REAL: Resilience and Adaptation using Large Language Models on Autonomous Aerial Robots

Andrea Tagliabue*
LIDS
MIT
atagliab@mit.edu

Kota Kondo*
LIDS
MIT
kkondo@mit.edu

Tong Zhao*
EECS
MIT
tzhao@mit.edu

Mason Peterson*
LIDS
MIT
masonbp@mit.edu

Claudius T. Tewari
LIDS
MIT
cttewari@mit.edu

Jonathan P. How
LIDS
MIT
jhow@mit.edu

Abstract: Large Language Models (LLMs) pre-trained on internet-scale datasets have shown impressive capabilities in code understanding, synthesis, and general purpose question-and-answering. Key to their performance is the substantial prior knowledge acquired during training and their ability to reason over extended sequences of symbols, often presented in natural language. In this work, we aim to harness the extensive long-term reasoning, natural language comprehension, and the available prior knowledge of LLMs for increased resilience and adaptation in autonomous mobile robots. We introduce **REAL**, an approach for **RE**silience and **A**daptation using **L**LMs. REAL provides a strategy to employ LLMs as a part of the mission planning and control framework of an autonomous robot. The LLM employed by REAL provides (i) a source of prior knowledge to increase resilience for challenging scenarios that the system had not been explicitly designed for; (ii) a way to interpret natural-language and other log/diagnostic information available in the autonomy stack, for mission planning; (iii) a way to adapt the control inputs using minimal user-provided prior knowledge about the dynamics/kinematics of the robot. We integrate REAL in the autonomy stack of a real multirotor, querying onboard an offboard LLM at 0.1-1.0 Hz as part the robot's mission planning and control feedback loops. We demonstrate in real-world experiments the ability of the LLM to reduce the position tracking errors of a multirotor under the presence of (i) errors in the parameters of the controller and (ii) unmodeled dynamics. We also show (iii) decision making to avoid potentially dangerous scenarios (e.g., robot oscillates) that had not been explicitly accounted for in the initial prompt design.

Keywords: LLMs, Adaptive Control, Aerial Robotics

1 Introduction

Creating mission planning and control capabilities that are adaptive and resilient to unexpected scenarios has been a large area of research in recent years. Adaptive control has enabled exceptional performance when addressing specific failure modes, such as disturbances [1, 2, 3], incorrect models/parameters [4, 5, 6], or poor controller tuning [7, 8, 9]. However, these approaches work best under a pre-defined set of failure modalities, and/or leverage accurate models/prior knowledge about the robot from the designer. Similarly, complex missions for autonomous mobile robots have been successfully managed through sophisticated state machines and mission planners [10, 11, 12, 13, 14, 15]. However, these planners often need to reason over a pre-defined set of states and/or observation models, identified through extensive efforts.

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

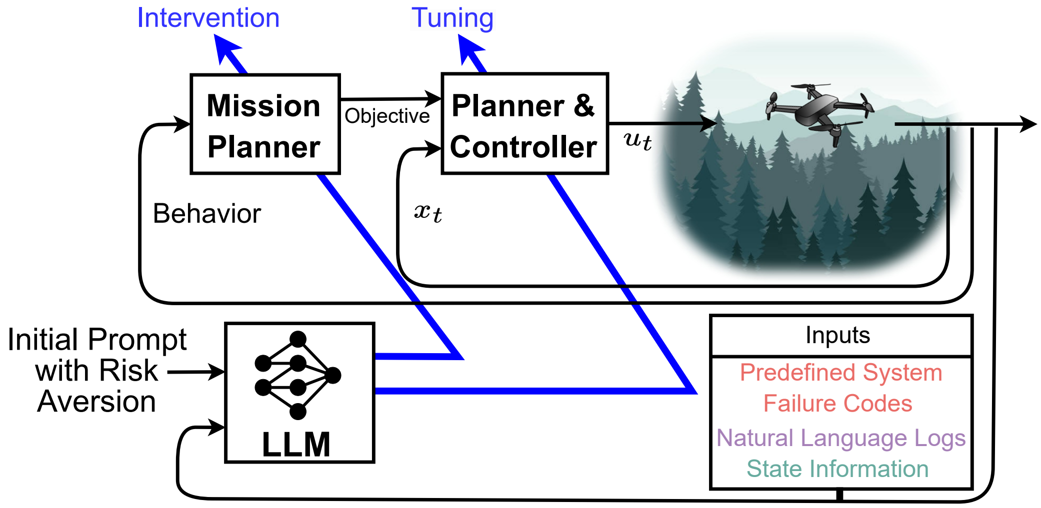


Figure 1: Schematic representation of our approach. We use a Large Language Model (LLM) to achieve adaptation from low-level to mission-scale decision-making, enabling resilience across the different dynamics and components of our autonomous system. Despite the ability of the LLM to apply changes to the low-level control of the multirotor, the LLM does not need to know specific details about the autonomy stack of the platform, instead using embedded prior knowledge to make decisions. In experiments, we demonstrate that an LLM queried from onboard the robot can reason about the current and desired state of the robot, deciding which corrective actions to apply to the control input to achieve the desired objective.

Recently, foundational models, and especially Large Language Models (LLMs) pre-trained on internet-scale datasets [16, 17, 18], have demonstrated impressive performance on a variety of reasoning problems, including natural language [19, 20] and mathematics [21]. This performance stems in part from the large size of their training data (e.g., internet-scale), which embeds a vast amount of prior knowledge into the weights of the model. Additionally, their billion-of-parameters model architectures enable reasoning over long sequences of symbols, causing them to be a natural choice for any problem that involves generating a sequence of symbols. The embedded prior knowledge and extended sequential reasoning capabilities have led to LLMs finding increased application in task planning and motion planning for robotics. In this context, the main focus of recent work has been planning for manipulation [22, 23, 24, 25, 26, 27, 28], using human input as a task specification and outputting calls to manipulator APIs. However, their potential has not been explored for *combined adaptive low-level control* and mission *planning/reasoning* on agile autonomous aerial robots.

In this paper, we present REAL (Resilience and Adaptation using LLMs), a method for harnessing the capabilities of LLMs for mission-planning and low-level adaptive control of an agile mobile robot, a multirotor UAV. Our work leverages the LLM's embedded prior knowledge of the UAV's dynamics to create adaptation throughout the stack, including altering low-level parameters, producing commands to better track trajectories, and making mission-level decisions. REAL uses a single human-crafted prompt (zero-shot prompting) to define minimal robot specification and task/controller API available to the LLM. Then, during real-time deployment, REAL receives as input a set of natural-language and numerical signals available onboard the multirotor which capture mission-relevant information at different timescales, including information about the dynamics of the robot and its high-level mission objectives. Then, based on these automatically generated robot prompts, REAL chooses the most suitable control/mission planning APIs that are executed by the robot. This feedback loop operates at about 0.1-1.0 Hz, while the prompts are processed remotely using the `OpenAI GPT-4 API`.

We evaluate REAL in hardware experiments, exposing our multirotor to a variety of performance-lowering conditions. Some of these conditions require low-level adaptation (e.g., by adjusting the commanded thrust), while others which require mission-level adaptation (e.g., by improving controller tuning or conducting an emergency landing). Through these experiments, we also show that the behavior of an LLM as an adaptive controller can be modified by the use of natural language cues (e.g. using stronger language when the instructions given to the LLM are safety-related and important to follow). Additionally, we show that although the LLM cannot be queried at a high rate (up to 1 Hz), it can still process and make suggestions in response to high-frequency information by making use of algorithmically pre-processed information.

Contributions:

- We present REAL, an approach to leverage prior knowledge in LLMs to enable online adaptation and decision-making across different time scales and components (*low-level controller*, *mission planner*) of the autonomy stack of an aerial robot. We leverage zero-shot prompting, and we show that our prompt requires minimal knowledge of the robot's model/dynamics and mission specifications.
- We present hardware experiments, demonstrating adaptation and decision-making capabilities using LLMs that improve the position control performance of the robot or regulate the safety of the mission. To the best of our knowledge, this is the first time that such capabilities have been demonstrated on an *aerial* robot.

2 Related Works

2.1 Adaptation at Mission-Scale and Low-Level Control

Adaptive Control. There are two broad categories of methods used for adaptive control: direct and indirect methods. Indirect methods aim at explicitly estimating models or parameters, which are leveraged in model-based controllers, such as MPC [29], to improve performance. Model/parameter identification include filtering techniques [30, 31], disturbance observers [32, 33, 34], set-membership identification methods [35, 36] or learning-based methods [5, 37]. Direct methods, instead, develop policy updates that improve a certain performance metric. These updates are often done to drive the behavior of the system towards that of a reference model, with the updates themselves involving changing the shallow layers of the DNN policy [1, 38, 39]. Other strategies include learning a policy update strategy offline using meta-learning [2, 40], or using parametric adaptation laws such as $\mathcal{L}_1$ adaptive control [3]. While many adaptive control strategies are able to improve low-level performance in real-world systems, these strategies often fail when mission-level adaptation is required. Our work provides the first example of a system that exhibits both low-level and mission-level adaptation.

Uncertainty-Aware Mission Planning. Mission-level adaptation is usually achieved with robot autonomy. State-of-the-art approaches to autonomy have involved the use of finite-state machines and uncertainty-aware planners [13, 15, 41], enabling autonomy on many systems, from a single autonomous car [10] to multiple heterogeneous robots [11, 12]. While these methods achieve impressive performance in the coordination of multiple autonomous systems, they do not leverage the internet-scale prior knowledge in LLMs that may be helpful in making decisions under natural language-based observations that are available at the system level (e.g., log), nor they require to specify observations models/mission states.

2.2 Foundational Models in Robotics

Foundational models have quickly found a variety of applications in robotics, with a focus on planning from natural language instructions. [42] develop a holistic foundational model that performs perception, planning, and control using internet-scale datasets to train a multi-modal foundational model that, given a goal described in natural language, can use video feed to plan and execute a sequence of commands to achieve that goal. [22] decodes an LLM weighted by skill affordances [43] from value functions to generate feasible plans for robots. [23, 24, 25] all translate a high-level instruction into a plan expressed in code, which is then executed by the robot. [26] uses an LLM to translate a natural language planning problem into a domain-specific language, then runs a classical planner to solve the problem. [28] uses an LLM to generate a plan in natural language, then uses a similarity measure to translate the plan from natural language into one executable by the robot. [27] uses closed-loop environmental feedback to improve the performance of using an LLM for planning and control in manipulation tasks. While existing methods have focused on task-level planning (especially for manipulation), our work leverages LLMs for combined mission management and low-level control on an agile aerial robot, demonstrating a new domain of possible deployment of LLMs-based reasoning.

3 Approach

3.1 Approach Overview

The objective of our work is to design a decision-making and adaptation mechanism that uses LLMs to enable successful and resilient mission execution in autonomous systems despite the presence of uncertainties and potentially unplanned/unexpected failures that may happen across different levels of the autonomy stack. The considered autonomous system is a multirotor, whose objective consists in reaching and hovering at a desired position. During the mission, the robot is subject to uncertainties, such as model errors or wind, that may cause a critical mission failure. The robot needs to understand how to mitigate the effect of those uncertainties and autonomously decide whether to abort the mission if the effects of those uncertainties cannot be corrected, based on a natural-language specified risk tolerance. Our approach, summarized in Fig. 1, leverages an LLM to trigger adaptive/resilient behaviors in the mission planning and control stack, taking as input available signals, pre-defined error codes, and natural-language based logs and error messages. In the following sections, we define in detail the interface between an existing autonomy stack and the LLM.

3.2 Autonomy Stack

3.2.1 Controller

We consider a multirotor controlled by a cascaded position and attitude controller. The employed position controlled is based on a Linear-Quadratic Regulator (LQR) that uses a hover-linearized model (derived from [44]) of an attitude-controlled robot of the form $\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t$, where $\mathbf{x} \in \mathbb{R}^8$ is the state and $\mathbf{u} \in \mathbb{R}^3$ is the control input. The state is $\mathbf{x} = [{}_W\mathbf{p}^\top, {}_W\mathbf{v}^\top, {}_I\phi, {}_I\theta]^\top$, where ${}_W\mathbf{p}^\top \in \mathbb{R}^3$ and ${}_W\mathbf{v}^\top \in \mathbb{R}^3$ represent, respectively, the position and velocity expressed in a world frame W. The quantities ${}_I\phi$ and ${}_I\theta$ denote the attitude of the robot, expressed as roll and pitch Euler angles in a gravity-aligned, yaw-fixed frame I, whose x-axis is aligned with the world reference frame W.

We define the control input $\mathbf{u}$ to be: $\mathbf{u} = [{}_I\phi_{\text{cmd}}, {}_I\theta_{\text{cmd}}, \delta f_{\text{cmd}}]^\top$, where δf_{cmd} denotes the linearized commanded thrust, and ${}_I\phi_{\text{cmd}}$ and ${}_I\theta_{\text{cmd}}$ are the commanded roll and pitch. These commands are executed by a cascaded attitude controller.

The control input is computed via:

$$\mathbf{u}_t = \bar{\mathbf{u}}_t + \mathbf{K}(\mathbf{x}_t - \mathbf{x}_t^{\text{ref}}) + \delta\mathbf{u}_t, \tag{1}$$

where $\bar{\mathbf{u}}_t$ represents the nominal command at hover and $\mathbf{x}_t^{\text{ref}}$ a desired reference trajectory computed by the mission-level planner. $\mathbf{K}$ is a linear gain matrix, obtained by solving the Discrete Algebraic Riccati Equation (DARE) using the linearized model $\mathbf{A}$, $\mathbf{B}$ and given positive-definite tuning matrices $\mathbf{R}$ and $\mathbf{Q}$. Key to this work, the additive control input $\delta\mathbf{u}_t$ represents an adaptive term that will be controlled by the LLM based on descriptions of the state of the system (error codes, logs), enabling adaptation at low-level control.

3.2.2 Mission Planner

The mission is managed by a finite state machine (FSM) that contains desired initial/terminal position setpoints, and timed transitions in between the desired states. Once a desired position is selected, the FSM generates reference trajectories (position, velocity) that are tracked via the position controller Eq. (1). Every state inside the FSM is connected to an `emergency_landing` action that leads to a safe state (on the ground below the robot), which can be triggered by the LLM upon seeing what it determines is sufficient cause to terminate the mission.

3.3 Prompt Design And Interface with the Autonomy Stack

In this section, we present the strategy to interface the control and mission/trajectory planning stack with the LLM. We use an approach inspired by [23], i.e., we leverage Python-based syntax to define the possible failure modes in the autonomy stack, as well as the description of a set of function callbacks (API) in our control framework available to execute corrective actions. However, our approach differs from [23], as we provide additional natural language instructions to express mission-level goals and trade-offs, i.e., the willingness to risk to continue the mission when complications arise, versus aborting the mission. Additionally, we limit the potentially dangerous execution of automatically generated Python code by providing the LLM with the instruction to call a set of pre-defined Python APIs. Last, in our experiments, the LLM is connected in a closed feedback loop with the rest of the autonomy stack, without human intervention beyond the initial prompt design.

Code Color Convention: Note that throughout this work we use the following convention: green denotes the initial prompt to the LLM; this prompt is hand-crafted by a human and is loaded at the start of the mission; grey denotes the query automatically generated by the autonomous system, and blue denotes the reply from the LLM, closing the feedback loop.

Our prompt begins with the following sentence:

Initial Prompt (Part 1)

```
# Inside the codebase of my multirotor I found the following python code:
```

This sentence introduces the LLM to the Python-based syntax that will be used next to list possible mission failures/issues, requirements, and actions available, and additionally introduces the LLM to the type of platform it needs to control. Next, we introduce a list of possible, easy-to-monitor, state-based errors and failures:

Initial Prompt (Part 2)

```
# list of possible issues/failures in mission planner/controller:
NO_ISSUE = 0
FLYING_TOO_HIGH = 3
FLYING_TOO_LOW= 4
FLYING_TOO_LARGE_POSITIVE_POSITION_ERROR_X = 7
FLYING_TOO_LARGE_NEGATIVE_POSITION_ERROR_X = 8
FLYING_TOO_LARGE_POSITIVE_POSITION_ERROR_Y = 5
FLYING_TOO_LARGE_NEGATIVE_POSITION_ERROR_Y = 6
```

These failures can be easily detected, and their corresponding number is fed as input to the LLM. Additionally, we found that the LLM is more easily able to interpret failures expressed in natural language than failures expressed in numerical signals (i.e., current trajectory tracking errors). The corresponding error codes are generated by comparing the current trajectory tracking error $\mathbf{p}_t - \mathbf{p}_t^{\text{des}}$, and by triggering an issue on the corresponding axis if the error exceeds a predefined threshold.

Next, we define a new fictitious Python variable and function call that computes the possible failures:

Initial Prompt (Part 3)

```
# check current failure using check_failure. outputs a list of possible failures, for example [2, 3],
# and a string with additional information. The string may be empty.
# Example current_failure: ([2, 3], 'position error = [0.1, -0.1, 1.5]')
current_failures = check_failures()
```

As in [23], we make use of Python comments to provide contextual information on the output of the function call and describe additional inputs that we will be feeding into the LLM, using the second term in tuple of `current_failure`. This extra input can be used to provide descriptive error messages or other information that is not known/does not need to be specified a priori, providing additional flexibility in the type of information that we can feed to the LLM.

Next, we provide the LLM with information about the system-level actions (APIs) that the LLM can select:

```
Initial Prompt (Part 4)
# possible failure mitigation strategies
from controller import (
  # modify control input
  increase_thrust, decrease_thrust, accel_positive_x, accel_negative_x, accel_positive_y, accel_negative_y,
  # Mission-level decisions
  emergency_landing, do_nothing,
  # Controller tuning - we use a LQR
  tune_controller_by_decreasing_the_cost_of_actuation_usage,
  tune_controller_by_increasing_the_cost_of_actuation_usage,
  tune_controller_by_increasing_penalty_on_position_errors,
  tune_controller_by_decreasing_penalty_on_position_errors,
)
```

These actions correspond to changes in the control inputs or to events in the mission planner. More specifically, `increase_thrust` and `decrease_thrust` increase/decrease an adaptive term in the control input, while `accel_positive_...` and `accel_negative_...` produce accelerations along an axis by increasing/decreasing the extra roll/pitch setpoints by a pre-specified amount. Additionally, `tune_controller_...` updates the corresponding part of the weight matrices $\mathbf{R}$ and $\mathbf{Q}$ of the position controller Eq. (1); the corresponding DARE is solved onboard and the resulting gain matrix updates $\mathbf{K}$.

Towards the end of the prompt, we switch back to natural language to provide mission specifications:

```
Initial Prompt (Part 5)
From now on, I provide you with the value of the variable "current_failure",
and your output needs to be your best guess of the function names in the python list
"list_of_function_names_to_be_executed_right_now".
For instance, your output: ["emergency_landing"],"low_battery_voltage",
"because the drone can hardly move it is safer to land"
Try to think like a drone control engineer.
```

This prompt specifies the output that we expect from the LLM (a list of names of functions the controller can execute). It additionally includes two elements that can help the LLM reason about its choice of actions, and a brief and long explanation of the issue. Following best prompting practices, we also encourage the LLM to role-play, i.e., thinking like a "drone control engineer".

In addition, we discourage the LLM from outputting planned future actions and encourage brevity in its explanations, by adding the following lines in the initial prompt:

```
Initial Prompt (Part 6)
DO NOT output function names to be called in the future, but account for past problems to come up
with your guess of the functions in "list_of_function_names_to_be_executed_right_now".
```

Last, we further make the LLM aware of the possibility of taking emergency landings:

```
Initial Prompt (Part 7)
If problems persist, do not hesitate to emergency land.
if your actions do not take the desired effect, you must perform an emergency landing.
```

We note that omitting these sentences was making the LLM less prone to trigger an emergency landing, while exaggerating the need to emergency land (e.g., using "MUST" instead of "must") made the LLM more prone to immediately trigger an emergency landing, potentially providing a natural-language avenue to specify willingness to take risks in an autonomous system.

4 Evaluation

4.1 Implementation Details

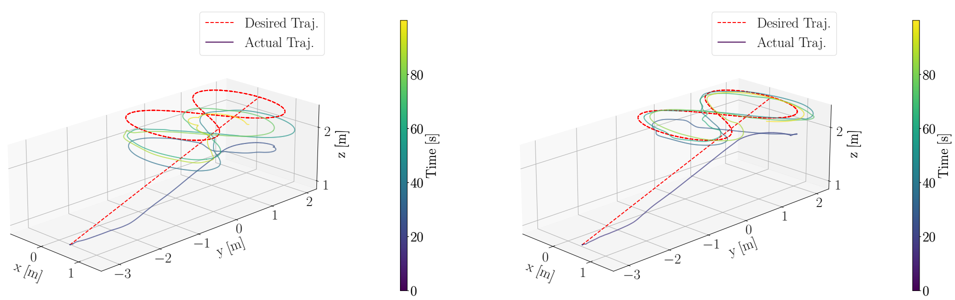
We perform real-world experiments by deploying REAL on a multirotor. The multirotor is equipped with an Intel® NUC™ 10. Our system operates in real-time on the NUC™ 10, driven by the Intel® Core™ i7-10710U Processor. All planning, control, and state estimation, which merges IMU data with a motion capture system, are executed onboard, except the LLM, which is queried from onboard, receiving replies generated via the `OpenAI GPT-4 API`. The UAV connects to the internet over Wi-Fi and queries the LLM at as high of a rate as possible. This results in the LLM running at from 0.1 to 1.0 Hz, depending on network latency and API usage.

4.2 Low-Level Adaptation and Controller Auto-Tuning

In this experiment, we evaluate the ability of the LLM to perform low-level adaptive control and decision-making, by adjusting the control input of our UAV based on the issues reported to the LLM. To introduce tracking error, we purposefully use the wrong value for the mass parameter while synthesizing the controller; the value of the mass parameter used is about 15% of the robot's true mass, resulting in a large altitude error. We then deploy the UAV in a mission that consists of taking off, following a figure-eight trajectory, and then landing. We repeat the experiment two times, with the difference that in the first experiment, to study the choice that the LLM would make absent this parameter, we removed the `tune_controller_by_..` in `Initial Prompt (Part 4)` that API call.

Fig. 2a shows the result without tuning the API call `tune_controller_by_..` in `Initial Prompt (Part 4)`, highlighting that throughout the mission, REAL succeeds at improving the altitude improve the trajectory tracking through the duration of the experiment (about $100\,\mathrm{s}$) by repeatedly calling the `increase_thrust` in `Initial Prompt (Part 4)`, obtaining an altitude tracking error within $30\,\mathrm{cm}$. Fig. 2b shows the results on the same trajectory, but with the `tune_controller_by_..` in `Initial Prompt (Part 4)` added back to the prompt. In this experiment, the LLM calls both commands, as shown by the conversation between the LLM and the robot during the experiment (please see Conversation 1 in the Appendix).

The altitude tracking error resulting from this experiment is within only $10\,\mathrm{cm}$. This shows that the LLM is capable of selecting multiple mission-relevant low-level control APIs, without providing detailed prior instructions on its choice, and whose usage is beneficial at improving the outcome of the mission.



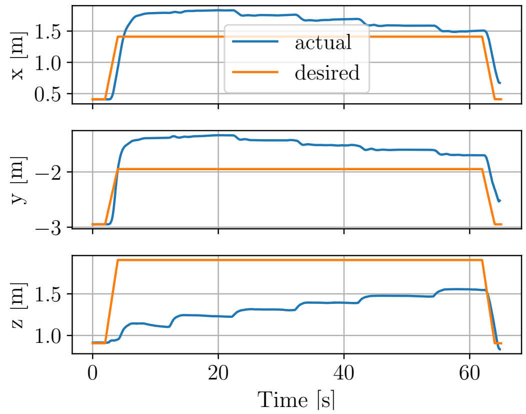
(a) Trajectory Tracking **without** Control Gain $\mathbf{K}$ Tuning

(b) Trajectory Tracking **with** Control Gain $\mathbf{K}$ Tuning.

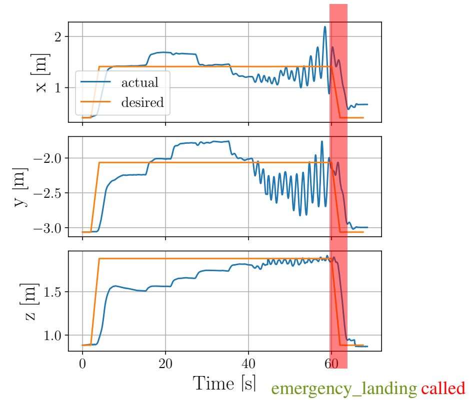
Figure 2: REAL's successful altitude (z-axis) adaptation during a figure eight trajectory ($0.25\,\mathrm{m/s}$) with an incorrect mass parameter used in the controller ($15\,\%$ error from nominal value). The top figure shows the experiment performed when the API calls `tune_controller_by_...` removed from the initial prompt, and the bottom figure shows the same experiment with the `tune_controller_by_...` APIs added back. These results illustrate that REAL can successfully interpret the system-provided error messages, and call useful APIs. Specifically, REAL achieves adaptation in the top plot by calling the `increase_thrust` API, successfully converging to the desired altitude. In the secondary scenario (bottom figure), REAL calls a combination of `tune_controller_by_...` and `increase_thrust` APIs, triggering changes in the controller matrix $\mathbf{K}$. This results in faster/better convergence to the desired trajectory.

4.3 Low-Level Adaptation to Unmodeled Dynamics

This experiment presents a more challenging scenario that highlights REAL's ability to correct for unmodeled dynamics. To test this, we place a large extra mass on the end of one of the multirotor's arms, creating an unmodeled torque disturbance. For brevity, the prompt is not shown, but adaptation is triggered by repeatedly selecting the expected API calls: `increase_thrust`, `accel_negative_y` and `accel_negative_x`. Fig. 3a shows the hardware experiment results of the LLM successfully reasoning how to eliminate error along each of its axes. Note that in this earlier experiment, roll and pitch torque function names were used in the prompt to control acceleration along the y and x axes respectively. We later found that the LLM was more consistent when using the commands for requesting acceleration in x and y directions directly.



(a) REAL reducing position errors across the x, y, and z-axes under unmodeled dynamics: In this experiment an additional weight of 210 g was added to one of the UAV's arms, creating a large unmodeled external torque that affects the attitude and position control of the UAV. REAL identifies the correct APIs to reduce this error, though convergence is slowed down due to the limited rate at which the API can be called.



(b) REAL's position adaptation combined with its ability to trigger emergency landing automatically. The UAV was configured with a $15\,\%$ mass-mismatch, that is successfully compensated by REAL in the initial phase of the experiment. Subsequently, we artificially introduced oscillations by pulling a cable attached to the UAV. Upon receiving a natural-language log that informs REAL of these oscillations, the LLM invokes the `emergency_landing` API. This experiment highlights REAL's ability to handle both lower-level control adaptations and higher-level mission-critical decisions such as aborting the mission for safety reasons.

Figure 3: Real's ability to perform low-level adaptation and trigger mission-relevant decision.

4.4 Mission-scale Decision Making via Unsafe State Detection and Automatic Mission Abortion

The purpose of this experiment is two-fold: (1) test the LLM's ability to make critical mission-level decisions in the event that unforeseen circumstances cause the UAV to lose control and (2) test the LLM's ability to process additional information that was not in the original prompt.

Since the LLM is only able to make adaptive corrections at a slow rate, we would expect it to call for an emergency landing in the event of loss-of-control. To simulate a loss-of-control event, we apply large external disturbances to the robot by pulling it via a rope, causing large oscillations along the y axis, as seen in Fig. 3b. Onboard the drone, we implement a module that stores a buffer of positions and uses a Fast Fourier Transform (FFT) to evaluate the maximum amplitude of any frequency content. If the maximum frequency content is above a threshold, this information is outputted to the log received by the LLM, as shown in the conversation history in Conversation 2 (please see Conversation 2 in Appendix). Although the LLM was not told it would receive any information about oscillations nor what to do if it did receive this information, it was able to make the critical decision to select the `emergency_land` API when it recognized that the UAV was in a dangerous condition via the provided log message `"DANGEROUS oscillations..."`. Fig. 3b additionally highlights the LLM's ability to correct the altitude errors of the UAV, caused by an additional 15% mass mismatch, as visible before the interaction begins.

5 Conclusions

We have presented REAL, a method to utilize LLMs for zeros-shot cross-stack adaptation and autonomy on mobile robots. The method works by leveraging LLMs' natural language understanding, its ability to reason over long sequences, and its embedded prior knowledge of the robot's model/dynamics. Our evaluation in hardware experiments has demonstrated promising first-steps towards low-level adaptation and increased mission-level resilience. In the future we would like to further extend this framework to more complex, multi-robot autonomous systems, where more complex reasoning is required to diagnose and recover from failures.

Acknowledgments

Work funded by the Air Force Office of Scientific Research MURI FA9550-19-1-0386.

References

[1] G. Joshi and G. Chowdhary. Deep model reference adaptive control. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4601–4608. IEEE, 2019.

[2] S. M. Richards, N. Azizan, J.-J. Slotine, and M. Pavone. Adaptive-control-oriented meta-learning for nonlinear systems. *Robotics: Science and Systems (RSS)*, 2021.

[3] N. Hovakimyan, C. Cao, E. Kharisov, E. Xargay, and I. M. Gregory. $\mathcal{L}_1$ adaptive control for safety-critical systems. *IEEE Control Systems Magazine*, 31(5):54–104, 2011.

[4] L. Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

[5] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *Robotics: Science and Systems (RSS)*, 2021.

[6] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Sparse identification of nonlinear dynamics with control (sindyc). *IFAC-PapersOnLine*, 49(18):710–715, 2016.

[7] A. Loquercio, A. Saviolo, and D. Scaramuzza. Autotune: Controller tuning for high-speed flight. *IEEE Robotics and Automation Letters*, 7(2):4432–4439, 2022.

[8] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe. Automatic lqr tuning based on gaussian process global optimization. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 270–277. IEEE, 2016.

[9] F. Berkenkamp, A. P. Schoellig, and A. Krause. Safe controller optimization for quadrotors with gaussian processes. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 491–496. IEEE, 2016.

[10] A. Furda and L. Vlacic. Towards increased road safety: Real-time decision making for driverless city vehicles. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 2421–2426, 2009. doi:10.1109/ICSMC.2009.5346395.

[11] M. Tranzatto, M. Dharmadhikari, L. Bernreiter, M. Camurri, S. Khattak, F. Mascarich, P. Pfreundschuh, D. Wisth, S. Zimmermann, M. Kulkarni, V. Reijgwart, B. Casseau, T. Homberger, P. D. Petris, L. Ott, W. Tubby, G. Waibel, H. Nguyen, C. Cadena, R. Buchanan, L. Wellhausen, N. Khedekar, O. Andersson, L. Zhang, T. Miki, T. Dang, M. Mattamala, M. Montenegro, K. Meyer, X. Wu, A. Briod, M. Mueller, M. Fallon, R. Siegwart, M. Hutter, and K. Alexis. Team cerberus wins the darpa subterranean challenge: Technical overview and lessons learned, 2022.

[12] A. Agha, K. Otsu, B. Morrell, D. D. Fan, R. Thakker, A. Santamaria-Navarro, S.-K. Kim, A. Bouman, X. Lei, J. Eadlund, M. F. Ginting, K. Ebadi, M. Anderson, T. Pailevanian, E. Terry, M. Wolf, A. Tagliabue, T. S. Vaquero, M. Palieri, S. Tepsuporn, Y. Chang, A. Kalantari, F. Chavez, B. Lopez, N. Funabiki, G. Miles, T. Touma, A. Buscicchio, J. Tordesillas, N. Alatur, J. Nash, W. Walsh, S. Jung, H. Lee, C. Kanellakis, J. Mayo, S. Harper, M. Kaufmann, A. Dixit, G. Correa, C. Lee, J. Gao, G. Merewether, J. Maldonado-Contreras, G. Salhotra, M. S. D. Silva, B. Ramtoula, Y. Kubo, S. Fakoorian, A. Hatteland, T. Kim, T. Bartlett, A. Stephens, L. Kim, C. Bergh, E. Heiden, T. Lew, A. Cauligi, T. Heywood, A. Kramer, H. A. Leopold, C. Choi, S. Daftry, O. Toupet, I. Wee, A. Thakur, M. Feras, G. Beltrame, G. Nikolakopoulos, D. Shim, L. Carlone, and J. Burdick. Nebula: Quest for robotic autonomy in challenging environments; team costar at the darpa subterranean challenge, 2021.

[13] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato. Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

[14] S. Omidshafiei, A.-A. Agha-Mohammadi, C. Amato, and J. P. How. Decentralized control of partially observable markov decision processes using belief space macro-actions. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 5962–5969. IEEE, 2015.

[15] A. Bouman, M. F. Ginting, N. Alatur, M. Palieri, D. D. Fan, T. Touma, T. Pailevanian, S.-K. Kim, K. Otsu, J. Burdick, et al. Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2518–2525. IEEE, 2020.

[16] OpenAI. Gpt-4 technical report, 2023.

[17] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[18] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[19] I. Drori, S. Zhang, R. Shuttleworth, L. Tang, A. Lu, E. Ke, K. Liu, L. Chen, S. Tran, N. Cheng, R. Wang, N. Singh, T. L. Patti, J. Lynch, A. Shporer, N. Verma, E. Wu, and G. Strang. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119, 2022. doi:10.1073/pnas.2123433119. URL https://www.pnas.org/doi/abs/10.1073/pnas.2123433119.

[20] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur, G. Gur-Ari, and V. Misra. Solving quantitative reasoning problems with language models, 2022.

[21] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021.

[22] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

[23] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[24] S. Huang, Z. Jiang, H. Dong, Y. Qiao, P. Gao, and H. Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model, 2023.

[25] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser. Tidybot: Personalized robot assistance with large language models, 2023.

[26] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023.

[27] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022.

[28] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[29] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[30] J. Svacha, J. Paulos, G. Loianno, and V. Kumar. Imu-based inertia estimation for a quadrotor using newton-euler dynamics. *IEEE Robotics and Automation Letters*, 5(3):3861–3867, 2020.

[31] V. Wüest, V. Kumar, and G. Loianno. Online estimation of geometric and inertia parameters for multirotor aerial vehicles. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1884–1890. IEEE, 2019.

[32] A. Tagliabue, A. Paris, S. Kim, R. Kubicek, S. Bergbreiter, and J. P. How. Touch the wind: Simultaneous airflow, drag and interaction sensing on a multirotor. In *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1645–1652.

[33] A. Tagliabue, M. Kamel, R. Siegwart, and J. Nieto. Robust collaborative object transportation using multiple MAVs. *The International Journal of Robotics Research*, 38(9):1020–1044, 2019.

[34] C. D. McKinnon and A. P. Schoellig. Unscented external force and torque estimation for quadrotors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5651–5657. IEEE, 2016.

[35] B. T. Lopez. *Adaptive robust model predictive control for nonlinear systems*. PhD thesis, Massachusetts Institute of Technology, 2019.

[36] J. P. How, B. Lopez, P. Lusk, and S. Morozov. Performance analysis of adaptive dynamic tube MPC. page 0785, 2021.

[37] A. Saviolo, J. Frey, A. Rathod, M. Diehl, and G. Loianno. Active learning of discrete-time dynamics for uncertainty-aware model predictive control. *arXiv preprint arXiv:2210.12583*, 2022.

[38] G. Joshi, J. Virdi, and G. Chowdhary. Design and flight evaluation of deep model reference adaptive controller. In *AIAA Scitech 2020 Forum*, page 1336, 2020.

[39] S. Zhou, K. Pereida, W. Zhao, and A. P. Schoellig. Bridging the model-reality gap with lipschitz network adaptation. *IEEE Robotics and Automation Letters*, 7(1):642–649, 2021.

[40] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural-fly enables rapid learning for agile flight in strong winds. *Science Robotics*, 7(66): eabm6597, 2022. doi:10.1126/scirobotics.abm6597.

[41] S. Omidshafiei, A.-A. Agha-Mohammadi, C. Amato, S.-Y. Liu, J. P. How, and J. Vian. Decentralized control of multi-robot partially observable markov decision processes using belief space macro-actions. *The International Journal of Robotics Research*, 36(2):231–258, 2017.

[42] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, P. Florence, C. Fu, M. G. Arenas, K. Gopalakrishnan, K. Han, K. Hausman, A. Herzog, J. Hsu, B. Ichter, A. Irpan, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, L. Lee, T.-W. E. Lee, S. Levine, Y. Lu, H. Michalewski, I. Mordatch, K. Pertsch, K. Rao, K. Reymann, M. Ryoo, G. Salazar, P. Sanketi, P. Sermanet, J. Singh, A. Singh, R. Soricut, H. Tran, V. Vanhoucke, Q. Vuong, A. Wahid, S. Welker, P. Wohlhart, J. Wu, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*, 2023.

[43] A. Zeng. *Learning Visual Affordances for Robotic Manipulation*. PhD thesis, Princeton University, 2019.

[44] M. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017. ISSN 2405-8963. doi:https://doi.org/10.1016/j.ifacol.2017.08.849. URL https://www.sciencedirect.com/science/article/pii/S2405896317313083. 20th IFAC World Congress.

6 Appendix

6.1 Conversation 1

```
Conversation 1
t = 3.92s: Prompt ([0], ")
t = 3.92s: Response list_of_function_names_to_be_executed_right_now: ["do_nothing"], "no_issue", "Since there
           are currently no issues detected with the drone, no corrective actions are needed at this time."
t = 8.47s: Prompt ([4, 6], 'z error is -0.14, y error is -0.44, ')
t = 8.47s: Response list_of_function_names_to_be_executed_right_now: ["increase_thrust", "accel_positive_y"]
           "flying_too_low_and_negative_y_position", "The drone is currently flying too low and also has a
           negative error in the Y-axis. Increasing thrust will correct the altitude issue, and accelerating in
           the positive Y direction will correct the positional error."
t = 14.03s: Prompt ([4, 6], 'z error is -0.63, y error is -1.18, ')
t = 14.03s: Response list_of_function_names_to_be_executed_right_now: ["increase_thrust", "accel_positive_y",
            "tune_controller_by_increasing_penalty_on_position_errors"], "flying_too_low_and_large_negative
            _y_position", "The drone is flying significantly too low and has a large negative error on the Y-axis.
            It's necessary to increase thrust and accelerate in the positive Y direction to correct these issues.
            Additionally, the large errors suggest it may be beneficial to tune the controller to penalize
            position errors more severely."
t = 19.0s: Prompt ([4, 5, 7], 'z error is -0.30, y error is 0.15, x
           error is 0.12, ')
t = 19.0s: Response list_of_function_names_to_be_executed_right_now: ["increase_thrust", "accel_negative_y",
           "accel_negative_x"], "flying_too_low_and_positive_position_errors", "The drone is below the desired
           altitude and has minor positive errors in both X and Y axis. To correct these, it should increase
           thrust to gain altitude, and decelerate in Y and X directions."
```

6.2 Conversation 2

```
Conversation 2
t = 6.07s: Prompt ([0], ")
t = 6.07s: Response list_of_function_names_to_be_executed_right_now: ["do_nothing"] reason: The reported
           information indicates that there are currently no discernible issues with the drone. Therefore, no
           actions are necessary at this time.
t = 13.95s: Prompt ([4, 6, 8], 'z error is -0.69, y error is -0.62,
            x error is -0.40, ')
t = 13.95s: Response list_of_function_names_to_be_executed_right_now: ["increase_thrust", "tune_controller_by
            _decreasing_the_cost_of_actuation_usage", "accel_positive_y", "accel_positive_x"] reason: The drone
            is flying too low and has negative position errors in both X and Y directions, hence, increasing
            thrust is the first step, along with acceleration in positive X and Y directions to correct the
            position errors. Besides, the cost of actuation usage should also be decreased to ensure a higher
            rate of control input changes for faster response. It's preferable to decrease the cost of
            actuation usage to make the drone respond more swiftly to the control inputs, as there are multiple
            issues happening simultaneously.
t = 57.48s: Prompt ([6, 8], 'y error is -0.65, x error is -0.28, VERY
            DANGEROUS oscillations on y-axis. Frequency is 0.67 [Hz],
            amplitude is 0.19 [m].')
t = 57.48s: Response list_of_function_names_to_be_executed_right_now: ["emergency_landing"] reason: The drone
            has large errors in both X and Y directions, and additionally, is exhibiting dangerous oscillations
            on the Y-axis. This indicates unstable flight dynamics which could be detrimental to the safety of the
            operation. An immediate emergency landing should be executed to prevent potential damage or hazards.
```