. As our pipeline allows for administrative control over the vector database, these errors can be eliminated by adding the right repositories for turning a drone through such formations. Nevertheless, it still performed better than GPT-4-1106 which failed 22% of the time compared to 0% in our pipeline. The code generated by GPT-4-1106 lacked the right preflight commands needed to arm the drone which is a critical flaw, given that Stage 1 consists only of basic capabilities that a UAV should have in order to fly.

In Stage 2, which focuses on operational requirements such as path-plotting, our pipeline outperformed GPT-4-1106 by a large margin, reducing the failure rate by 73 percentage points. Further, our pipeline successfully completed its assigned task 73% of all tests. This is again a positive sign that our pipeline can generate accurate code for operational requirements. For instance, Test 17 commanded the drone to move in an equilateral triangle of length 5 metres. Our system accurately calculated the angles for turning the drone through the manoeuvre and completed a perfect triangle. The flight path recorded can be found in Fig 5, where our pipeline completely succeeded while GPT-4 failed.

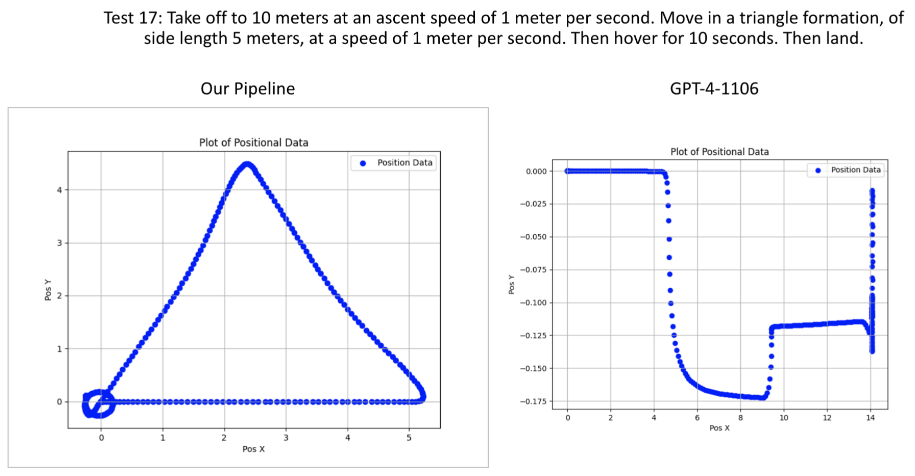

Fig 5: Flight path for the multirotor operating under GPT-4-1106 code versus our pipeline's code in Test 17

These findings suggest that if this pipeline was applied to real-world environments, non-technical drone operators can use simple natural language prompts to accurately direct drones in whichever path they specify. A further example of this would be Test 19, in which the drone was instructed to fly in a figure of 8. No further instructions concerning how exactly the drone was to plot its path or make a smooth continuous turn were given, nor was there a relevant repository in the initial database. Yet, the system was able to generate the following code, which successfully flew a figure of 8 in AirSim.

Sample Drone Code 1 Test 19 (Fly in a Figure of 9) | The following code is generated by our pipeline

```
import airsim
import time
import math
TAKEOFF_ALTITUDE = 10  # Target altitude for takeoff in meters
ASCENT_SPEED = 1       # Ascent speed in meters per second
FLYING_SPEED = 5       # Flying speed in meters per second
RADIUS = 20            # Radius of each circle in the figure of 8 in meters
HOVER_DURATION = 10    # Duration to hover in seconds
client = airsim.MultirotorClient()
client.confirmConnection()
client.enableApiControl(True)
def takeoff(target_altitude, ascent_speed): # Function to take off to a certain altitude at a given speed
    landed = client.getMultirotorState().landed_state
    if landed == airsim.LandedState.Landed:
        print("Taking off...")
        client.takeoffAsync().join()
        client.moveToZAsync(-target_altitude, ascent_speed).join()
    else:
        print("Drone is already airborne.")
def fly_figure_of_eight(radius, speed): # Function to fly in a figure of 8 pattern
    waypoints = []
    for angle in range(0, 720, 10):  # Two circles, each 360 degrees
        rad = math.radians(angle)
        if angle <= 360:
            x_val = radius * math.sin(rad)
            y_val = radius * (1 - math.cos(rad))
        else:
             x_val = radius * math.sin(rad)
            y_val = radius * (-1 + math.cos(rad))
        waypoints.append(airsim.Vector3r(x_val, y_val, -TAKEOFF_ALTITUDE))
    for waypoint in waypoints:
        client.moveToPositionAsync(waypoint.x_val, waypoint.y_val, waypoint.z_val, speed).join()
def hover(duration): # Function to hover for a specified duration
    print("Hovering...")
    time.sleep(duration)
def land(): # Function to land the drone
    print("Landing...")
    client.landAsync().join()
    client.armDisarm(False)  # Disarm the drone motors
    client.enableApiControl(False)  # Release control of the drone
def perform_mission(): # Function to perform the mission
    try:
        takeoff(TAKEOFF_ALTITUDE, ASCENT_SPEED)
        fly_figure_of_eight(RADIUS, FLYING_SPEED)
        hover(HOVER_DURATION)
        land()
    except Exception as e:
        print(f"An error occurred: {str(e)}")
        land()
perform_mission() # Execute the mission
```

The flight paths of our pipeline versus GPT-4-1106 for Test 19 is shown in Fig 6. Our pipeline completely succeeded in flying a figure of 8, while GPT-4 failed.

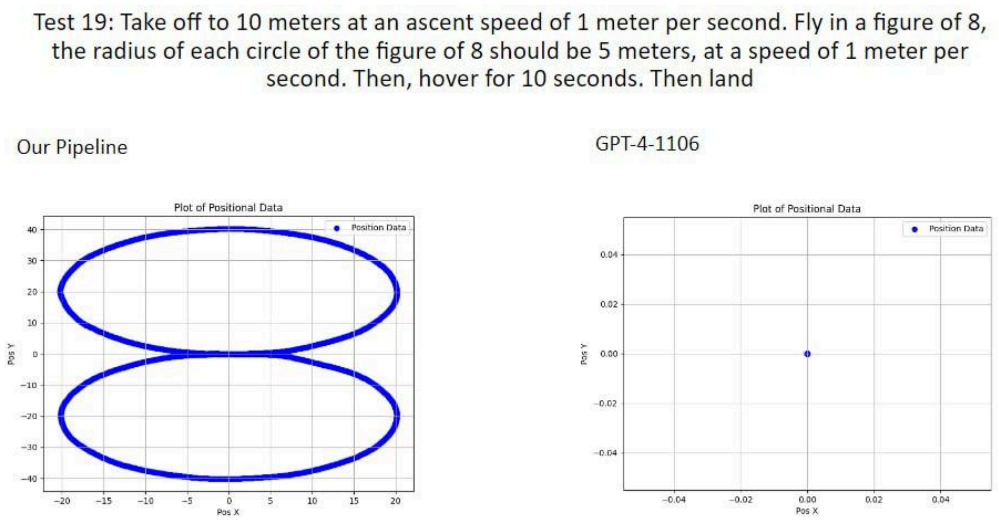
Test 19: Take off to 10 meters at an ascent speed of 1 meter per second. Fly in a figure of 8, the radius of each circle of the figure of 8 should be 5 meters, at a speed of 1 meter per second. Then, hover for 10 seconds. Then land

Fig 6: Flight path for the multirotor operating under GPT-4-1106 code versus our pipeline's code in Test 19

Another example of the multirotor flight path for both test and control set-ups for Test 15 (flying in a circle) is shown in Fig 7. Our pipeline completely succeeded while GPT-4-1106 failed.

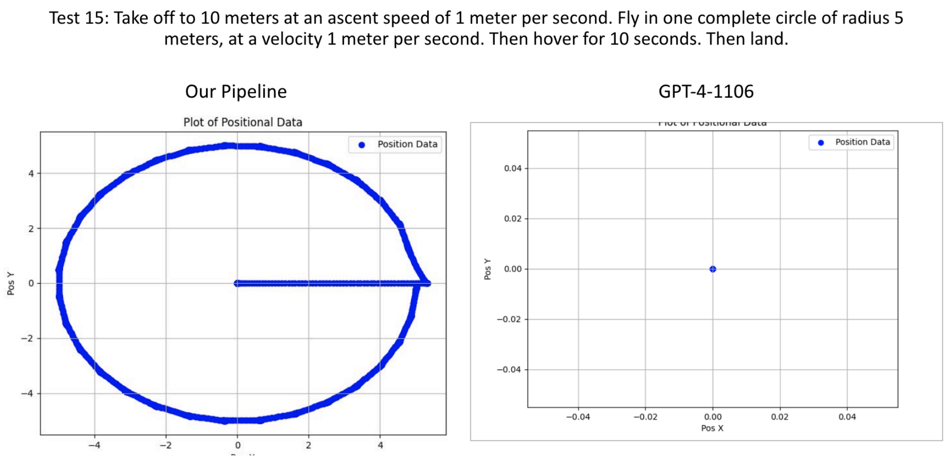
Test 15: Take off to 10 meters at an ascent speed of 1 meter per second. Fly in one complete circle of radius 5 meters, at a velocity 1 meter per second. Then hover for 10 seconds. Then land.

Fig 7: Flight path for the multirotor operating under GPT-4-1106 code versus our pipeline's code in Test 15

In Stage 3, which involved more technical advanced skills such as obstacle avoidance and complex navigation, our pipeline again fared much better than GPT-4-1106 which failed every single run, while our system completely succeeded in 9 out of 15 runs with a 0% failure rate. The system intelligently searched for relevant repositories like the Bug2 algorithm, an efficient path-planning, obstacle-avoidance method [13] to be used as additional context for generating drone code, heightening its success rate. This proves the concept of a truly adaptive system combined with a multi-agent setup, effective in enhancing code accuracy for complex tasks. These results confirm the robustness of our approach, indicating a significant advancement in drone programming for dynamic and demanding environments.

Conclusion

In conclusion, through this project, we have developed an end-to-end pipeline that takes in natural language commands from a non-technical drone operator and generates drone code that succeeds in directing the multirotor within AirSim to fulfil the given objective. Our benchmarking against GPT-4-1106 further proved the concept of our multi-agent adaptive RAG system. Specifically, we have proven our initial hypotheses that adopting a multi-agent setup, with each LLM agent prompt-engineered for a specific function, and that using an adaptive RAG system to retrieve relevant repositories of drone code as additional context, can significantly improve the accuracy and reliability of the drone code generated, with respect to existing powerful LLMs such as GPT-4-1106. Such a novel approach would prove useful for modern homeland security use-cases, involving dynamic adaptation and quick turnaround time without over-reliance on operator skill, or availability of skilled pilots. Further development of our pipeline could see such a system operationalised as a key tool in drone technologies, contributing profound implications to defence and security.

Future Work

Currently, our algorithm is at a Technology Readiness Level (TRL) [14] of TRL 4. In order to push this technology further towards real-world use, real drone testing needs to be conducted, rather than virtually simulated. Such testing was intended and will be carried out but not within the timeline of this report. Instead, experimental verification to get our algorithm to TRL 5-7 would be done as future work.

In addition, the user interface is to be developed, whereby a simulation of the drone code is shown as a way for the user to verify accuracy before it is uploaded to the UAV. Most importantly, the code processing duration is to be further shortened. Our current code takes approximately 30 minutes to run each time. Although this length is shorter than the time generally taken to set up a drone for flight and hence is already viable, it would be ideal to increase the time efficiency of our algorithm to future-proof our algorithm with the decreasing set-up duration that drone systems appear to be heading towards.

Acknowledgements

We would like to thank our awesome mentors: Perrie, Gang Le, Jeremy and David. Together they've supported us through our YDSP journey, since our very first onboarding to the day we offboarded. To fellow interns Benedict Lee Rui Yang and Jonathan, thank you for your hard work and dedication which made this project possible.

Annex

Annex 1: Table of tasks and natural language commands used in our testing framework

Test	Task Description	Natural Language Command
1	Hover and Land [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second, then hover in place for 15 seconds. Land the drone at a descent speed of 1 metre per second.
2	Moving for a set amount of time [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward in a straight line for 10 metres at a forward speed of 1 metre per second. Land the drone at a decent speed of 1 metre per second.
3	Moving and then returning to origin point [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward in a straight line for 10 metres at a forward speed of 1 metre per second. Then return to origin and land.
4	Basic Motions [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward by 10 metres at a forward speed of 1 metre per second. Then turn 90 degrees to the right, and move forward by 10 metres at a speed of 1 metre per second. Then land.
5	Basic Motions [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward by 10 metres at a forward speed of 1 metre per second. Then turn to face the local east, and move forward by 10 metres at a speed of 1 metre per second. Then land.
6	Basic Motions [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward by 10 metres at a forward speed of 1 metre per second. Then turn 90 degrees to the left, and move forward by 10 metres at a speed of 1 metre per second. Then land.
7	Basic Motions [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward by 10 metres at a forward speed of 1 metre per second. Then turn to local west, and move forward by 10 metres at a speed of 1 metre per second. Then land.
8	Basic Motions [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward by 10 metres at a forward speed of 1 metre per second. Then turn to face the local east, and move forward by 10 metres at a speed of 1 metre per second. Then turn to face local south, and move forward by 10 metres at a speed of 1 metre per second. Then land.

9	Basic Motions [Stage 1]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward by 10 metres at a forward speed of 1 metre per second. Then turn to face local west, and move forward by 10 metres at a speed of 1 metre per second. Then turn to face local south, and move forward by 10 metres at a speed of 1 metre per second. Then land.
10	Moving in a Square [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. Move in a square of side length 10 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.
11	Moving in a Square [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. While constantly facing local north, move in a square formation where each side of the square is 10 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.
12	Moving in a Rectangle [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. Move in a rectangle, where the length is 10 metres and the breadth is 5 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.
13	Moving in a Rectangle [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. While constantly facing local north, move in a rectangle formation, where the longer length is 10 metres and the shorter breadth is 5 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.
14	Moving in a Square, followed by a rectangle [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. While facing local north, move in a square of side length 5 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then while facing local north, move in a rectangle, where the length is 10 metres and the breadth is 5 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.
15	Moving in a circle [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. Fly in one complete circle of radius 5 metres, at a velocity 1 metre per second. Then hover for 10 seconds. Then land.
16	Moving in a circle [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. While facing local north, move in a complete circle of radius 5 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.
17	Moving in a triangle [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. Move in a triangle formation, of side length 5 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.

18	Moving in a triangle [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. While facing local north, fly in a triangle formation, of equal length 5 metres, at a speed of 1 metre per second. Then hover for 10 seconds. Then land.
19	Moving in a figure 8 [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. Fly in a figure of 8, the radius of each circle of the figure of 8 should be 5 metres, at a speed of 1 metre per second. Then, hover for 10 seconds. Then land.
20	Moving in a figure 8 [Stage 2]	Take off to 10 metres at an ascent speed of 1 metre per second. While facing local north, move in a figure of 8, the radius of each circle of the figure of 8 should be 5 metres, at a speed of 1 metre per second. Then, hover for 10 seconds. Then land.
21	Moving to a point while avoiding one object [Stage 3]	Take off to 10 meters at an ascent speed of 1 metre per second. Your destination is 30 metres local north in your current reference frame. Fly at a speed of 1 metre per second, making sure to avoid any obstacles in your path to reach the destination.
22	Moving to a point while avoiding two objects [Stage 3]	Take off to 10 meters at an ascent speed of 1 metre per second. Your destination is 30 metres local north in your current reference frame. Fly at a speed of 1 metre per second, making sure to avoid any obstacles in your path to reach the destination.
23	Moving to and from a point, avoiding obstacles [Stage 3]	Take off to 10 meters at an ascent speed of 1 metre per second. Your destination is 30 metres local north in your current reference frame. Fly at a speed of 1 metre per second, making sure to avoid any obstacles in your path to reach the destination. After this, return to the origin point, making sure to avoid any obstacles.
24	Moving in a square formation, avoiding obstacles [Stage 3]	Take off to 10 metres at an ascent speed of 1 metre per second. Move in a square of side length 10 metres, at a speed of 1 metre per second. If you encounter obstacles, avoid it and return back to the original square pathway. After completing the square, hover for 10 seconds and then land.
25	Moving forward, encircling obstacle, then return [Stage 3]	Take off to 10 metres at an ascent speed of 1 metre per second. Move forward until you are 1 metre from the obstacle at a speed of 1 metre per second. Encircle the obstacle at current height before returning to origin point and landing.

References

[1] J. Morgan, J. Perez, J. Wade, and S. Krishnan, “Security in Drones,” arXiv [cs.CR], 2023 (accessed Jan. 1, 2024).

[2] K. Okada and E. Hayakawa, “Flow-based ROS2 programming environment for control drone,” in Communications in Computer and Information Science, Cham: Springer International Publishing, 2020, pp. 449–453 (accessed Jan. 1, 2024).

[3] M. Bilan, “Hallucinations in LLMS: What you need to know before integration,” Master of Code Global, https://masterofcode.com/blog/hallucinations-in-llms-what-you-need-to-know-before-integration (accessed Jan. 1, 2024).

[4] T. Lacoma and S. Winkelman, “Google Bard explained: What this AI-powered ChatGPT competitor can do,” Android Police, 08-Feb-2023. [Online]. Available: https://www.androidpolice.com/google-bard-explained/. (accessed Jan. 1, 2024).

[5] R. Carter, “How to talk to an LLM: Prompt engineering for beginners,” UC Today, 24-Nov-2023. [Online]. Available: https://www.uctoday.com/unified-communications/how-to-talk-to-an-llm-llm-prompt-engineering-for-beginners/. (accessed: Jan. 1, 2024).

[6] “lancedb/README.md at main · lancedb/lancedb,” GitHub. https://github.com/lancedb/lancedb/blob/main/README.md (accessed Jan. 1, 2024).

[7] “Retrieval Augmented Generation (RAG),” Cohere AI. [Online]. Available: https://docs.cohere.com/docs/retrieval-augmented-generation-rag. (accessed Jan. 1, 2024).

[8] “What is retrieval-augmented generation (RAG)?,” Oracle.com, 19-Sep-2023. [Online]. Available: https://www.oracle.com/sg/artificial-intelligence/generative-ai/retrieval-augmented-generation-rag/. (accessed Jan. 1, 2024).

[9] Safevideo, “Safevideo/autollm: Ship rag based LLM Web Apps in seconds.,” GitHub, https://github.com/safevideo/autollm (accessed Jan. 1, 2024).

[10]“Google Search Results in Python,” GitHub, Jan. 02, 2024. https://github.com/serpapi/google-search-results-python (accessed Jan. 2, 2024).

[11] “UA Pilot Licence,” CAAS - CWP. https://www.caas.gov.sg/public-passengers/unmanned-aircraft/ua-regulatory-requirements/ua-pilot-licence (accessed Jan. 1, 2024).

[12] S. Bubeck et al., “Sparks of Artificial General Intelligence: Early experiments with GPT-4,” arXiv [cs.CL], 2023.(accessed Jan. 1, 2024).

[13] MehdiShahbazi, “Mehdishahbazi/AirSim-multirotor-bug2-algorithm: Python implementation of Bug2 algorithm to navigate a quadcopter/multirotor in the AirSim simulator.,” GitHub, https://github.com/MehdiShahbazi/AirSim-Multirotor-Bug2-Algorithm (accessed Jan. 2, 2024).

[14] “What are Technology Readiness Levels (TRL)?,” Twi-global.com. [Online]. Available: https://www.twi-global.com/technical-knowledge/faqs/technology-readiness-levels. (accessed Jan. 1, 2024).