

# Automated UAV Controller Synthesis via LLM-Generated Control Logic and Particle Swarm Optimisation

**Abstract**—LLM-guided evolutionary frameworks for program synthesis have demonstrated strong results across combinatorial benchmarks; however, such frameworks have not been widely explored for robotic control systems. This paper presents an automated framework for synthesising control logic and optimising numerical coefficients, enabling the deployment of generated controllers onto robotic platforms. To produce deployable controllers, the framework separates control logic synthesis from numerical parameter optimisation, with controller coefficients optimised via Particle Swarm Optimisation using a Markov Decision Process reward signal. The resulting controllers are evaluated in a custom UAV simulation environment, validated using PX4 Software-In-The-Loop, and subsequently deployed on a physical UAV. Controller performance is evaluated on Lemniscate and Lissajous trajectory-tracking tasks and compared against PID with disturbance observer and Linear Quadratic Regulator baselines. Across both trajectories, the framework-generated control law achieves improved tracking accuracy relative to the baseline controllers, with a minimum reduction in mean squared error of approximately 38% in real-world experiments. These results demonstrate the feasibility of deploying automatically synthesised control logic on a physical UAV, bridging automated program synthesis and real-world control deployment.

Large Language Models, Optimisation, Control, Unmanned Aerial Vehicles

## I. INTRODUCTION

DESIGN and testing of high-performance control systems for Unmanned Aerial Vehicles (UAVs) is a difficult and time-consuming process that relies on expert knowledge. While classical control systems such as Proportional-Integral-Derivative (PID) and Linear Quadratic Regulator (LQR) remain widely used, their performance is sensitive to modelling inaccuracies, parameter selection, and operating conditions. More recently, reinforcement learning (RL) methods have shown promise in automating controller design; however, their reliance on training data and the sim-to-real gap continue to limit their widespread adoption in real-world systems.

In parallel, Large Language Models (LLMs) have emerged as powerful tools used for much more than simple question and answer chatbots [?]. LLMs are now being utilised for program synthesis and optimisation across a wide range of domains [?]. Recent work has demonstrated that LLMs can generate functional code, assist local search, and even outperform human-designed heuristics in combinatorial tasks [?]. Despite this progress, the application of LLMs to control system synthesis remains unexplored, with little evidence that LLM-generated algorithms can be deployed on physical systems. Key challenges include the appropriate selection of program parameters, evaluation of control systems under model mismatch, and the transition from simulation to real-world execution.

This work addresses these challenges by proposing an automated framework that combines LLM-based program synthesis with classical optimisation and real-world validation. Rather than relying on LLMs to optimise continuous parameters directly, we deliberately decouple controller structure from parameter tuning. In the proposed approach, LLMs are responsible for generating control logic, while real-valued coefficients are optimised using Particle Swarm Optimisation (PSO). This separation closely aligns with established practices within Symbolic Regression (SR) and evolutionary control, enabling each component to operate within its own strengths.

Controller designs are refined through a multi-shot evolutionary querying strategy. Previous controller programs and their performance metrics are incorporated into subsequent LLM queries, enabling a guided optimisation procedure with quantitative feedback. An actor-critic LLM loop is employed, whereby one model proposes control logic and another provides recommendations based on results.

The proposed framework is evaluated on a thrust-and-torque quadcopter control problem. Controllers are trained in a custom UAV environment, validated in PX4 SITL, and finally deployed onto a real UAV. Performance is assessed on two challenging trajectory-tracking tasks and compared against a PID+DOB and LQR controller. The results demonstrate that the generated control system achieves superior tracking and robustness across real-world and simulated experiments.

The primary contribution of our paper is the deployment of a generated control system to a UAV. Our results position LLM-assisted controller design as a viable and scalable tool for engineering applications, bridging the gap between automated controller design and real-world deployment.

## II. RELATED PAPERS

The problem this work focuses on solving is the deployment of generated algorithms in the real world. The current technical issues faced when trying to implement such a solution are related to the choice of the parameters for a given computer program, and scoring the quality of the produced computer programs within the simulation due to model mismatch. Our specific contributions, to that end, are: multi-shot local policy search, training and test environments, and a method that utilises a UAV platform to test generated control algorithms.

In the literature, the most related paper to our work appears to be ‘LLM-Assisted Local Search for Policy Synthesis’ [?]. With Sadamine, Baier, and Lelis’s work, they demonstrate how LLMs can be used to create programmatic policies for various tasks in a single shot. Notably, our work differs from theirs in two separate instances: our policies are tested in the real world, and we focus on multiple queries to the LLM for program

synthesis. Based on our previous work, it appears that a wide approach to program synthesis has provided better results than a small initial population for performing the local search.

In terms of evolutionary papers related to ours, the basis for this paper is FunSearch [?]. Within FunSearch, the authors demonstrated how evolutionary queries can be utilised to surpass human performance on combinatorial problems. Compared to our work, which focuses on both evolutionary and local parameterisation, the FunSearch method is costly and yields significantly worse results. The primary reasons behind the increased cost of FunSearch are related to local parameterisation. It is unlikely that the FunSearch technique, on its own, can be used to create a high-quality control system within a reasonable time or without incurring significant excess computational cost. The inefficiencies of FunSearch are due to the choice of coefficient values within program synthesis. Opposed to simply altering a variable value, the FunSearch technique typically opts for a more complex solution to a problem.

Li et al. [?] focused on an enumerative search technique for program synthesis. Li et al.’s work focuses primarily on the synthesis of programs for solving a complex dataset (SyGuS), with a continuous bi-directional communication feedback loop for failed programs. The primary contribution of their research is the creation of a ‘probabilistic Context-Free Grammar (pCFG)’, a list of failed programs, to help the LLM achieve a correct solution. A similar technique is used in our methodology. Opposed to giving a specific name to this technique, we define how and why we implemented it. For our work, we instead focus on the implicit information provided to the LLM through evolutionary queries and epigenetic program selection. Our evolution queries include previous examples of programs, along with a meta-heuristic, the ‘score’ of a program. Theoretically, providing the programs in the query as our approach, as opposed to theirs, provides increased context for the model to make intelligent decisions. Our work, rather than focusing on achieving a broad application of the technique, aims to apply the optimisation to a single use case. This allows us to incorporate information easily and include meta-heuristic information into our queries—a difficult task to achieve in the broad sense.

A related work that might help support the theoretical results of this approach, which has not been implemented, is the usage of multimodality within query generation before program synthesis. For example, Huang et al. [?] performed work utilising GPT-4’s multimodal features, demonstrating how multimodality can enhance optimisation performance. In the context of our work, implementing this makes sense simply due to the ease and clarity with which information about a trajectory can be displayed in a visual format. However, a decision was made not to incorporate multimodal queries within our results, primarily due to our incentive to keep the monetary cost of generation at zero. We believe that ensuring the optimisation process is free increases much-appreciated scrutiny and facilitates widespread adoption.

For a recent review and survey of papers covering the relevant topic area, the authors recommend either Wu et al. [?] or Huang et al. [?]. Both papers provide a comprehensive

overview of the current progress in Large Language Models and Optimisation.

The reasoning behind this decision is that PSO has proven effective on a range of control problems. For example, researchers have applied PSO to automatically tune PID gains for UAV attitude and position control, achieving improved trajectory tracking performance without manual tuning [?].

To our knowledge, no one has yet deployed a generated algorithm in the real world.

### III. METHODOLOGY

Our approach employs a dual-query Large Language Model (LLM) framework to automatically synthesise a thrust/torque UAV controller. A primary critic LLM first generates a high-level specification for a controller, and a secondary critic LLM analyses simulation responses and suggests feedback. Collaboration between the actor and critic forms an actor-critic LLM loop, motivated by successes in LLM-assisted program synthesis [?]. LLMs can produce functional code for most tasks, but pairing with a critic or verification step improves reliability [?]. When implemented in this manner, the LLM is proposed to refine its output based on performance, which is synonymous with self-reflection and In-Context-Learning (ICL) [?]. By incorporating a critic model, we improve performance, similar to recent LLM approaches that use an agent to verify actions [?]. The two LLMs operate together: the primary LLM proposes a control logic structure, and the secondary LLM implements the logic and provides quantitative feedback.

A key and deliberate feature of our method is the abstraction of code logic and the optimisation of numerical coefficients. The LLM pair performs high-level decision making and program synthesis (e.g., controller design, conditional logic), with the real-valued coefficients (e.g., vehicle mass and controller gains) of the program optimised via classical methods. The reasoning behind this deliberate abstraction is simple: LLMs excel at writing structured code from descriptions and making high-level decisions, yet lack direct real-value optimisation capability. Our approach mirrors that of Symbolic Regression, whereby a genetic programming technique evolves a high-level topology, with classical optimisers tuning constants [?]. Similarly, our approach produces a human-readable control policy, with placeholder values treated as tunable parameters. This separation of responsibilities is advantageous as it combines expert knowledge ingested by LLMs during training with the search capabilities of optimisation for real-valued parameters. We refer to this separation of duties as program specification (controller design) and program parameterisation (optimisation of parameters).

To implement the program parameterisation stage, we use Particle Swarm Optimisation (PSO). In our specific case, each particle represents a set of program coefficients (e.g., PID gains). The PSO algorithm optimises the coefficients to minimise the trajectory-tracking error for a given program. The simulated fitness score enables the PSO algorithm to converge towards a suitable parameter set for each controller. We modified the standard PSO algorithm to enhance the global stability criteria for our problem.

Firstly, the inertia decays over time, resulting in exploitation of the search space nearer the end of the search [?]:

$$\omega(t) = \omega_{\max} - \left( \frac{\omega_{\max} - \omega_{\min}}{T} \right) t \quad (1)$$

with  $\omega_{\max}$  and  $\omega_{\min}$  representing the maximum and minimum inertia, and  $T$  representing the total number of iterations.

Time-varying acceleration coefficients for social and personal coefficients are also applied to the algorithm [?]:

$$c_1(t) = c_{1,\min} + (c_{1,\max} - c_{1,\min}) \frac{t}{T} \quad (2)$$

$$c_2(t) = c_{2,\max} - (c_{2,\max} - c_{2,\min}) \frac{t}{T} \quad (3)$$

And finally, the velocity is clamped such that particles cannot move too fast across the search space [?]:

$$PSv_{\max}(t) = v_{\max, \text{initial}} - \left( \frac{v_{\max, \text{initial}} - v_{\max, \text{final}}}{T} \right) t. \quad (4)$$

To further refine the generated solution, another local search step is performed upon the optimal program before deployment and testing. We take the best solution from PSO and perform a larger optimisation step to ensure that a better set of coefficients has not been missed. That is, a coarse initial optimisation is performed, followed by a finer second optimisation on the best-scoring program. The justification for this step stems from analyses of PSO, which indicate that PSO converges to local optima without guaranteeing global optimality [?]. This final search exploits the best solution and fine-tunes parameters for minor performance improvements. Essentially, PSO is used in a global sense under broad search constraints, and a subsequent search is performed to refine the result. A combination of a final, deeper refinement stage with a fast-searching stage enables the finalisation of our algorithm, which identifies a near-optimal set of coefficients for the generated controller.

#### A. Implementation pipeline

Notably, all the steps listed below are automated, and our algorithm requires only a description of the problem and the loss function to begin optimisation. See Figure ?? for a general overview of this automated process.

We begin by querying the LLM with a detailed description of the task at hand and specific requirements. For our case, this prompt also included an example base controller (PID). It is not necessary to include a base template, as discussed in [?]; however, faster convergence is achieved with a given base. The prompt also includes structural and typographic information (C++, return values, frame translations). We also ensure that the prompt consists of details on placeholder coefficient gains and how to access them. The actor LLM's response produces an initial set of viable population members. For each response we receive, we ensure that it compiles without runtime or compilation errors before assuming it is a valid response. For any set of invalid responses received, the actor LLM is queried again for a new output until a complete set of admissible programs is available for testing.

The admissible set of responses is then integrated into the simulation environment to perform optimisation. We run our RVO stage to estimate each controller's performance. The final fitness for each computer program is recorded in a Table, along with the program's identification number (ID) and score. The loss function is designed so that the minimum score in the Table above corresponds to the identification number of the optimal-performing program. Any number of elite population members can be saved for the next generation and appended to the next generation's query. Realistically and pragmatically, in our algorithm, it only makes sense to keep the most elite population member.

The environment and experiments are an ad-hoc UAV simulation based on Flightmare (details of which are listed in Section ??) [?]. The environment allows us to place the LLM's output 'in-testing', enabling automated observation of behaviours with performance metrics.

The automated metrics are provided to the critic LLM, along with the code, to integrate high-level observational critiques into the actor's control design. During this stage, multimodality could be a key feature. Providing a visual demonstration of the current optimal controller output could lead to better insights into actionable goals for the actor to implement. For example, the critic that provides information such as 'the drone overshoots the target' offers much better insight into the program's performance than descriptions of performance metrics. Multimodality has not been implemented in our algorithm due to both cost constraints and scope. The critic output and observations are appended to the query for the next generation of the population, thereby providing closed-loop feedback. This collaborative feedback loop creates an actor-critic dynamic and aligns with emergent LLM practices of using self-reflection to refine solutions.

Upon receiving the critic's response regarding program performance, we begin defining the query for the actor to produce the next generation. The new query starts again with the problem description and specific requirements, which we consider static and never change during program evolution. Exponentially sampled (based on score) examples of previous population members are included in the new query. The reasoning behind an exponential sample of the program history is based on our understanding of Epigenetic DNA, whereby previous genes are not forgotten; they are less likely to be expressed. After joining the previous examples onto our new query, we add the critic's response. This new query forms the basis for the actor's implementation of the next-generation population.

The process is repeated until either the convergence criteria are met or the maximum number of iterations is reached. In summary, the stages are: query construction, actor program synthesis, real-valued optimisation, population selection, and critiquing. A general overview of this process is provided for easy viewing in Figure ??.

#### B. Simulated training environment

The equations used to describe the UAV and its movement across time are supplied by the vast research done by [?], [?], [?], [?].

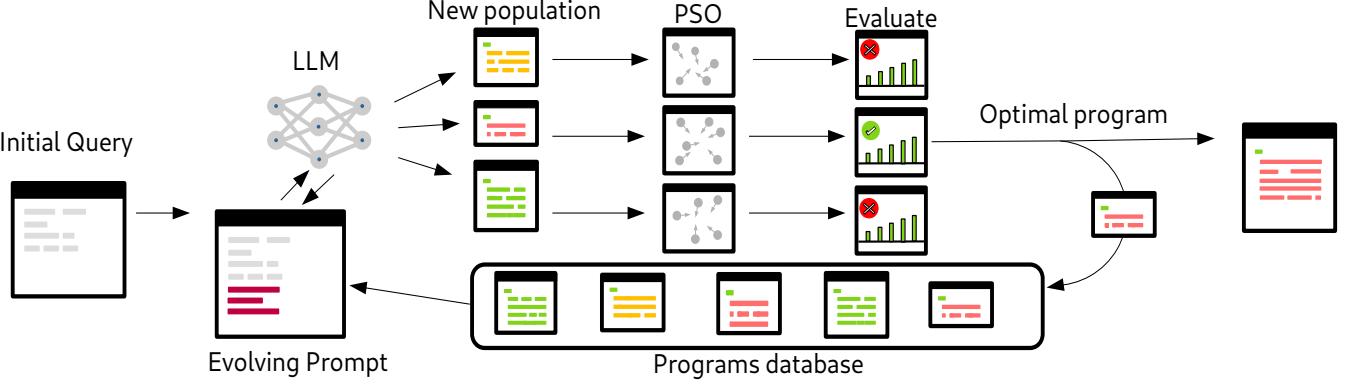


Fig. 1: **System overview:** An overview of the LLM for optimization process.

To control the UAV, we utilise mass-normalised collective thrust and body-torque ( $A \in \mathbb{R}^4$ ) measured in Newtons. The upper bound of the collective thrust is calculated as the maximum thrust produced by all four motors, as determined by measurements. The maximum body torque is the most significant differential between any two given motors.

To simulate thrust from the given outputs, the control policy output is first converted into motor angular velocity (RPM), denoted  $r_i^{des} \in \mathbb{R}^4$ , the equations for which are given by

$$r^{des} = IA\omega'_t + \omega' \times (I\omega'), \quad (5)$$

with  $A$  being the motor allocation matrix, provided as

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ l\sqrt{\frac{1}{2}} & -l\sqrt{\frac{1}{2}} & -l\sqrt{\frac{1}{2}} & l\sqrt{\frac{1}{2}} \\ -l\sqrt{\frac{1}{2}} & -l\sqrt{\frac{1}{2}} & l\sqrt{\frac{1}{2}} & l\sqrt{\frac{1}{2}} \\ k_1 & -k_2 & k_3 & -k_4 \end{bmatrix}.$$

The RPM is then passed through a first-order motor model to simulate the motor's internal dynamics,

$$\dot{r}_i = \frac{1}{\alpha_i}(r_i^{des} - r_i) \quad (7)$$

where  $\alpha_i$  is the motor ramp time for the  $i$ th rotor.

A linear model based on data from the UAV motors is used to convert the RPM of the motor into thrust:

$$f_i = r_i^2 \cdot M_c \quad (8)$$

Where  $M_c$  is the constant for the motor-propeller pair. These thrust values allow the definition of a three-axis body torque matrix  $\eta \in \mathbb{R}^3$  and collective thrust  $c$ :

$$\eta = \begin{bmatrix} \frac{l}{\sqrt{2}}(f_1 - f_2 - f_3 + f_4) \\ \frac{l}{\sqrt{2}}(-f_1 - f_2 + f_3 + f_4) \\ k_1 f_1 - k_2 f_2 + k_3 f_3 - k_4 f_4 \end{bmatrix}, \quad (9)$$

$$c = f_1 + f_2 + f_3 + f_4, \quad (10)$$

where  $l$  is the UAVs arm length,  $f_i$  is the thrust produced by the motors, and  $k_i$  is the rotor drag co-efficient.

Given the filtered body torque and collective thrust, it is possible to transition between timesteps with the following equations (note Lagrange notation for derivatives):

$$P' = V \quad (11)$$

$$V'_{wb} = Q_{wb} \odot c - G - RDR^T v \quad (12)$$

$$Q'_{wb} = \frac{1}{2}\lambda(\omega_{wb}) \cdot q_{wb} \quad (13)$$

where  $\lambda(\omega_{wb})$  is a skew-symmetric matrix of the vector  $(0, \omega^T)^T = (0, \omega_x, \omega_y, \omega_z)^T$ .

$$\omega'_{wb} = I^{-1} \cdot (\eta - \omega_{wb} \times I\omega_{wb}) \quad (14)$$

where  $G = [0, 0, -g]^T$ ,  $R \in \mathbb{R}^{3,3}$  is a rotation matrix of the form  $R = \{X_b, Y_b, Z_b\}$ ,  $D$  is a diagonal matrix defining rotor drag coefficients  $D = \text{diag}(d_x, d_y, d_z)$ ,  $\odot$  denotes the multiplication of a vector and quaternion, and the inertia matrix  $I$  of the UAV is given by  $I = \text{diag}\{I_{xx}, I_{yy}, I_{zz}\}$  in which each  $I_* \in \mathbb{R}$  is the moment of inertia about each of the UAV's body-axes.

The following equation is used to estimate the UAV's inertia, which we use throughout the simulation.

$$\hat{I} = \frac{m}{12}l^2 D \quad (15)$$

For the observation state of the policy, Brownian motion noise is added to the linear and angular velocities, simulating both a random walk and a per-timestep disturbance. The noise added to the environment is somewhat complicated to write down mathematically, so a GitHub link is provided for those who wish to view here.

Each timestep, the controllers receives a state  $S = [P, \Omega, P', \Omega']$ ,  $S \in \mathbb{R}^{12}$ , and provides the desired body torque and collective thrust  $A = [a_0, a_1, a_2, a_3]$ ,  $A \in \mathbb{R}^4$ .

The values we use for our UAV are provided in Table ??

All results are provided and calculated using either the Mean Squared Error (MSE) or Sum of Squared (SSE) errors. Thus, both SSE and MSE are given in  $m^2$  units. These are all calculated as the mean error across all axes.

TABLE I: Quadcopter Model Parameters

| Symbol                  | Value                 |
|-------------------------|-----------------------|
| $g$                     | 9.81 m/s <sup>2</sup> |
| $\alpha^{-1}$           | 0.0125 s              |
| $\Delta t_{\text{sim}}$ | 0.008 s               |
| $M_c$                   | $7.84 \times 10^{-6}$ |
| $k_i$                   | $1.75 \times 10^{-4}$ |
| $m$                     | 2.0 kg                |
| $l$                     | 0.225 m               |
| $D$                     | diag(4.5, 4.5, 7)     |

### C. Experiments

Our custom simulation environment is designed as a Markov Decision Process (MDP) chain, which integrates UAV dynamics. It is possible to reset the simulation, view rewards, apply control actions, and retrieve observations at each timestep. This allows for easy integration with the RVO optimisation step, in which the cumulative sum of rewards serves as the loss function for each rollout, and the simulation can be reset for each particle. The simulation operates with an 8ms interval for physics updates and control loops. By framing our simulation as an MDP, we ensure reproducibility and ease of scenario configuration. All simulations were run on a 32-core computer without using a GPU. Each simulation rollout took a few seconds, so the total optimisation and testing of the generated controls took around 5 minutes per iteration. For our full simulation, we ran for 100 generations, using a population size of 30. For the PSO parameters, we used 300 particles with 10 dimensions. For fine-tuning the final model, we used 10,000 particles across 10 dimensions.

### D. Trajectory Tracking

The primary task for the generated controller was to accurately track an oscillating lemniscate trajectory. We chose a lemniscate as the reference position, as it is a complex trajectory to follow. The lemniscate requires the UAV to perform continuous turns in opposite directions, thereby exciting the UAV's dynamics and challenging the controller's response. We define the lemniscate trajectory as:

$$P_r^t = \begin{bmatrix} rad \sin(\theta(t)), \\ rad \sin(2\theta(t)), \\ z_0 + \sigma(t) A \sin(\theta(t)) \end{bmatrix} \quad (16)$$

with  $rad$  controlling the circle radius,  $z_0$  controlling the height,  $t$  is the timestep multiplied by  $\Delta t$ ,  $A$  controlling the vertical offset, and  $\theta(t)$  provided as

$$\theta(t) = 2\pi \frac{t}{T}. \quad (17)$$

Our experiment is designed so that each rollout completes in 40 seconds, yielding 10,000 simulation timesteps. As noted in prior work, lemniscate trajectories exhibit rapid, coordinated attitude changes while maintaining stability. The initial state for the quadcopter at  $t = 0$  is provided as some distance away from the trajectory's initial position:  $P_x = 1$ ,  $P_y = -1$ . Additionally, the controller starts with a large yaw offset. Implementing an initial positional offset and rotational bias allows us to estimate the controller's performance for large and

small reference errors within a single simulation run. Tracking performance is evaluated using a standard metric: the Sum of Squared Error of the position coordinates

$$L(\pi) = \sum_t^T (P^t - P_r^t)^2. \quad (18)$$

### E. Lissajous Trajectory

The controllers are also tested on a Lissajous trajectory. The Lissajous trajectory is not observed by any controller in the training environment, ensuring that the controllers and parameters are not overfitting to any single trajectory. The Lissajous trajectory is a good choice for our Out-Of-Distribution (OOD) testing, as it provides a strong snap impulse error to the controller reference error. Testing the controllers on two complex trajectories, each with its own difficulties, demonstrates the controllers' robustness across various trajectory types.

We formulate the trajectory reference as

$$P_r^t = \begin{bmatrix} R_x \sin(\kappa_x t) \\ R_y \sin(\kappa_y t + \phi_y) \\ A_z \sin(\kappa_z t + \phi_z) \end{bmatrix} \quad (19)$$

with  $\kappa_x$ ,  $\kappa_y$ , and  $\kappa_z$  controlling the frequency of oscillations across each axis. No starting position offset is used for the Lissajous trajectory.

### F. PID+DOB controller

One baseline controller used to compare the generated model's performance is a PID cascade controller with a Disturbance Observer (DOB), which provides online parameter updates to compensate for model inaccuracies.

PSO-optimised gains (gathered from running the PID+DOB controller in the training environment) for the PID cascade are provided as:

TABLE II: PID Cascade Gains

| Controller       | P Gain  | I Gain | D Gain  |
|------------------|---------|--------|---------|
| Position X       | 2.48209 | N/A    | 2.02742 |
| Position Y       | 3.53132 | N/A    | 1.62606 |
| Position Z       | 7.12605 | N/A    | 1.62871 |
| Orientation      | 12      | 0      | 0       |
| Angular Velocity | 4.5     | 4.5    | 4.0     |

The **I** term is not used within the velocity and position controller, with the multiplicative integration value instead replaced directly by the DOB output.

As a quadcopter is underactuated and only produces collective thrust for positional translations, we convert the required linear accelerations from the velocity PD+DOB into orientations through the formula:

$$\omega_t^x = \text{atan2} (||V_t'^x||, ||V_t'^z||) \quad (20)$$

$$\omega_t^y = -\text{atan2} (||V_t'^y||, ||V_t'^z||) \quad (21)$$

$$\omega_t^z = 0 \quad (22)$$

using small-angle approximation. Here,  $\omega_t^x$  is the target roll,  $\omega_t^y$  the pitch target, and  $\omega_t^z$  the yaw target.

To convert the desired angular velocity into body torque for the fixed frame, we use the following equation, derived and provided by the equations within the simulation:

$$b = IG\omega_t' + \omega' \times (I\omega'). \quad (23)$$

Here,  $G$  represents a diagonal matrix of  $P$  gains for the conversion from angular velocity into desired body torque ( $b$ ).

#### G. LQR controller

Each timestep, the LQR controller receives a state  $x$

$$x = \begin{bmatrix} P_x & P_y & P_z & P'_x & P'_y & P'_z & \phi & \theta & \psi & p \\ q & r & & & & & & & & \end{bmatrix}^\top \in \mathbb{R}^{12}, \quad (24)$$

where  $P = [x \ y \ z]^\top$  is the position,  $P' = [P'_x \ P'_y \ P'_z]^\top$  the linear velocity,  $\eta = [\phi \ \theta \ \psi]^\top$  the roll, pitch, and yaw Euler angles, and  $\omega = [p \ q \ r]^\top$  the body angular rates.

The control input vector  $u$  defined as

$$u = [T \ \tau_x \ \tau_y \ \tau_z]^\top. \quad (25)$$

where  $T$  is the total collective thrust and  $\tau_x, \tau_y, \tau_z$  are the body torques.

The nonlinear dynamics are linearised into a state-space model around the hover point using  $A$  and  $B$ :

$$\dot{x} = Ax + Bu, \quad (26)$$

with the  $A$  given by

$$A = \begin{bmatrix} 0_{3 \times 3} & I_3 & 0_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & 0_{3 \times 3} & 0 & g & 0 \\ 0_{3 \times 3} & 0_{3 \times 3} & -g & 0 & 0 & 0_{3 \times 3} \\ 0_{3 \times 3} & 0_{3 \times 3} & 0 & 0 & 0 & 0_{3 \times 3} \\ 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & I_3 \\ 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \end{bmatrix}, \quad (27)$$

and  $B$  provided as

$$B = \begin{bmatrix} 0_{3 \times 4} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0_{3 \times 4} \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix}, \quad (28)$$

We formulate the state weighting matrix  $Q$  as

$$Q = \text{diag}(Q_{\text{pos},xyz}, Q_{\text{vel},xyz}, Q_{\text{ang},xyz}, Q_{\text{rate},xyz}), \quad (29)$$

and the control effort matrix  $R$  as

$$R = \text{diag}(R_{\text{thrust}}, R_{\text{torque},xyz}). \quad (30)$$

The optimal feedback gain is obtained from the continuous-time Algebraic Riccati Equation (ARE)

$$A^\top P + PA - PBR^{-1}B^\top P + Q = 0, \quad (31)$$

whose solution  $P \in \mathbb{R}^{12 \times 12}$  yields the optimal gain

$$K = R^{-1}B^\top P \in \mathbb{R}^{4 \times 12}. \quad (32)$$

The ARE is solved iteratively by integrating

$$\dot{P} = A^\top P + PA - PBR^{-1}B^\top P + Q. \quad (33)$$

To implement the controller, let

$$x_d = [p_d^\top \ v_d^\top \ \eta_d^\top \ \omega_d^\top]^\top \quad (34)$$

denote the desired state. The state error is defined as

$$e = \begin{bmatrix} p - p_d \\ v - v_d \\ \eta - \eta_d \\ \omega - \omega_d \end{bmatrix}. \quad (35)$$

The control input is computed as

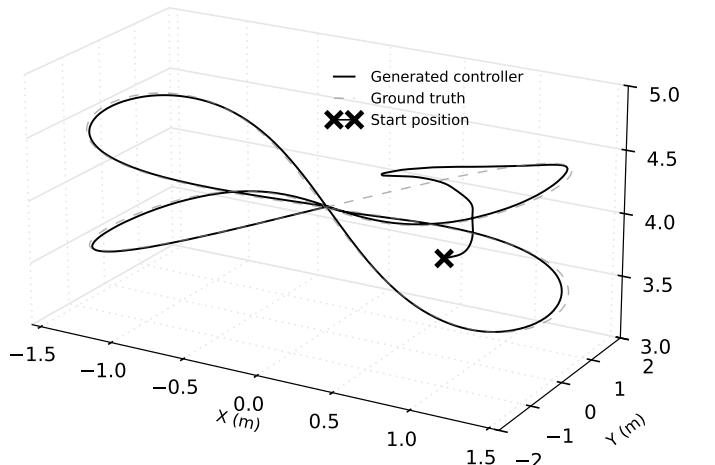
$$u = u_{\text{eq}} - Ke, \quad (36)$$

with the hover equilibrium input

$$u_{\text{eq}} = \begin{bmatrix} mg \\ 0 \\ 0 \\ 0 \end{bmatrix}. \quad (37)$$

The state and control gains  $Q$  and  $R$  are obtained by running the controller in the training environment and estimating via PSO. We perform this tuning the same across all controllers.

## IV. TRAINING RESULTS



**Fig. 2: Training results:** Generated control system output in the training environment vs baseline trajectory. The trajectory to be followed is shown as a dotted line. Also indicated here is the large positional offset the controller begins with, allowing for estimation of controller performance to both large and small reference discrepancies in a single simulation run.

Training results for the generated controller are provided in Figure ???. The final SSE for the generated controller in the training environment was  $62.433m^2$  (after fine-tuning with PSO), with the initial controller achieving an MSE of  $1652.80m^2$ . The generated controller demonstrates accurate path following for the lemniscate trajectory. Qualitatively, the UAV's motion appears smooth and closely matches the desired trajectory, with no apparent issues. This outcome is comparable to advanced controllers in the literature that follow lemniscate trajectories with minimal error.

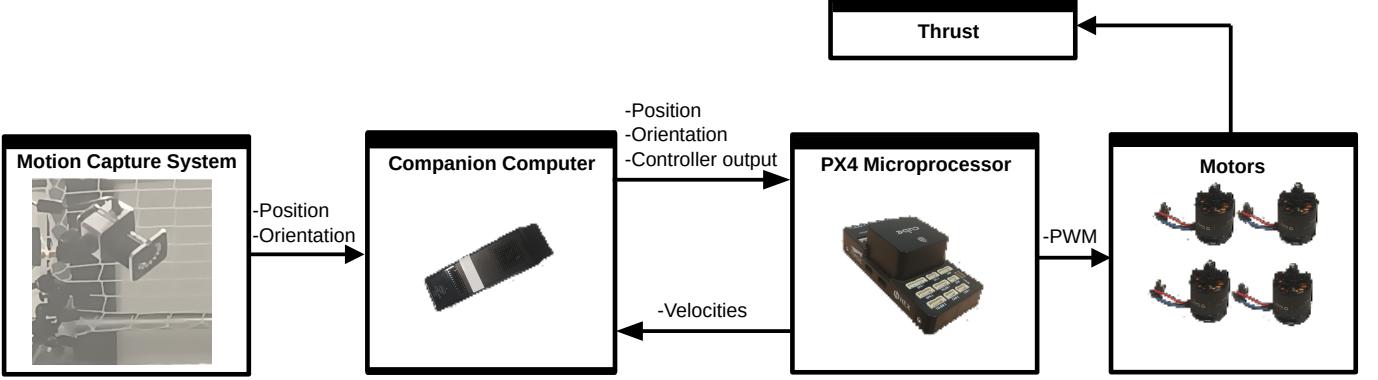


Fig. 3: Real-world deployment of the generated control system onto a UAV with a motion capture system. The motion capture system sends precise position and orientation information to the companion computer through a TCP connection in an ENU coordinate frame. The companion computer converts the ENU frame into a NED frame and publishes it to the PX4 board. The PX4 board sends vehicle odometry gathered from an Extended Kalman Filter (EKF) back to the companion computer, which subsequently sends body-torque and collective thrust commands to the microprocessor. The microprocessor performs control allocation to convert the normalised thrust commands into PWM commands and interfaces them to the motors. The motors produce thrust, which alters the forces acting on the UAV, thereby changing its position and orientation.

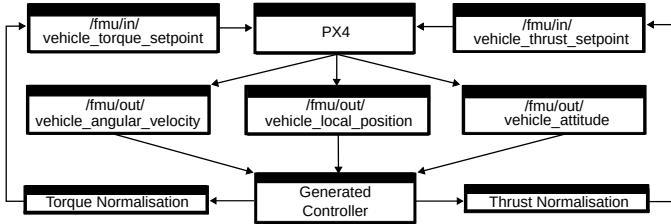


Fig. 4: **PX4-ROS 2 interaction:** Interaction between PX4 and the ROS2 nodes via a publisher/subscriber design pattern.

#### A. Software-in-the-loop (SITL) testing

As the generated controller produced acceptable results in the training environment, we prepared the algorithm for real-world deployment by transitioning it into a ROS 2 node that interfaces with PX4. PX4 enables SITL testing before deployment with high-fidelity simulations that integrate seamlessly into PX4 firmware. The SITL simulations offer a form of OOD testing and are higher quality and offer more insight into actual UAV performance than our training environment [?], [?].

The ROS2 node was integrated into PX4 Autopilot using the Micro-XRCE-DDS bridge. PX4 utilises a publisher/subscriber design pattern to exchange information between the onboard flight controller and an offboard controller. This integration allows us to treat the generated controller as a part of the PX4 firmware, benefiting from low-latency communication and access to internal state estimations. The generated controller, as a ROS 2 node, required some minor implementation changes before it functioned as intended. PX4 requires that the controller output is both normalised relative to the motor output and in a NED world frame. The generated controller, as designed, outputs absolute torque and thrust values (N) in a FLU world frame. We send the thrust and torque commands to PX4 at a rate of 8ms, with the controller receiving sensor updates at the same rate.

A diagram illustrating the interaction between the generated

controller and PX4 via PX4 topics is shown in Figure ??.

TABLE III: Per-lap Performance Metrics Comparison SITL

| Controller               | MSE                        | Std. dev                   | Best MSE                   |
|--------------------------|----------------------------|----------------------------|----------------------------|
| <b>Lemniscate (SITL)</b> |                            |                            |                            |
| Generated Controller     | <b>0.0304m<sup>2</sup></b> | 0.0031m <sup>2</sup>       | <b>0.0245m<sup>2</sup></b> |
| PID+DOB                  | 0.0418m <sup>2</sup>       | <b>0.0012m<sup>2</sup></b> | 0.0393m <sup>2</sup>       |
| LQR                      | 0.0533m <sup>2</sup>       | 0.0143m <sup>2</sup>       | 0.0744m <sup>2</sup>       |
| <b>Lissajous (SITL)</b>  |                            |                            |                            |
| Generated Controller     | <b>0.0084m<sup>2</sup></b> | <b>0.0006m<sup>2</sup></b> | <b>0.0071m<sup>2</sup></b> |
| PID+DOB                  | 0.0107m <sup>2</sup>       | 0.0008m <sup>2</sup>       | 0.0088m <sup>2</sup>       |
| LQR                      | 0.0116m <sup>2</sup>       | 0.0017m <sup>2</sup>       | 0.0090m <sup>2</sup>       |

Table ?? quantitatively summarises the results of 10 laps in the SITL testing environment for both the lemniscate and Lissajous trajectories.

On the lemniscate trajectory, the generated control system offered a mean performance increase of 27.27% compared to the best-performing hand-made controller (PID+DOB). The worst lap of the generated control system outperformed the best lap of the PID+DOB control system. The reduced variance and standard deviation within laps of the PID+DOB control system are likely due to the DOB controller reaching its disturbance-estimation limits in the presence of model inaccuracies. For the generated control system, no disturbance observer or online model estimation is available, which contributes to the increased variation per lap.

SITL results are displayed visually in Figure ??.

#### V. DEPLOYMENT AND REAL WORLD RESULTS

The UAV is built using a Holybro Pixhawk 6X, which houses and runs the PX4 firmware. A USB connecting the companion computer and flight controller are plugged into the Holybro Pixhawk 6X. For real-world deployment, instead of using the position and orientation information provided by the PX4 IMU, we use a VICON motion tracking system, which overrides the PX4-provided estimates. The motion tracking system provides more accurate estimates of the UAV's

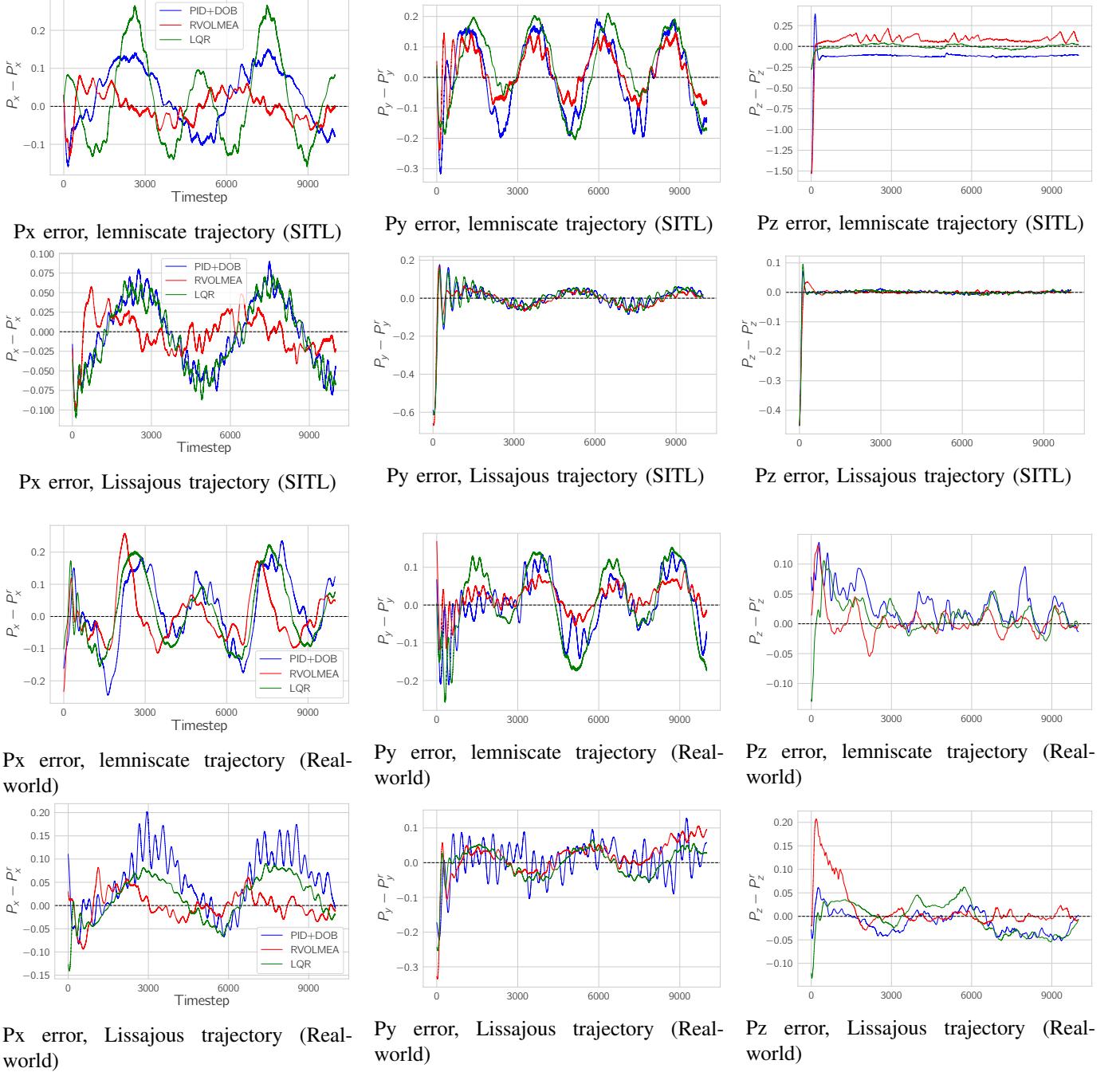


Fig. 5: **Results:** Comparison of SITL and real-world tracking errors for the lemniscate and Lissajous trajectories. The label RVOLMEA is used to identify the generated control system. Variation between SITL and real-world results is expected, given the change in environments. Consistent with the existing literature, the SITL results are closer to real-world results than those from the training environment. The figures show the reference error over time for the best lap of any given controller (the lowest MSE lap).

position, velocity, and orientation. Figure ?? describes our deployment setup in a visual format.

The results of the experiments for the real-world experiments are also visualised in Figure ??.

A results Table for the real-world results is provided in Table ???. Within the results, the average is taken over 10 laps of the trajectory. ‘MSE’ refers to the MSE of all laps, and ‘Best MSE’ is the lap with the lowest MSE. The generated controller

yielded strong results in the real world. An MSE of  $0.0098 m^2$  precision within the lemniscate trajectory highlights the strengths of our methodology, whilst also showcasing areas for improvement. We specifically highlight the area for improvement in our results, namely, the direct deployment and iteration in the real world.

The generated controller exhibited greater robustness than the PID+DOB. All runs were performed across two charge

TABLE IV: Per-lap Performance Metrics Comparison Real world

| Controller                     | MSE                        | Std. dev                   | Best MSE                   |
|--------------------------------|----------------------------|----------------------------|----------------------------|
| <b>Lemniscate (REAL WORLD)</b> |                            |                            |                            |
| Generated Controller           | <b>0.0098m<sup>2</sup></b> | <b>0.0010m<sup>2</sup></b> | <b>0.0090m<sup>2</sup></b> |
| PID+DOB                        | 0.0329m <sup>2</sup>       | 0.0125m <sup>2</sup>       | 0.0187m <sup>2</sup>       |
| LQR                            | 0.0261m <sup>2</sup>       | 0.0028m <sup>2</sup>       | 0.0201m <sup>2</sup>       |
| <b>Lissajous (REAL WOLRD)</b>  |                            |                            |                            |
| Generated Controller           | <b>0.0084m<sup>2</sup></b> | <b>0.0006m<sup>2</sup></b> | <b>0.0071m<sup>2</sup></b> |
| PID+DOB                        | 0.0116m <sup>2</sup>       | 0.0008m <sup>2</sup>       | 0.0088m <sup>2</sup>       |
| LQR                            | 0.0116m <sup>2</sup>       | 0.0017m <sup>2</sup>       | 0.0090m <sup>2</sup>       |

cycles of a lithium-ion battery, with each run using approximately 20% of the battery charge. We observe sensitivity to battery charge across runs in the PID+DOB controller. The achieved Z-axis component is reduced as the battery charge decreases. Unlike the PID+DOB controller, the generated control system shows no sensitivity to the charge, with the achieved accuracy remaining consistent throughout the experiments.

The achieved accuracy is higher for the generated control system than for the baseline controller. Across the X and Y axes. The generated control system achieved close agreement with the target tracking across the Z-component throughout all experiments.

It is possible that both control systems can achieve better performance through real-world specific tuning. It is important to note that these results are obtained without any tuning in either SITL testing or the real world. The entire process is automatic: the gains for the control systems are determined using PSO within the training simulator.

## VI. CONCLUSION

In conclusion, we have designed and tested an LLM-synthesised control system using two simulators. The simulators showed that the generated control system is reliable for unmanned flight.

After testing the control system in the real world, we provide proof that generated code can be used in a Reinforcement Learning-esque fashion to create functional real-world code.

Because we used a training environment, the generated controllers are subject to the same limitations as reinforcement learning, such as overfitting to the training environment (known as the sim-to-real gap). We base this conclusion on a 3x reduction in performance from the training environment to the real world. Also notable is the lack of online parameter estimation for the generated control systems. The cause is primarily our methodology. No online parameter estimation is required, as PSO already optimises the model towards our known model parameters during the simulation.

The choice of our task, unmanned flight, reflects the growth and capabilities of LLMs to synthesise high-quality code across many queries.

This work opens the door to the application of agentic AI in engineering design and to the creation of many different control systems for the real world. Primarily, the following steps in this work are the creation of certified control systems, which can each be tested in the real world without the need for an intermediate testing environment.

Creating a methodology for unmanned flight is difficult. Simulated training environments are almost always required because poor control systems can lead to catastrophic failure in the real world. It would be interesting to apply this methodology directly to the real world, using online parameter estimation rather than PSO. This way, the limitations identified in this paper (overfitting and lack of generalisation) can be mitigated through real-world scoring and iteration. These control systems would need to be implemented on an unmanned vehicle, such as a TurtleBot, that is not in any intrinsic danger, so that faulty control systems can be easily discarded without damaging the machine executing them. A link for viewing the implementation of all controllers with the lemniscate trajectory is provided here. The generated control system can be found under the function ‘rvolmeaControl’.