Learned heuristics for a time-optimal online motion primitive planner

Helene J. Levy, Grace J. Kwak, Brett T. Lopez

***Abstract*— PhD student Helene J. Levy and Assistant Professor Brett T. Lopez are currently developing a novel planning strategy to intelligently sample the state space using motion primitives when given a coarse reference path generated by a discrete graph-based search algorithm. By exploiting key information inherent to the reference path, the proposed planner can strategically sample motion primitives in regions known to make progress to the goal, leading to significant improvements in computation time and robustness in trajectory generation. Additionally, this framework allows for sampling in a higher dimensional state space, i.e. acceleration, which contributes to smooth kinodynamic path plans. However, the number of primitives that the planner must generate grows exponentially with the number of path waypoints and the number of state samplings. Thus, the planner quickly becomes infeasible for online deployment. The first half of this thesis presents an in-depth analysis of selected primitives and several algorithms for benchmarking performance (greedy, greedy lookahead, and random). The second half presents initial results using a fully-connected neural network in the behavior cloning (supervised learning) paradigm.**

I. INTRODUCTION

A. Societal context

Urgent real-world missions, such as those in emergency response and search-and-rescue operations, require a highly flexible task force. Leveraging the agility, speed, and advanced capabilities of UAVs like autonomous quadcopters, these missions can efficiently tackle urgent situations. In regions affected by disasters, drones can rapidly assess damage, find survivors, and deliver essential resources, potentially preventing loss of lives caused by delays in standard relief procedures. Drones' capability to quickly collect and send live data results in faster decision-making and improved resource distribution. Nevertheless, the use of self-governing drones for urgent tasks presents significant ethical and privacy issues, as the need for quick action must be weighed against the impact on personal freedoms and the importance of employing this technology responsibly. In addressing these challenges, it is essential for society to establish clear regulations, ethical guidelines, and international collaboration in order to fully utilize autonomous drones for the common good and minimize potential risks.

B. Technical context

Real-time deployment of autonomous vehicles in large and complex environments requires efficient, localized path planning. While quadrotors have achieved top speeds in piloted competitions, autonomous motion planning algorithms have struggled to achieve the low computation time necessary for practical execution of such fast flights. Motion planning requires ensuring safety (collision checking) as well as generating high quality, dynamically feasible paths for a quadrotor to traverse. It is further complicated by payload constraints requiring computationally light algorithms for the onboard hardware. An effective strategy at lowering computation time has been the use of so-called motion primitives: a library of pre-determined actions or paths computed offline but selected online. Motion primitives are short optimized trajectories with a fixed time horizon allowing for quick generation compared to single trajectory optimization methods.

Helene J. Levy and Brett T. Lopez are with the Department of Mechanical and Aerospace Engineering and Grace J. Kwak is a student in the Department of Electrical and Computer Engineering, University of California Los Angeles, Los Angeles, CA, USA. {hjlevy, btlopez, gracekwak}@ucla.edu

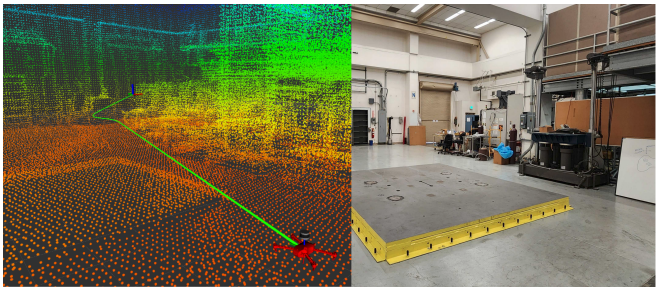


Fig. 1: Path plan through maze of boxes in VECTR lab at UCLA. The informed motion primitive planner takes in point cloud data and runs a coarse graph search on the enviroment. After that, motion primitives are intelligently sampled given information from the coarse path to generate a smooth, collision free trajectory.

C. Contributions

This capstone thesis makes the following contributions:

1) A greedy, greedy-lookahead, and random state sampling strategies
2) An in-depth analysis of selected state samples
3) Infrastructure for learning-enabled state sampling features
4) A simple fully-connected neural network for state sampling

II. STATE OF THE ART

The integration of vehicle dynamics for motion planning has been a substantial and extensively explored area of research. Differentially flat systems such as quadrotors are able to take advantage of polynomial trajectory formulations generated by solving minimum jerk or snap optimization problem [1], [2]. One approach to generate an optimal trajectory for a large environment is to obtain waypoints from coarse search-based or sampling-based techniques and then optimize the trajectory between these points [3], [4]. Other methods involve formulating the free-space of environment

into convex polyhedra or representing trajectories as convex sets and then solve a convex optimization problem [5]–[7]. While these paths are well constructed, solving optimization problems online is generally computationally intractable and may lead to numeric instabilities, especially in large environments.

Motion primitives emerged as a compelling solution to address the computational complexity of trajectory optimization methods and generate dynamically feasible paths. By imposing a fixed time horizon on the optimization problem, motion primitives may be pre-computed offline, providing a significant advantage in terms of computation efficiency. [8] first introduced an efficient method to generate and check motion primitives for input feasibility. Additionally, [9] showcased the effectiveness of pre-computed motion primitives combined with perception for a high-speed flight applications. While motion primitives are able to achieve computation speeds necessary for fast autonomous platforms, they are inherently myopic, limiting their ability to plan for long-term trajectories.

Kinodynamic motion planning is a field dedicated to addressing motion planning problems which simultaneously solve kinematic and dynamic constraints [10]. The term was coined emphasize the fusion of planners dedicated to each respective constraint. Recognizing that motion primitive generation is essentially a form of sampling, it is unsurprising that numerous kinodynamic planning works have incorporated other sampling-based strategies, such as RRT* and FMT*, into their methodologies [11]–[13]. Other works explore the integration of motion primitives in a kinodynamic formulation through graph search algorithms. [14] was one of the first works to successfully showcase a search-based algorithm using a lattice of motion primitives for use on quadcopters. Additionally, [15] introduced a receding horizon framework employing a hybrid A* search over a set of motion primitives.

III. PROBLEM FORMULATION

A. Notation

Assuming a fixed environment represented by a point cloud, $\mathcal{P}$, we assign a start position, p_{start} and goal position, p_{goal}. Let the collision-free reference path generated by a discrete graph search algorithm be composed of node points $\mathbf{n} = \{n_0, n_1, ..., n_M\}$. The set of points $\mathbf{n}$ is further pruned to a sparse set of waypoints $\mathbf{w} = \{w_0, w_1, ..., w_N\}$ where $N \leq M$. The environment is separated into a set of regions $\mathcal{R} = \{\mathcal{R}_0, ..., \mathcal{R}_{N-2}\}$. For each waypoint $w_i \in [w_1, w_{N-1}]$, there exists an intersecting hyperplane $\mathcal{H}_i$ to divide the neighboring environment into regions $\mathcal{R}_{i-1}$ and $\mathcal{R}_i$. Let the set of unit vectors defining reference path headings be denoted as $\mathbf{r} = \{\hat{\boldsymbol{r}}_0, ..., \hat{\boldsymbol{r}}_{N-2}\}$, where vector $\hat{\boldsymbol{r}}_i$ is associated with the reference path in region $\mathcal{R}_i$. We represent the set of motion primitives generated at each search point as $\mathbf{s} = \{\boldsymbol{s}_0(t), \boldsymbol{s}_1(t), ..., \boldsymbol{s}_L(t)\}$. A motion primitive's state, $\boldsymbol{s}_i(t)$, defined over time horizon, T, is composed of it's position, velocity, and acceleration trajectories, i.e., $\boldsymbol{s}_i(t) = [\boldsymbol{p}_i(t), \boldsymbol{v}_i(t), \boldsymbol{a}_i(t)]^\top,\ \forall t \in [0, T]$.

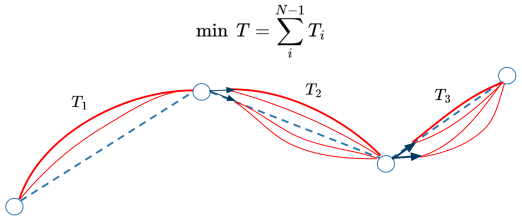


Fig. 2: The blue dotted line represents the sparse A* trajectory our planner uses as a reference. A tree of time-optimal motion primitives is sampled at each waypoint shown in red.

B. System Dynamics for Motion Primitive Generation

Quadcopters are differentially flat systems [16] allowing for the use of polynomial descriptions of flat state variables $\boldsymbol{\sigma} = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^\top$. Here yaw, ψ, is not considered in planning because it does not affect the coordinate system dynamics. In accordance with [2], the quadcopter is represented by a triple integrator system. Let state vector be defined as $\mathbf{x} = \begin{bmatrix} \boldsymbol{x} & \dot{\boldsymbol{x}} & \ddot{\boldsymbol{x}} \end{bmatrix}^\top = \begin{bmatrix} \boldsymbol{p} & \boldsymbol{v} & \boldsymbol{a} \end{bmatrix}^\top$ and the control input be defined as $\mathbf{u} = \dddot{\boldsymbol{x}} = \boldsymbol{j}$. Here, $\boldsymbol{p}$, $\boldsymbol{v}$, $\boldsymbol{a}$, $\boldsymbol{j}$ are position, velocity, acceleration, and jerk respectively. The quadcopter is represented by the following linear system.

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{1}$$

where

$$A = \begin{bmatrix} 0 & \mathbf{I}_3 & 0 \\ 0 & 0 & \mathbf{I}_3 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 \\ 0 \\ \mathbf{1} \end{bmatrix}$$

Following polynomial coefficient generation for motion primitives, they are later pruned according to the following conditions to satisfy thrust constraints.

$$\|\boldsymbol{v}\|_2 \leq v_{max}, \quad \|\boldsymbol{a}\|_2 \leq a_{max}, \quad \|\boldsymbol{j}\|_2 \leq j_{max}.$$

IV. METHODS

Performing a search over several motion primitives can be very time consuming to generate and check each motion primitive for collisions. Thus, in order to achieve real-time speeds on an autonomous vehicle, it is crucial to limit the search to areas more likely to take the vehicle to the goal point. Our planner limits the search by generating motion primitives with information from a reference path known to be collision free.

A. Reference Path Generation

We first use A* algorithm to generate a collision-free reference path, ignoring system dynamics. The path is further decomposed into a sparse set of waypoints, $\mathbf{w}$, by iterating through each node and directly drawing a straight line segment until collision. As shown in Figure 2, the nodes of this sparse path now serve as waypoints and the line segments between them are separated into corresponding regions, $\mathcal{R}$. The waypoints are then are used to build a tree of motion primitives.

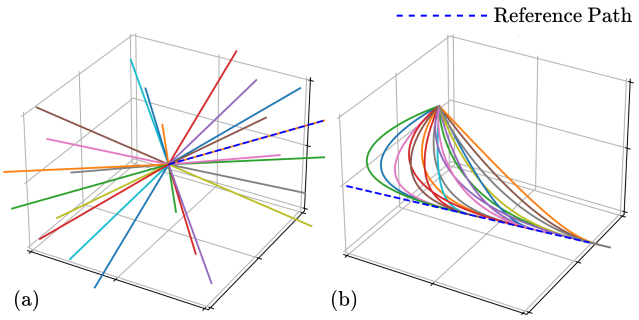


Fig. 3: Expansion of motion primitives (a) shows the set of fixed-end acceleration motion primitives generated from rest. Final accelerations are radially sampled with azimuth $= [-\pi, \pi]$, zenith $= [\frac{7}{18}\pi, \frac{11}{18}\pi]$ and rotation R applied to shift main axis in direction of reference path (b) shows the set of colinear motion primitives generated from rest

B. Motion Primitive Generation

The motion primitives in this work are derived from a minimum time optimization problem. The calculations are repeated three times for each flat variable in the coordinate plane $\begin{bmatrix} x & y & z \end{bmatrix}^\top$. First, we assume a double integrator system. We further refine this path for triple integrator dynamics.

1) Double Integrator: We begin with known waypoints and sample velocities

$$\begin{aligned} \min \quad & J = \int_0^T 1 \; dt \\ \text{s.t.} \quad & \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t) \\ & |u(t)| \leq u_{max} \\ & \mathbf{x}(0) = \mathbf{x}_0, \; \mathbf{x}(T) = \mathbf{x}_f \end{aligned}$$

where $\mathbf{x} = [\, p \; v \,]^\top$ and the control input u is acceleration a.

From pontryagin's minimum principle the solution is known to be of the form

$$u^*(t) = \begin{cases} \pm u_{max}, & 0 \leq t \leq t_s \\ \mp u_{max}, & t_s < t \leq T \end{cases}$$

2) Triple Integrator:

$$\begin{aligned} \min \quad & J = \int_0^T 1 \; dt \\ \text{s.t.} \quad & \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t) \\ & |u(t)| \leq u_{max}, \; |a(t)| \leq a_{max} \\ & \mathbf{x}(0) = \mathbf{x}_0, \; \mathbf{x}(T) = \mathbf{x}_f \end{aligned}$$

where $\mathbf{x} = [\, p \; v \; a \,]^\top$ and the control input u is jerk j.

$$u^*(t) = \begin{cases} \pm u_{max}, & 0 \leq t \leq t_1 \\ 0, & t_1 < t \leq t_2 \\ \mp u_{max}, & t_2 < t \leq t_3 \\ 0, & t_3 < t \leq t_4 \\ \pm u_{max}, & t_4 < t \leq T \end{cases}$$

C. 1D to 3D

We have to solve the double and triple integrator in 3 axes. We are limited by the longest shortest time horizon.

D. Direction Sampling

In order to ensure forward sampling, two primary directions are sampled in velocity space. One is represented by the vector pointing toward the next waypoint and the other being the normal to the separating hyperplane (we call this the "plane normal vector"). Each separating hyperplane is defined to be the axis of symmetry between path segments on either side of the current waypoint and is represented by the set $\mathcal{H}$.

$$\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^3 : \boldsymbol{a}^\top \mathbf{x} + b = 0\} \tag{2}$$

where

$$\begin{aligned} \boldsymbol{a} &= \frac{\hat{\boldsymbol{r}}_i + \hat{\boldsymbol{r}}_{i+1}}{\|\hat{\boldsymbol{r}}_i + \hat{\boldsymbol{r}}_{i+1}\|_2} \\ b &= -\boldsymbol{a}^\top w_{i+1} \end{aligned}$$

E. Pruning Motion Primitives

1) Max State Constraints: The motion primitive path is iteratively sampled for violations of maximum state constraints as discussed in problem formulation.

2) Collision Checking: Collision checking motion primitives adopts the strategy developed in [9], by sampling the path at the next possible collision using a kd-tree search.

F. Time-Saving Pruning Algorithms

The pruning methods above are purely motivated by safety. It is also advantageous to prune out primitives for the sake of planning time, i.e. to prune out (or even, not generate in the first place) primitives which are unlikely to yield the path with the fastest execution time. This is because it is computationally expensive to generate each primitive, taking about 1ms on a standard laptop.

Let s be the number of state samples at each path segment. The "Full" primitive generation algorithm generates all possible primitives at each waypoint. Thus, at waypoint number w, the number of generated primitives $p = s^w$. With the goal of minimizing both planning time and execution time relative to the Full algorithm, I developed and analyzed the motion primitive pruning algorithms below.

1) Greedy: Before diving into a learning-enabled approach, I was advised to develop algorithms that used simple heuristics in order to be used as benchmark performance evaluators. My first such algorithm was a greedy algorithm.

In contrast to the Full algorithm, at each path segment, the Greedy algorithm selects the primitive with the minimum execution time and prunes out all other primitives. As a result, the number of generated primitives p at waypoint w is s, which is constant. 4

2) Greedy Lookahead: Motivated by the result from the Greedy algorithm, I implemented a "Greedy Lookahead" (GLA) algorithm which would aim to address the shortcomings of the former.

The GLA algorithm works as follows, and stops after processing the second-to-last waypoint:

1) Generate all primitives for next 2 waypoints.
2) Find lowest-cost 2-segment path.

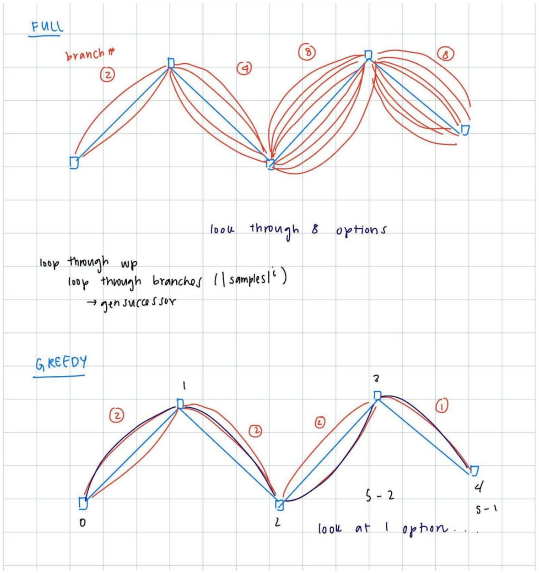


Fig. 4: Diagram of the Full algorithm and the Greedy algorithm.

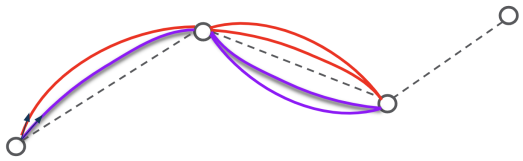


Fig. 5: An illustrative example for when there are 2 state sample options. GLA will generate 6 total primitives and find the primitive pair (one parent and one child) with the minimum total execution time (in this example, the shadowed purple parent and shadowed purple child). On the next iteration, GLA will only expand the lowest-cost child and its siblings (the two purple primitives).

3) Only keep the parent and its children.

(Figure 5)

3) Learning-enabled: Neural networks are non-linear function approximators that learn complex data distributions from inputs to outputs. By simply analyzing the path waypoints, it seems possible for a neural network to predict what endstate (v_f and a_f) to sample at each path segment.

The distributions of selected endstates for v_f (Figure 6) and a_f (Figure 7) immediately reveal two trends that can be learned from the path waypoints alone. As the waypoints get closer together ("scale" increases), the Full planner selects more primitive endstates with lower v_f magnitude. As waypoints get farther apart ("scale" decreases), the Full planner selects more primitive endstates with higher a_f magnitude.

As such, I chose the path waypoints and the initial state as the inputs to the neural network. I chose the outputs to be the predicted v_f and a_f magnitude and direction relative to the plane normal vector defined above. (Figure 8)

As proof of concept, I implemented a simple neural network architecture:

2 Fully-connected hidden layers

(including Batch normalization, ReLU activation, Dropout regularization)

Softmax output layer

Each based on the same 8192 paths from srand=9 (3wps each)			
	Scale=1	Scale=5	Scale=10
Class:	Frequency:	Frequency:	Frequency:
-1.00°_0.00mag	0.79	3.3	3.6
10.00°_2.50mag	0.81	2.54	3.87
10.00°_5.00mag	3.47	3.66	2.87
20.00°_2.50mag	0.6	2.43	0.63
20.00°_5.00mag	0.94	0.83	0.9
30.00°_2.50mag	0.33	0.83	1.59
30.00°_5.00mag	0.96	0.77	0.31
40.00°_2.50mag	0.12	0.7	0.4
40.00°_5.00mag	0.84	0.74	1.56
50.00°_2.50mag	0.16	0.87	0.55
50.00°_5.00mag	1.11	0.49	0.73
60.00°_2.50mag	0.09	1.68	1.93
60.00°_5.00mag	1.38	0.6	0.16
70.00°_2.50mag	0.11	0.57	0.43
70.00°_5.00mag	1.55	0.13	0.09
80.00°_2.50mag	0.15	0.55	0.94
80.00°_5.00mag	1.72	0.18	0.06
90.00°_2.50mag	0.15	0.31	0.16
90.00°_5.00mag	1.59	0.07	0.68
-90.00°_2.50mag	2.62	3.93	4.02
-90.00°_5.00mag	10.84	1.29	0.61
100.00°_2.50mag	0.17	0.45	0.92
100.00°_5.00mag	1.99	0.33	0.05
110.00°_2.50mag	0.16	0.62	0.44
110.00°_5.00mag	1.83	0.12	0.15
120.00°_2.50mag	0.11	0.37	0.26
120.00°_5.00mag	0.98	0.38	0.11
130.00°_2.50mag	0.2	0.79	0.89
130.00°_5.00mag	1.15	1.97	0.85
140.00°_2.50mag	0.23	1.17	1.05
140.00°_5.00mag	1	0.59	0.66
150.00°_2.50mag	0.1	0.66	0.43
150.00°_5.00mag	0.76	1.21	0.61
160.00°_2.50mag	0.21	0.76	1.29
160.00°_5.00mag	0.85	0.79	0.49
170.00°_2.50mag	0.24	0.99	2.78
170.00°_5.00mag	0.95	1.88	1.86
180.00°_2.50mag	0.49	2.32	1.65
180.00°_5.00mag	3.43	3.3	2.16

Fig. 6: Percentage distribution of selected v_f. Note that for 0 magnitude, there is no notion of direction, so -1.0 was selected symbolically.

G. Parameters

Our planner relies on several parameters for operation. The following parameters were used to perform comparison metrics and to generate paths in Figure 9.

TABLE I: Algorithm Parameters

v_{max}	a_{max}	j_{max}
10 m/s	10 m/s^2	60 m/s^3

V. RESULTS

The main goal of this paper was to get a superior motion primitive algorithm capable of real-time performance.

A. Time-Saving Pruning Algorithms

1) Greedy: The Greedy algorithm performs surprisingly well when at least one of these conditions are met (Figures 10, 11, 12):

1) waypoints must be relatively far apart from each other, at least on the order of 1m or 10m

2) the maximum velocity v_f must be much larger than the maximum acceleration a_f

3) the path segments should not make sharp turns

	Each based on the same 8192 paths from srand=9 (3wps each)			
		Scale=1	Scale=5	Scale=10
	Class:	% chosen:	% chosen:	% chosen:
45	-1.00°_0.00mag	11.01	12.45	29.99
46	10.00°_2.50mag	1	0.74	0.46
47	**10.00°_5.00mag**	0.52	0.01	0.04
48	20.00°_2.50mag	0.85	1.17	0.65
49	**20.00°_5.00mag**	0.57	0.01	0.02
50	30.00°_2.50mag	1.09	1.97	1.03
51	**30.00°_5.00mag**	0.92	0.04	0.01
52	40.00°_2.50mag	1.27	2.73	2.09
53	**40.00°_5.00mag**	1.06	0.06	0.02
54	50.00°_2.50mag	1.76	4.35	3.71
55	**50.00°_5.00mag**	1.67	0.18	0.02
56	60.00°_2.50mag	1.83	5.76	4.76
57	**60.00°_5.00mag**	3.22	0.48	0.01
58	70.00°_2.50mag	1.86	6.98	6.95
59	**70.00°_5.00mag**	6.58	1.51	0.02
60	80.00°_2.50mag	1.84	8.61	9.34
61	**80.00°_5.00mag**	10.12	2.49	0.06
62	90.00°_2.50mag	2.21	9.7	12.4
63	**90.00°_5.00mag**	12.66	3.94	0.02
64	100.00°_2.50mag	1.86	8.42	8.78
65	**100.00°_5.00mag**	10	2.4	0.05
66	110.00°_2.50mag	1.82	6.75	7.15
67	**110.00°_5.00mag**	6.59	1.4	0.05
68	120.00°_2.50mag	1.92	5.54	4.66
69	**120.00°_5.00mag**	3.13	0.57	0.01
70	130.00°_2.50mag	1.49	4.24	3.43
71	**130.00°_5.00mag**	1.64	0.11	0
72	140.00°_2.50mag	1.29	3.2	2.05
73	**140.00°_5.00mag**	1.25	0.09	0
74	150.00°_2.50mag	1.37	1.79	0.85
75	**150.00°_5.00mag**	0.98	0.09	0.02
76	160.00°_2.50mag	1.21	1.21	0.67
77	**160.00°_5.00mag**	0.67	0.02	0.05
78	170.00°_2.50mag	0.7	0.6	0.32
79	**170.00°_5.00mag**	0.57	0.02	0.02
80	180.00°_2.50mag	1.03	0.33	0.23
81	**180.00°_5.00mag**	0.46	0.01	0.02

Fig. 7: Percentage distribution of selected a_f. Note that for 0 magnitude, there is no notion of direction, so -1.0 was selected symbolically. Additionally, the a_f state options include -90.0 degrees due to the intuition of centripetal force.

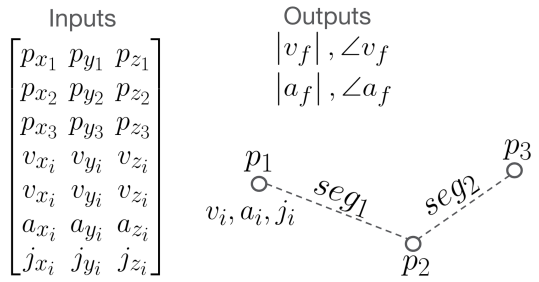


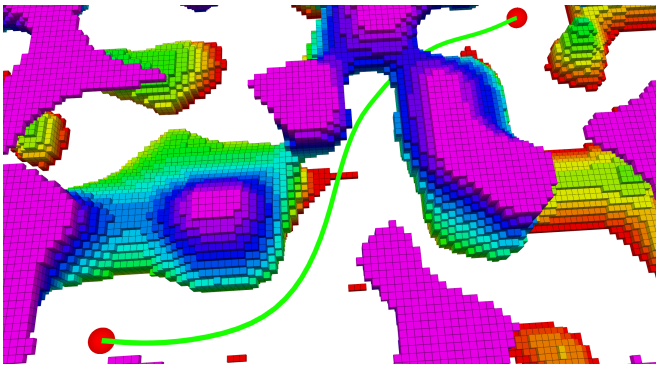
Fig. 8: Problem setup with inputs and outputs.

This makes intuitive sense. Consider a path that contains a sharp turn between segment s_1 and s_2. Then at s_1, the Greedy algorithm will myopically select the primitive endstate with maximum velocity and acceleration in the direction of s_1, when it would have been wiser to "look ahead" and slow down in preparation for s_2. If the $a_f >> v_f$, then this does not significantly affect execution time because at s_2, the drone dynamics can compensate for this myopic selection. Similarly, if the waypoints are farther apart from each other, then the drone has more distance during which to compensate for the myopic selection.

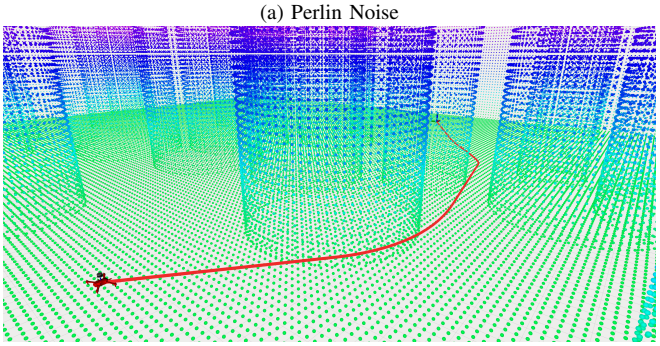
2) Greedy Lookahead: In general, these were the results:

For planning time: Greedy $<$ GLA $<$ Full

For execution time: Full $<$ GLA $<$ Greedy



(a) Perlin Noise



(b) Random Forest

Fig. 9: Final motion primitive path plans in (a) Perlin noise pointcloud and (b) a computer generated random forest.

		planning_time	execution_time
2	Trial #0(FULL)	42.9884	19.2801
3	Trial #0(GREEDY)	0.0252085	19.3531
4			
5	Trial #1(FULL)	39.6463	22.2535
6	Trial #1(GREEDY)	0.0288023	22.3982
7			
8	Trial #2(FULL)	37.8163	20.5758
9	Trial #2(GREEDY)	0.0261452	20.5787
10			
11	Trial #3(FULL)	38.9622	20.3971
12	Trial #3(GREEDY)	0.0271274	20.4849
13			
14	Trial #4(FULL)	38.2099	19.2039
15	Trial #4(GREEDY)	0.0265536	19.372
16			
17	Trial #5(FULL)	38.5165	14.6363
18	Trial #5(GREEDY)	0.0284088	14.6694

Fig. 10: Greedy pruning generally is only 0.1 seconds slower than the execution time for Full, while achieving a planning time speedup of 1500x. However, this is the case for randomly selected points in a 30x30x30m empty point cloud, in which waypoints are far apart.

Original random waypoints

Planner type	Planning time	Execution time
Greedy	0.0049	23.8774
Full	0.2446	23.8613

Scaled down by 10

Planner type	Planning time	Execution time
Greedy	0.0048	9.0647
Full	0.2311	8.2801

Fig. 11: In one trial, I generated a set of randomly selected points and then scaled them down by a factor of 10, keeping all other simulation parameters identical. The result was that the Greedy algorithm's execution time was much slower for the closer waypoints (0.78 seconds versus 0.01 seconds).

In some paths, GLA seemed yield fast Full-like execution times.

For planning time: Greedy $\approx$ GLA $<<$ Full

For execution time: Full $<$ GLA $<$ Greedy

However, in other paths, GLA seemed yield slow Greedy-

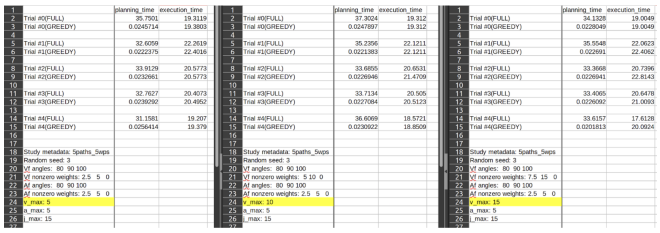


Fig. 12: As the ratio of v_{max}/a_{max} increases, all other simulation parameters remaining the same, Greedy performs worse compared to Full.

	A	O	P
1		planning_time	execution_time
2	Trial #0(FULL)	35.2549	7.19226
3	Trial #0(GREEDY)	0.0237805	7.46686
4	Trial #0(GREEDY_LOOKAHEAD)	0.726771	7.19785
5			
6	Trial #1(FULL)	34.5455	7.95828
7	Trial #1(GREEDY)	0.0261835	8.04059
8	Trial #1(GREEDY_LOOKAHEAD)	0.731207	7.95828
9			
10	Trial #2(FULL)	35.5158	7.62675
11	Trial #2(GREEDY)	0.0239042	7.88829
12	Trial #2(GREEDY_LOOKAHEAD)	0.716814	7.64515
13			
14	Trial #3(FULL)	39.277	7.45118
15	Trial #3(GREEDY)	0.0219467	8.57486
16	Trial #3(GREEDY_LOOKAHEAD)	0.711185	7.47746
17			
18	Trial #4(FULL)	30.9111	6.91654
19	Trial #4(GREEDY)	0.021316	8.5616
20	Trial #4(GREEDY_LOOKAHEAD)	0.769973	7.60213
21			
22			
23	Study metadata: 5paths_5wps		
24	Random seed: 3		
25	Scaling factor: 10		
26	Vf angles: 80 90 100		
27	Vf nonzero weights: 2.5 5 0		
28	Af angles: 80 90 100		
29	Af nonzero weights: 2.5 5 0		
30	v_max: 5		
31	a_max: 5		
32	j_max: 15		

Fig. 13: GLA consistently yielded near-identical execution times as Full (including in cases where Greedy's execution time was much slower than Full's), but consistently about 30x slower planning times than Greedy.

like execution times.

For planning time: Greedy $<<$ GLA $\approx$ Full

For execution time: Full $<$ GLA $<$ Greedy

The execution time for GLA was roughly a few orders of magnitude slower than Greedy but faster than Full, depending on the number of state samplings and other parameters. GLA outperformed Greedy in paths for which "looking ahead" by exactly one waypoint was worthwhile, i.e. the increase in planning time was worth the decrease in execution time (Figure 13). This naturally led to the question, how can we develop a threshold or heuristic for whether to "look ahead" and by how much? This seemed like a complex problem that would be appropriate for a machine learning algorithm.

3) Learning-enabled: After training on a small dataset with about 100 training examples per class, with a few hyperparameter sweeps (Figure 14), I attained a test accuracy of 8.027%. Although this number is far below 100%, since there are 37 different classification options, random guessing accuracy would be only 2.703%. Therefore, this basic neural network shows promise.

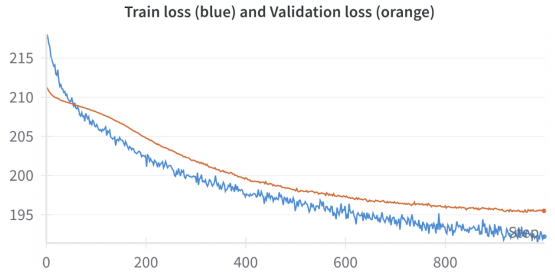


Fig. 14: Training and validation loss during training.

	OUR METHOD			SOA [1]		
waypoint #	planning time (ms)	execution time (s)	full time (s)	planning time (ms)	execution time (s)	full time (s)
4	3.25	5.864	5.867	42.92	5.862	5.905
5	4.13	7.75	7.754	56.14	7.431	7.487
6	4.13	7.973	7.977	61.95	7.102	7.164
7	3.94	11.56	11.564	80.77	11.042	11.123
8	4.26	12.162	12.166	65.98	11.178	11.244

Fig. 15: Our proposed planner yields 10x faster planning times and similar execution times to the state of the art presented by Wang et. al. [17]

VI. Simulation Results

See Figure 15 for preliminary results comparing our planner's timing performance with state-of-the-art.

VII. Conclusions

In this work, a new forward-looking framework for generating and searching over motion primitives was proposed. By strategically sampling motion primitives based on a reference path the planner is able to achieve computation speeds necessary for online planning.

For paths with far-apart waypoints, the greedy approach achieves near-identical execution time with 100x faster planning time. Depending on the state sampling choices, the greedy look-ahead approach can overcome the myopic property of the greedy approach while maintaining a 10x faster planning time than the full generation method. Initial results using a neural network appear promising but require more work in generating more training data, exploring RNN-based architectures, and tuning model hyper-parameters.

Future work includes hardware testing to validate the real-world applicability of our approach, the implementation of a post-processing smoothing step to further improve the execution time, and extension to a receding-horizon framework which enables navigation in unknown environments with online perception.

References

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[2] M. Hehn and R. D'Andrea, "Quadrocopter trajectory generation and control," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1485–1491, Jan. 2011.

[3] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *International Symposium of Robotic Research*, 2016, pp. 649–666.

[4] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, July 2017.

[5] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 42–49.

[6] J. Tordesillas and J. P. How, "MADER: Trajectory planner in multiagent and dynamic environments," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 463–476, Feb. 2022.

[7] J.Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "FASTER: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 922–938, Apr. 2022.

[8] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, Dec. 2015.

[9] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 5759–5765.

[10] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, no. 5, pp. 1048–1066, Nov. 1993.

[11] R. E. Allen and M. Pavone, "A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance," *Robotics and Autonomous Systems*, vol. 115, pp. 174–193, May 2019.

[12] S. M. LaValle and J. J. K. Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, May 2001.

[13] H. Ye, X. Zhou, Z. Wang, C. Xu, J. Chu, and F. Gao, "TGK-planner: An efficient topology guided kinodynamic planner for autonomous quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 494–501, Apr. 2021.

[14] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in SE(3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, July 2018.

[15] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, Oct. 2019.

[16] M. J. V. Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *International Journal of Robust and Nonlinear Control*, vol. 8, no. 11, pp. 995–1020, Dec. 1998.

[17] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," 2022.