

# $\pi$ -Flight: Automated Synthesis of Interpretable UAV Control Programs via Monte Carlo Tree Search

Author Names Omitted for Review

Affiliations Omitted for Review

**Abstract**—Designing reliable and agile controllers for unmanned aerial vehicles (UAVs) is challenging: handcrafted PID or model-based designs are brittle in cluttered environments, while large neural policies are hard to verify and interpret. This paper presents  $\pi$ -Flight, a framework that automatically synthesizes *interpretable* UAV control programs from a domain-specific language (DSL). Each control program is a structured decision rule over measured states and control primitives, constrained by a safety shell that enforces thrust and attitude limits. A graph neural network (GNN) encodes program structure and predicts value priors, and Monte Carlo Tree Search (MCTS) explores the discrete program space guided by these priors and rollouts in a high-fidelity simulator. We demonstrate on agile flight benchmarks that  $\pi$ -Flight discovers compact programs that achieve comparable or better tracking performance than tuned baselines, while exposing program logic that can be inspected and manually adjusted. The safety shell guarantees hard constraints on control amplitudes, and the MCTS+GNN coupling allows efficient search over thousands of candidate programs per iteration.

**Index Terms**—UAV control, interpretable control program, Monte Carlo Tree Search, graph neural networks, safety constraints

## I. INTRODUCTION

UAVs are increasingly deployed in agile and safety-critical tasks such as inspection, search-and-rescue and autonomous cinematography. In such settings, controllers must satisfy three properties simultaneously: (i) strong tracking and disturbance rejection, (ii) respect of safety limits on thrust and attitude, and (iii) interpretability to human operators. Classical PID or model-based controllers meet (ii) and are interpretable, but require extensive manual tuning and degrade under model mismatch. End-to-end neural network policies trained by reinforcement learning can achieve impressive agility, yet their internal representations are opaque and difficult to certify.

This work proposes  $\pi$ -Flight, an automatic synthesis framework for UAV control programs. Instead of representing the controller as a black-box network, we define a domain-specific language (DSL) of symbolic control programs built from measurable state variables (e.g., position and velocity errors) and primitive control operations (e.g., proportional and integral terms, saturations, gains). A controller is a small program in this DSL, which can be rendered as human-readable pseudo-code. A safety shell wraps any such program and enforces constraints on total thrust, torque and their rates of change, ensuring that synthesized controllers remain within hardware and stability limits.

Searching directly over all possible programs is combinatorial and intractable.  $\pi$ -Flight addresses this by combining Monte Carlo Tree Search (MCTS) with a graph neural network (GNN):

- MCTS performs look-ahead search in the discrete space of partial programs, branching on grammar expansions and program edits.
- The current partial program is encoded as a graph (abstract syntax tree with typed nodes and edges), and a GNN produces both value estimates and structural priors to guide tree expansion.
- Each completed candidate program is evaluated in a parallel UAV simulator, using the safety shell during rollout to clip unsafe commands while still exposing the underlying program behaviour.

By iterating between search, simulation-based evaluation and GNN updates,  $\pi$ -Flight learns to propose promising program structures and efficiently refines them. The resulting controllers are compact programs that can be inspected, debugged and adjusted by control engineers.

The main contributions of this paper are:

- A program synthesis view of UAV control, based on a safety-aware DSL of interpretable control programs.
- A MCTS+GNN framework for exploring the space of control programs, where programs are represented as graphs and evaluated in parallel physics simulation.
- An empirical study on agile UAV flight tasks showing that synthesized programs match tuned baselines in tracking performance while providing explicit structural priors and safety envelopes.

## II. PROBLEM FORMULATION

We consider a quadrotor UAV modeled by a continuous-time nonlinear system

$$\dot{x} = f(x, u), \quad (1)$$

where  $x$  collects position, attitude and linear/angular velocities, and  $u$  denotes the four rotor thrust commands. A reference trajectory  $x^{\text{ref}}(t)$  is given, and the control objective is to design a feedback law  $u = \pi(x, x^{\text{ref}})$  that minimizes tracking error while respecting control constraints and providing interpretable structure.

In  $\pi$ -Flight, the policy  $\pi$  is not an arbitrary function but a control program  $P$  written in a DSL. The DSL exposes observable signals such as position and velocity errors and

combines them using a small set of operators (weighted sums, nonlinearities, saturations and gains). Any program  $P$  is executed at a fixed control rate and produces a raw command  $\tilde{u} = P(x, x^{\text{ref}})$ , which is passed through a safety shell

$$u = \mathcal{S}(\tilde{u}), \quad (2)$$

that clips total thrust and torques and limits rates of change. This shell enforces hard constraints independent of the internal program structure.

We optimize over programs  $P$  to minimize a trajectory-level cost

$$J(P) = \mathbb{E}[\Phi(x_{0:T}, u_{0:T})], \quad (3)$$

where  $\Phi$  penalizes tracking error, energy and constraint violations along simulated rollouts. Expectations are approximated by Monte Carlo runs in a high-fidelity simulator. The search space for  $P$  is combinatorial; we therefore employ MCTS guided by a GNN-based value function over program graphs.

### III. METHODOLOGY: $\pi$ -FLIGHT FRAMEWORK

This section describes the main components of  $\pi$ -Flight: the control program DSL, the safety shell, the GNN representation of programs and the MCTS-based search procedure.

#### A. Control Program DSL

The DSL defines how control programs are constructed. Programs are abstract syntax trees whose leaves are typed signals (e.g., position error along each axis, velocity error, attitude error), and whose internal nodes are operations such as weighted sum, scaling, saturating nonlinearities and simple conditional selections. A small number of production rules specify how trees can grow, ensuring that all programs are syntactically valid and type-consistent.

Each instantiated program corresponds to a concise pseudo-code controller that a human engineer can inspect. For example, a program may encode “apply a proportional gain on vertical position error plus a damping term on vertical velocity, then saturate total thrust within hardware limits”. By operating at the level of symbolic programs,  $\pi$ -Flight exposes the structure of the controller instead of burying it in continuous weights.

#### B. Safety Shell

Regardless of the program structure, the generated command must respect actuator and safety limits. The safety shell  $\mathcal{S}$  wraps any program and enforces:

- bounds on total thrust and individual rotor commands,
- limits on roll and pitch angles implied by thrust distribution,
- smoothness via rate limits on command changes.

These constraints are implemented as deterministic clipping and redistribution operations that are inexpensive to evaluate. Importantly, they do not depend on the internal parameters of  $P$ , so any synthesized program is guaranteed to operate within the same safety envelope.

#### C. Program Graph Representation and GNN

To guide search, we embed programs as graphs. Each node corresponds to a DSL symbol (signal or operator) with type annotations, and edges represent parent–child relations in the syntax tree as well as lateral connections capturing shared sub-expressions. A GNN processes this graph to produce:

- a scalar value estimate  $V(P)$  approximating negative cost  $-J(P)$ ,
- logits over possible grammar expansions at frontier nodes, which define a prior over MCTS actions.

The GNN is trained online from the outcomes of simulated rollouts: programs that achieve lower cost provide targets to update  $V$  and the expansion priors. This yields structural priors that prefer, for instance, reusing beneficial sub-expressions or adding damping terms when oscillations are detected.

#### D. MCTS in Program Space

MCTS operates over partial programs. Each tree node represents a partially expanded program graph, and actions correspond to applying a DSL production rule at a selected frontier node. The search alternates between selection (using an upper-confidence bound that combines GNN priors and visit counts), expansion (adding a new node or operation), simulation (completing the program and evaluating it in the simulator) and backpropagation of the obtained cost.

To amortize the cost of physics simulation, we evaluate many candidate programs in parallel using a GPU-based UAV simulator. For each iteration, a batch of promising leaf programs is selected from the search tree, rolled out under the safety shell across multiple randomized initial conditions, and aggregated into an empirical cost estimate. These evaluations feed back into both the MCTS statistics and the GNN training set.

## IV. EXPERIMENTS

We summarize training progress and evaluation throughput using the latest training session.

#### A. Learning Curve

Fig. 1 shows the real reward vs. iteration.

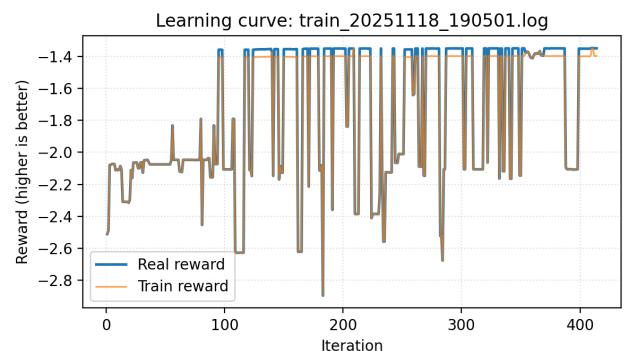


Fig. 1. Learning curve (real reward over iterations).

## B. Evaluation Throughput

Fig. 2 plots ms/program across iterations; Fig. 3 shows the distribution.

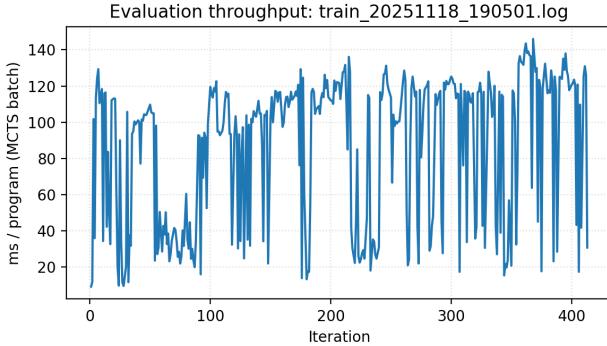


Fig. 2. Evaluation throughput (lower is better).

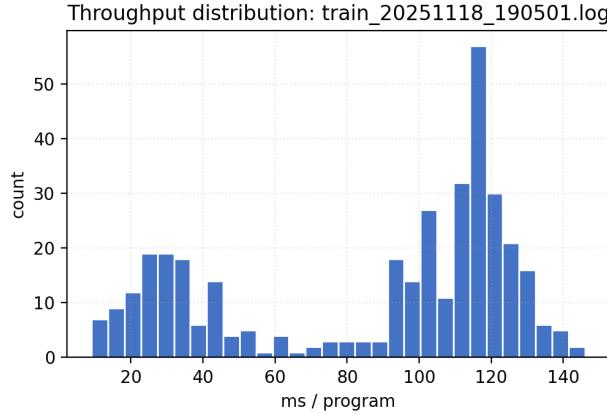


Fig. 3. Distribution of ms/program across iterations.

## C. Summary

Table I summarizes the latest run.

TABLE I  
SUMMARY OF LATEST TRAINING RUN (TRAIN\_20251118\_190501.LOG)

Best real reward	-1.3472
Last real reward	-1.3497
Throughput mean (ms/program)	86.8
Throughput p50 / p90 / p95	104.5 / 125.2 / 129.6

## ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

## REFERENCES

## REFERENCES

- [1] Placeholder reference for draft compilation.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.