AutoTune: Controller Tuning for High-Speed Flight

Antonio Loquercio*, Alessandro Saviolo*, and Davide Scaramuzza

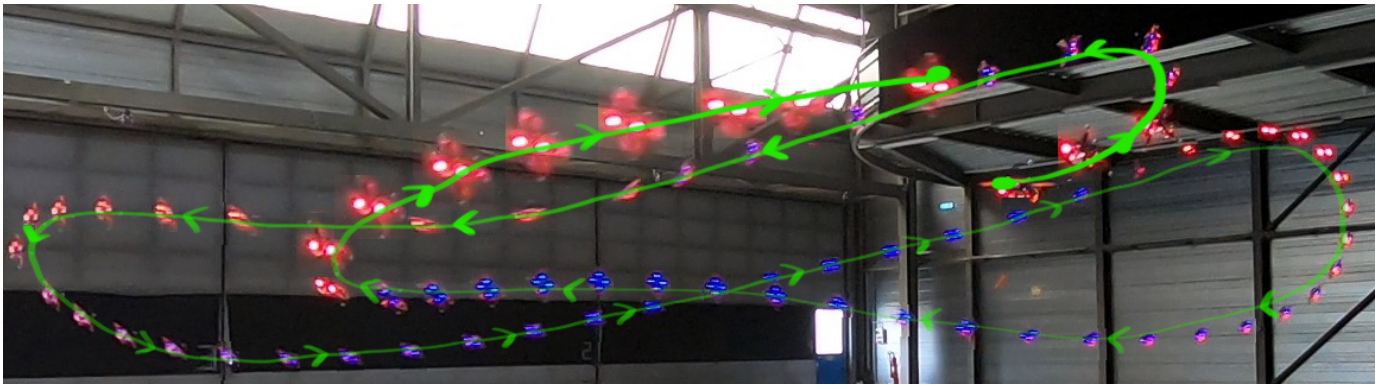


Fig. 1: Our quadrotor flies a time-optimal trajectory with speeds over $50\,\mathrm{km\,h^{-1}}$. We automatically find a controller configuration that can fly such a high-speed maneuver with a novel sampling-based method called AutoTune. To get a better sense of the speed achieved by the quadrotor, please watch the supplementary movie.

***Abstract*—Due to noisy actuation and external disturbances, tuning controllers for high-speed flight is very challenging. In this paper, we ask the following questions: How sensitive are controllers to tuning when tracking high-speed maneuvers? What algorithms can we use to automatically tune them? To answer the first question, we study the relationship between parameters and performance and find out that the faster the maneuver, the more sensitive a controller becomes to its parameters. To answer the second question, we review existing methods for controller tuning and discover that prior works often perform poorly on the task of high-speed flight. Therefore, we propose AutoTune, a sampling-based tuning algorithm specifically tailored to high-speed flight. In contrast to previous work, our algorithm does not assume any prior knowledge of the drone or its optimization function and can deal with the multi-modal characteristics of the parameters' optimization space. We thoroughly evaluate AutoTune both in simulation and in the physical world. In our experiments, we outperform existing tuning algorithms by up to 90% in trajectory completion. The resulting controllers are tested in the AirSim Game of Drones competition, where we outperform the winner by up to 25% in lap-time. Finally, we validate AutoTune in real-world flights in one of the world's largest motion-capture systems. In these experiments, we outperform human experts on the task of parameter tuning for trajectory tracking, achieving flight speeds over $50\,\mathrm{km\,h^{-1}}$**

Supplementary Material

Video and code are at https://youtu.be/m2q_y7C01So and https://github.com/uzh-rpg/mh_autotune.

I. Introduction

Flying high-speed trajectories with a quadrotor requires the platform's controller to be meticulously tuned [1], [2]. The complex relationship between parameters and performance, empirically shown in Fig. 2, is caused by unavoidable factors such as imperfect modeling and external disturbances. This work is motivated by the following questions: What are the characteristics of this optimization space? How can we automatically find controller parameters for high-speed flight?

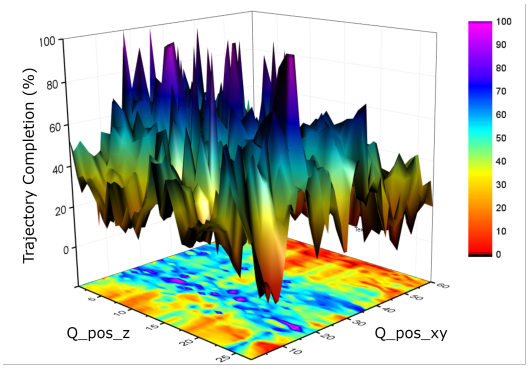


Fig. 2: Trajectory completion (%) as a function of two parameters of a model-predictive controller. The trajectory completion measures the percentage of trajectory successfully tracked by the controller. The high speed and high angular accelerations required by time-optimal trajectories make the controller extremely sensitive to its parameters.

Tuning controller parameters to fly high-speed maneuvers is difficult due to three main challenges: (i) the objective function (i.e. the relationship between controller parameters and performance) is highly non-convex (See Fig. 2); (ii) the tuning process only relies on noisy evaluations[1] of the objective function at adaptively chosen parameters, but not to the function itself or its gradients; (iii) different parts of the trajectory, e.g. a sharp turn or a straight-line acceleration, generally require different controller behaviors, hence dynamically changing parameters.

The traditional approach for automatic tuning and adaptive control, generally known as MIT rule [3], requires to express

Manuscript received: September, 9th, 2021; Revised December, 9th, 2021; Accepted January, 11th, 2022. This paper was recommended for publication by Editor Tamim Asfour upon evaluation of the Associate Editor and Reviewers' comments. The first two authors contributed equally. The work was done at the Robotics and Perception Group, University of Zurich, Switzerland (http://rpg.ifi.uzh.ch) and was supported by the National Centre of Competence in Research (NCCR) Robotics, through the Swiss National Science Foundation (SNSF), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 864042).

[1]Due to noise the same controller parameters can yield different performance on multiple runs.

the desired performance metric, *e.g.* the average tracking error over the entire maneuver, as a quadratic function of controller parameters, and then optimizes the controller with gradient-based optimization [4], [5], [6]. However, expressing the long-term performance on a high-speed maneuver with respect to the parameters of a receding horizon controller (*i.e.* the optimization function depicted in Fig. 2) is generally intractable. Indeed, it requires to know *a priori* the exact model of the quadrotor and the disturbances acting on it during flight, *e.g.* noisy actuation and aerodynamic effects. Instead of analytically computing it, another line of work proposes to iteratively estimate the optimization function, and use the estimate to find optimal parameters [7], [8], [9]. However, these methods make over-simplifying assumptions on the objective function, e.g. convexity or relative Gaussianity between observations. Such assumptions are generally not suited for controller tuning to high-speed flight, where the function is highly non-convex (c.f. Fig. 2). To remove any assumption, model-free methods propose to directly search for optimal parameters using sampling. Such methods are however built on heuristics not necessarily suited to high-speed flight and generally require thousands of iterations to converge [10].

In this paper, we propose a novel sampling-based algorithm specifically tailored to the problem of high-speed flight, rooted in statistical theory: AutoTune. Given an initial, low-performance controller, AutoTune optimizes its parameters to maximize a user-defined metric, e.g. track completion. In contrast to traditional adaptive control, *e.g.* the MIT rule, it does not require to analytically express the optimization function with respect to the controller parameters, nor assumptions about the optimization function. Similarly to model-free sampling-based methods, AutoTune does neither require prior knowledge of the platform model and external disturbances. However, to make sampling computationally tractable, our approach uses Metropolis-Hastings sampling (M-H) [11] and several strategies specifically tailored to the problem of high-speed flight. Specifically, motivated by the observation that different parts of a trajectory require different controller behaviors, we propose a strategy to break down a trajectory into components with different behaviors, e.g. sharp descent or planar acceleration. Despite controller parameters being different for each component, they are all optimized jointly to favor optimality over the entire trajectory. In addition, to speed up convergence, we train a regressor to predict good initialization parameters.

We perform an extensive evaluation in two simulators and in the physical world in a large tracking arena of $30 \times 30 \times 8\mathrm{m}$ volume. In these experiments, we find out that: (i) the faster a maneuver is, the more sensitive a controller becomes to its parameters, and (ii) the optimization function is multi-modal, *i.e.* multiple controller configurations lead to the desired performance. We empirically show that our approach can tune controllers up to 90 percentage points better than previous work in terms of trajectory completion. We then validate the controller parameters found by AutoTune in simulation on a physical platform. These parameters decrease the tracking error with respect to the ones tuned by a human expert, enabling the quadrotor to achieve speeds over $50\,\mathrm{km\,h^{-1}}$. Overall, our work makes the following contributions:

- We present a novel sampling-based method for tuning quadrotor controllers on the task of high-speed flight.
- We show that our method outperforms existing methods for automatic controller tuning and enables quadrotors to fly time-optimal trajectories both in simulation and in the physical world in one of the world's largest motion-capture systems.
- We provide interesting insights into the relationship between the parameters of a receding horizon controller and its flight performance on high-speed maneuvers.

II. Related work

The simplest option available to robotic researchers for controller tuning is to use domain knowledge, i.e. experience, to tune controllers' parameters. However, tuning by hand often translates in a tedious and time-consuming trial-and-error process, difficult even for the simplest maneuvers. Besides, human intuition often provides an inherent bias to the experiments, which results in sub-optimal performance and calls for a more principled parameter tuning approach.

In line with adaptive control, the classic approach for controller tuning analytically finds the relationship between a performance metric, *e.g.* tracking error or trajectory completion, and optimizes the parameters with gradient-based optimization [3], [4], [5], [6]. However, doing so requires to analytically derive the performance of a receding-horizon controller over a possibly long maneuver, which is intractable given the model errors, the noisy actuation, and other perturbations possibly acting on the platform during flight, *e.g.* aerodynamics effects. Approximating these effects numerically is possible for short maneuvers [12], [13], [14], but the more complex the maneuver or the system is, the more difficult the identification becomes, making these methods impractical for tuning controllers to fly time-optimal maneuvers. When a precise model of the platform is not available, methods like L1 adaptive control can estimate model errors online and account for them in a reactive fashion [15]. However, these methods trade off robustness with performance, that leads to sub-optimal behaviour during high-speed motion. In addition, they also have hyper-parameters to tune, so they could be complementary to our approach.

Motivated by this difficulty, another family of approaches estimates the relationship between the controller's performance and its parameters directly from data [16], [8], [7]. Through multiple experiments, both the estimate and the parameters are iteratively updated. However, doing so requires making additional assumptions on the shape of the function. The assumptions commonly used in the literature are: (i) relative normality between all observations according to some pre-defined kernel, as in Bayesian Optimization [8], [16], [9], and (ii) the function can be described by a parametric distribution, e.g. a Gaussian, as typical in inverse optimal control [7]. When the relationship between parameters and performance is very complex, as it is the case for time-optimal trajectories, these assumptions generally cause a poor fitting of the function, which results in sub-optimal tuning performance.

If demonstrations by a human expert are available, another option consists of using inverse reinforcement learning [17], [18], but this is generally not the case with time-optimal trajectories, which can be faster than the trajectories flown by the best human pilots [1].

To relax the assumptions required by the previous methods, another family of algorithms proposes to directly search for the optimal controller parameters by sampling. The main advantage of these algorithms is that they can deal with highly non-convex functions, like the one between parameters and performance in time-optimal trajectories. However, while this approach has encountered high successes with grounded robots [19], it's validity for agile systems like quadrotors is still unclear. Indeed, it was applied only on very simple tasks, *e.g.* hovering [20]. The major issue is how to define the parameters sampling procedure. We propose to do so with Metropolis-Hastings sampling [11]. Prior work successfully deployed this technique for collision-free planning in complex environments [21]. However, their approach performed a single large optimization, which is challenging and computationally expensive. In contrast, we design a trajectory segmentation procedure to divide the optimization in small sub-problems, which are faster and easier to solve.

III. Preliminaries and Overview

We define the task of high-speed flight through a series of waypoints as finding a policy minimizing the following cost:

$$\min_{\pi} J(\pi) = \mathbb{E}_{\rho(\pi)}\left[t(\pi)\right], \tag{1}$$

$$\text{subject to } \|\mathbf{u}[k]\| \leq \mathbf{u}_c \tag{2}$$

$$\mathbf{s}[k+1] = f(\mathbf{s}[k], \mathbf{u}[k]), \tag{3}$$

where $\mathbf{s}[k]$ is the quadrotor's state at time k, $\mathbf{u}[k]$ is the input, $\rho(\pi)$ is the distribution of possible trajectories induced by the controller π, and $t(\pi)$ is the time required to fly through all waypoints. The solution of Eq. (1) is a policy that minimizes the time to pass through all waypoints by respecting the platform dynamics (Eq. (3)) and actuation constraints (Eq. (2)). Given the series of waypoints and the platform's specifics, we approximate a solution to Eq. (1) with non-convex optimization [1], and then track the resulting trajectory $\boldsymbol{\tau}_r$ with a receding-horizon controller.

Analytical controllers, e.g. the Linear Quadratic Regulator (LQR), aim to solve the tracking problem directly. However, they disregard the platform dynamics and actuation constraints, resulting in poor controller performance when there is a mismatch between the model and the physical system [22]. In these conditions, the controller gains are tuned to maximize performance on the physical system, generally with iterative learning approaches [12]. Conversely, model predictive controllers (MPC) propose to solve a finite horizon version of the tracking problem in a receding horizon fashion:

$$\pi(\mathbf{x}[t_0]) = \min_{U} \left[\sum_{k=t_0}^{t_0+t_h} \mathbf{x}[k]^\top \boldsymbol{Q} \mathbf{x}[k] + \mathbf{u}[k]^\top \boldsymbol{R} \mathbf{u}[k] \right] \tag{4}$$

$$\text{subject to } \|\mathbf{u}[k]\| \leq \mathbf{u}_c$$

$$\mathbf{x}[k+1] = f(\mathbf{x}[k], \mathbf{u}[k]),$$

where $x[k] = \boldsymbol{\tau}_r[k] - \mathbf{s}[k]$ denotes the difference between the state of the platform and the corresponding reference at time k, $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the state and input cost matrices, and t_h is the horizon length, generally much smaller than the entire trajectory duration t. In contrast to the LQR, the platform constraints and dynamics are directly taken into account by the controller. This results in better behavior in case of mismatch between the model and the physical system [22]. This approach, however, requires to tune the controller parameters $\boldsymbol{Q}$, $\boldsymbol{R}$, and t_h to minimize a user-defined metric, *e.g.* the tracking error, over the entire trajectory [2]. Tuning these parameters is challenging since it is not possible to analytically find the relationship between them and the long-horizon cost, as possible for LQR controllers via the MIT rule [3].

Other approaches for high-speed trajectory tracking are non-linear geometric controllers and adaptive ones. However, a recent study has shown that in the absence of a precise model of the system, model-predictive control generally outperforms other approaches in the task of high-speed flight [23].

To improve the performance of MPC, we propose a strategy to split a trajectory into parts that require different controller behavior, hence different parameters. The parameters are optimized jointly to favor global optimization. Additionally, we initialize the search from a good guess of parameters to reduce sampling time. These parameters are predicted by a regressor trained on previously optimized tracks. Fig. 3 shows a summary of the proposed approach to tune controllers for high-speed flight. Despite being specific for MPC, we hypothesize that similar conclusions could be drawn when tuning different controllers. The next section presents each aspect of our method in detail.

IV. Method

A. Metropolis-Hastings Sampling

In statistics, the Metropolis-Hastings (M-H) algorithm [24] is used to obtain a sequence of random samples from a desired distribution $P(w)$ which can't be directly accessed. To generate the samples, the M-H algorithm requires a score function $d(w)$ which is proportional to the density $P(w)$. Samples are produced in an iterative fashion: the next sample w_{t+1} comes from a distribution $t(w_{t+1}|w_t)$, referred to as transition model, which only depends on the current sample w_t. As transition model $t(w_{t+1}|w_t)$ we select a Gaussian with constant variance $\sigma = 5$ centered on w_t. We keep this transition model fixed for all experiments. The next sample w_{t+1} is then accepted and used for the next iteration, or it is rejected, discarded, and the current sample w_t is re-used for the next iteration. Specifically, the sample is accepted with probability equal to

$$\alpha = min(1, \frac{d(w_{t+1})}{d(w_t)}) = min(1, \frac{P(w_{t+1})}{P(w_t)}). \tag{5}$$

Therefore, M-H always accepts a sample with a higher score. However, the move to a sample with a smaller score will sometimes be rejected, and the higher the drop in score $\frac{1}{\alpha}$,

[2]Parameters are equivalent up to scale. To account for this effect, we keep the cost on inputs $\boldsymbol{R}$ constant

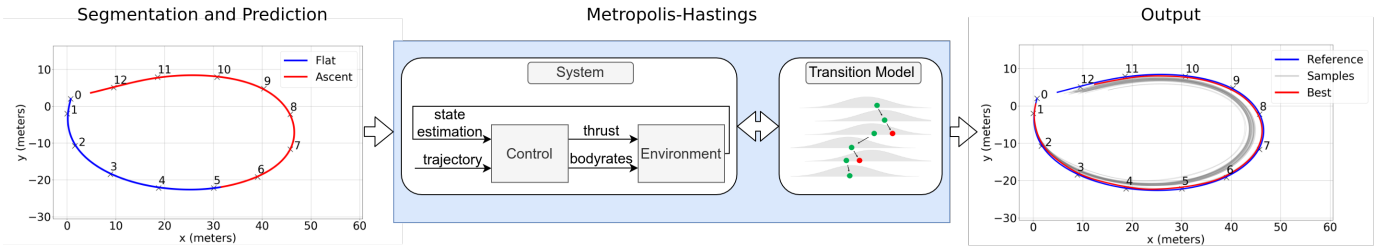


Fig. 3: We compute a minimum-time trajectory passing through all waypoints [1]. The trajectory is then segmented in parts that require different controller behaviors, and initial parameters for each segment are predicted with a regressor. The parameters are then jointly optimized with M-H sampling over multiple rollouts.

the smaller the probability of acceptance. Therefore, many samples come from the high-density regions of $P(w)$, while relatively few from the low-density regions. Intuitively, this is why the empirical sample distribution $\hat{P}(w)$ approximates the target distribution $P(w)$.

In this work, we use the M-H algorithm to find the parameters of a controller flying time-optimal trajectories. In this case, $P(w) = 1/Zd(w)$, where w are MPC parameters, Z is an unknown normalization factor, and $d(w)$ is the score function:

$$d(w) = \exp(-m(w)), \tag{6}$$

where $m(w)$ is a metric measuring the performance (*e.g.* time) the controller accumulates over the entire trajectory. According to $P(w)$, the points with maximum probability are the ones with higher score. However, in the task of controller tuning, we are not interested in approximating the distribution of controller parameters $\hat{P}(w)$, but to find, with as few samples as possible, parameters that enable tracking a trajectory accurately. We continue the sampling procedure up to when we find a solution satisfying some user-defined performance metrics, *e.g.* tracking error or trajectory completion. When found, we re-evaluate the solution for four times to account for the randomness of the simulation.

This setup makes the use of Metropolis-Hasting sampling equivalent to simulated annealing with constant temperature [25]. Despite varying-temperature simulated annealing providing the asymptotic guarantee of global optimality, it generally requires a significantly larger number of samples with respect to its constant-temperature counterpart [26] and a specifically designed heuristic to define the cooling function [25]. Therefore, since we are not interested in the global optimum but only in a controller configuration satisfying a user-defined performance metric, we keep the temperature to a constant value. In addition, due to the temperature-based sampling, simulated annealing does not provide the possibility to approximate the distribution of controller parameters (c.f. Fig. 2). Conversely, MH can approximate the distribution and used to study the characteristics of the optimization space.

B. Scoring Performance with Time

According to Eq. (1), we define the metric $m(w)$ in Eq. (6) to be the time t to pass all waypoints, *i.e.* $m(w) = J(\pi(w))$. However, it is not clear how to define this metric when the drone misses a waypoint or crashes before the end of the trajectory. To solve this problem, we propose to stop the experiment whenever the drone misses a waypoint or crashes.

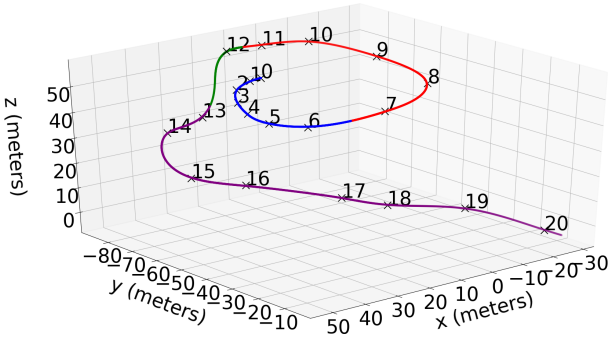


Fig. 4: Before starting the sampling, AutoTune segments a trajectory according to the gradient of z. The above maneuver was split into flat (blue), ascent (red), drop (green, steep descent), and descent (purple).

In this case, a penalty equal to the shortest path between the drone position and all further waypoints is added to the time. In such a way, it is possible to distinguish between parameters w that make the drone crash in the early stage of a trajectory and the ones that can complete the trajectory to the end.

C. Trajectory Segmentation

Complex high-speed trajectories require different controller behaviors along the track. For example, consider the reference trajectory illustrated in Fig. 4. The initial segment, depicted in blue, is approximately planar but has a large curvature in the x-y plane. In contrast, the drop segment, depicted in green in Fig. 4, has a large gradient in height, but little motion in the x-y plane. Clearly, these two segments need different controller behaviors to be successfully tracked. The blue planar segment requires the controller to be very precise on x-y tracking, but less controller authority is needed on the z-plane. Conversely, the large drop in altitude of the green segment necessitates very accurate tracking of the reference on the z plane, but less on the x-y one. Accounting for this behavior is important for time-optimal trajectories, where the drone is always close to its physical limits. Therefore, global parameters are likely to fail to track the entire trajectory, no matter how many samples are generated.

Motivated by this observation, we split the trajectory into multiple segments according to the height gradient of the reference. In each segment different parameters are assigned to the controller. Specifically, we assign each point to the class *flat*, *ascent*, or *descent* if the difference in height $g_z = z[k] - z[k+1]$ with its successor is $|g_z| < 1m, g_z \geq 1m, g_z \leq -1m$, respectively. This segmentation condition is kept fixed for all experiments and ablated in the appendix. The resulting segments are then clustered such that the minimum segment

Track	Max Vel $[\mathrm{m\,s^{-1}}]$	Random Search	Bayesian Optimization					PSO (1000)	CMA-ES (1000)	Gradient Boosting	AutoTune (Ours)
			Mat	LocalPer	RQ	SQ	Per				
Circle	20	100	100	100	100	100	100	100	100	100	**100**
Circle	34	65	80	65	60	60	30	100	100	65	**100**
Drop	20	40	100	50	50	50	40	100	100	50	**100**
Flip	16	80	80	80	80	80	80	100	100	80	**100**
Spiral	53	0	10	0	0	10	0	10	10	10	**100**
Qualifier	20	50	75	60	60	60	30	75	100	65	**100**
Final	22	10	40	40	30	30	25	40	100	40	**100**

TABLE I: Comparison of AutoTune with the baselines. All approaches have a maximum budget of 200 samples, except PSO and CMA-ES, whose budget is 1000 samples. While all baselines perform well on easy maneuvers, their performance drops when the speed and angular acceleration required by the trajectory increases. Conversely, AutoTune can always find parameters to fly the entire trajectory.

duration is 2 seconds. Eventually, all the descent and ascent segments with a slope higher than 45° are recursively split into two equal parts, where the first is assigned to the class *steep* and the second remains assigned to the original class. Fig. 4 shows the result of the segmentation algorithm in one of our testing maneuvers. To account for the strong correlations between segments and keep the optimization global over the trajectory, the controller parameters associated with each segment are updated jointly. More details about the joint optimization process and other segmentation examples are available in the appendix.

D. Sampler Initialization

The Metropolis-Hastings algorithm requires an initial parameter configuration w_0 to initialize the sampling. Instead of using a random initialization, we propose to use an informed guess for w_0. Specifically, we use a Gradient Boosting regressor [27] with default parameters to predict initial controller parameters for each trajectory segment. The training data for this regressor are controller parameters found to be optimal on 5 training trajectories different in layout from the testing ones. A different regressor is trained for each type of trajectory segment, i.e. flat, ascent, descent, and steep. Five features including information about the reference trajectory are used for prediction: the number of points in the segment, the slope of the line connecting the first and last point of the segment, as well as their height difference, and the mean velocity and acceleration. These features have been selected with a cross-validation procedure.

Overall, the idea of predicting an initial guess w_0 with a regressor trained on previously seen trajectory experimentally shows to drastically reduce the number of samples (up to 88%) to find controller parameters for flying a time-optimal trajectory. More details about the training data, the training procedure, and an ablation study of the features are available in the supplementary material.

V. Experiments

We design our evaluation procedure to address the following questions: Can AutoTune find controller parameters to fly high-speed trajectories? What are the characteristics of the optimization space of controller parameters for the task of high-speed flight? Do the tuned controllers improve performance on a physical platform? Furthermore, we validate our design choices with ablation studies. We encourage the reader to watch the supplementary video for qualitative results.

A. Experimental Setup

We use for our experiments two simulators known for their physical and visual realism: Microsoft AirSim [28], and Flightmare [29]. Our sampling procedure is strongly favored by the high speed at which they can simulate physics (up to 10K times real-time). We test AutoTune on six trajectories selected to evaluate controller performance under strong accelerations and high-speed on all axes.

For comparison, we use four baselines for controller tuning. A naive one (*Random Sampling*) which randomly samples parameters independently and uniformly on all axis with a variance of 5. Moreover, we compare to the strong baselines of *Bayesian Optimization* [9] with multiple choices of the Gaussian kernel; *Particle Swarm Optimization* [30] (PSO), with 10 particles and using $0.5, 1, 2$ as, respectively, inertia weight, cognitive constant, and social constant; and *Covariance Matrix Adaptation* Evolution Strategy (CMA-ES) [31]. . All baselines start tuning from the same point as ours: the trajectory is divided into parts and the regressor predicts initial parameters. Note that the traditional tuning methods based on gradient-based optimization [14], [13], [3] are impractical for this task, given the difficulty to explicitly find the relationship between the parameters of our receding-horizon MPC controller and the tracking performance over the entire maneuver.

We define the metric of *Trajectory Completion* (TC) to compare the different approaches. Formally, this metric is defined as:

$$TC = \frac{\sum_{i \in waypoints} \mathbb{1}[i]}{\sum_{i \in waypoints} 1}, \tag{7}$$

where the indicator function for waypoint i is

$$\mathbb{1}[i] = \begin{cases} 1 & \text{if drone at distance} < d \text{ from } i \text{ at } t_r(i), \\ 0 & \text{otherwise,} \end{cases}$$

and $t_r(i)$ is the time when the reference τ_r predicts the quadrotor to pass the waypoint i. Whenever a waypoint is missed by more than $d = 1.3\,\mathrm{m}$ (gate radius for our drone racing experiments) or the drone crashes, the experiment is stopped and the metric calculated. We use this metric in our experiment since it is easy to interpret and can be compared across different experiments.

B. Tracking Minimum-Time Trajectories

We first evaluate the performance of AutoTune compared to the baselines. The results are summarized in Table I. AutoTune

Team	Qualification Round			Final Round		
	Lap Time [s]	Max Vel [m/s]	Avg Vel [m/s]	Lap Time [s]	Max Vel [m/s]	Avg Vel [m/s]
QuetzalC++	42.01	17.20	8.13	53.52	28.19	9.82
Chuchichaschtli	37.58	18.96	9.11	53.49	18.70	9.54
Dedale	30.11	16.49	11.33	39.78	20.02	12.88
AutoTune	**24.05**	**21.68**	**14.06**	**38.09**	**19.83**	**14.01**

TABLE II: Game of Drones 2019 leaderboards. AutoTune outperforms the winner of the competition in both qualification and final round.

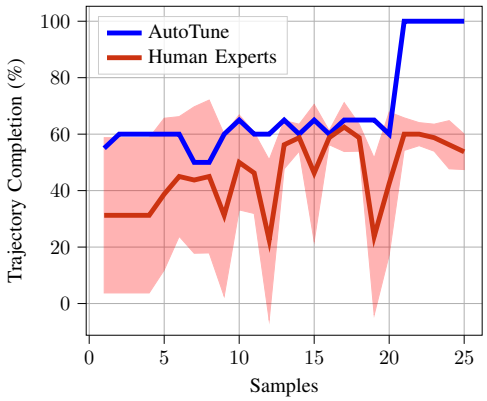


Fig. 5: Comparison between AutoTune and a set of human experts on the Qualifier track (Fig. 4) in simulation. Despite having access to more information than AutoTune, no human was able to find suitable parameters.

is consistently the best across all maneuvers. For easy maneuvers, e.g. the slow circle or the flip, almost any controller configurations can complete the track, and all methods can find a viable solution. However, for more difficult maneuvers, the gap between our approach and the baselines widens, reaching up to 90% in the Spiral track. On the most difficult tracks, the Bayesian baseline has difficulties fitting the optimization function, while PSO generally gets stuck into a local minimum after few samples. Given enough samples, CMA-ES achieves very good results, with the exception of the Spiral trajectory, where it consistently fails to pass through the second gate.

Fig. 5 shows the comparison between the tuning performance of AutoTune and four human experts over time. All humans are graduate students expert in quadrotor research and not authors of this paper. They have additional sources of information with respect to our algorithm, i.e. the 3D trajectory flown by the drone. The results show that tuning parameters by hand is extremely challenging for humans, given the non-intuitive relationship between parameters and performance.

We additionally compare our approach to the top three methods in the 2019 AirSim Game of Drones competition [28]. We compare the methods both on the qualification and final round of the competition. The results of this experiment are summarized in Table II. On the qualifier track flying AutoTune achieves a lap-time of $24.05\,\mathrm{s}$, while the winner only reaches the goal in $30.11\,\mathrm{s}$, with a lap-time 25% longer than ours. Also in the final round, AutoTune outperforms the winner of the competition with a $1.7\,\mathrm{s}$ margin, completing the track in approximately 5% less time. Interestingly, our approach converges to a policy with a maximum speed not necessarily higher than others. However, we achieve an average velocity higher than baselines, and therefore a faster lap-time. This experiment shows that tuning controller parameters with an automated procedure allow quadrotors to fly faster trajectories.

C. Application in the Real World

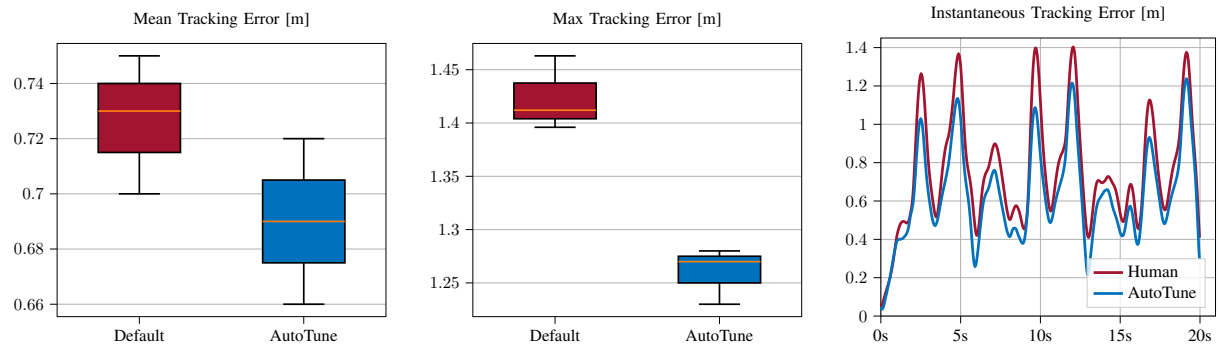
AutoTune can be used to tune the controller of a physical platform. To do so, we compute a minimum-time trajectory double Split-S trajectory of 21 waypoints [1]. This trajectory is used to tune the controller in the Flightmare simulator. The resulting controller is then evaluated on a physical platform in a tracking arena of volume $30 \times 30 \times 8\mathrm{m}$, where the quadrotor achieves speeds over $50\,\mathrm{km\,h^{-1}}$. Figure 6 shows the results of this experiment. AutoTune improves average tracking error by 6% and decreases the maximum displacement from the reference by 12%. In addition, our controller parameters give more consistent performance over multiple runs than the baseline.

D. Robustness to Changes in Mass, Velocity, and Track Layout

In this section, we study the robustness of the parameters found by AutoTune to changes in drone's mass, flight speed, and track layout. All experiments are made on the Qualifier track. Figure 7 show the results of these experiments. The parameters are overall robust to changes in the quadrotor mass and can complete the task even for very different settings. When changing the maximum speed achieved during flight (Fig. 7-b), we observe that a faster trajectory requires more precise tuning. Finally, we test whether parameters generalize between different maneuvers. To favor generalization, we copy parameters for each segment independently. The results (Fig. 7-c) show that, when the maneuver used for tuning is different from the testing one, the performance generally drops. One possible solution to this problem would be to do automatic fine-grained segmentation of trajectories. Indeed, we hypothesize that smaller trajectory segments, despite being more difficult to optimize, would transfer better between different layouts.

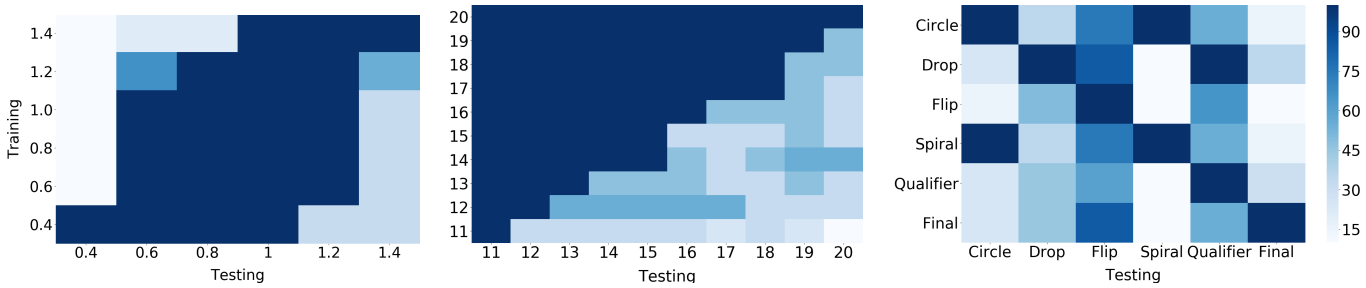
E. Robustness to Initial Conditions

In this section, we study the evolution of the controller parameters during optimization for different initialization conditions. Specifically, we start the sampling procedure from 3 random initializations and tune the controller with our approach in the Flightmare simulator on the *Qualifier* track (Fig. 4). Figure 8 shows the results of this experiment. AutoTune finds parameters to reach 100% of trajectory completion for all initial conditions. However, while some require as little as 50 samples, others require up to 1K to converge. Interestingly, the approach follows different paths in the optimization space for every initialization. In addition, the sampling converges



(a) (b) (c)

Fig. 6: Results in the real world. After tuning the parameters in simulation, we evaluate the best configuration found by AutoTune on a physical platform. We compare the performance with the parameters tuned by a human. We perform three runs for each parameter set. We also report the error as a function of time (c) for the best run of each approach.



(a) Quadrotor Mass [kg] (b) Max velocity [$\mathrm{m\,s^{-1}}$] (c) Track Layout

Fig. 7: Performance analysis when parameters used for tuning (Training) is different to the one used during execution (Testing). Overall, the parameters are robust to imperfect identification of the mass, and the faster the maneuver, the more sensitive the controller is to the parameters. However, the controllers require to be tuned specifically to the maneuver. When the training and testing maneuvers are different, performance generally drops.

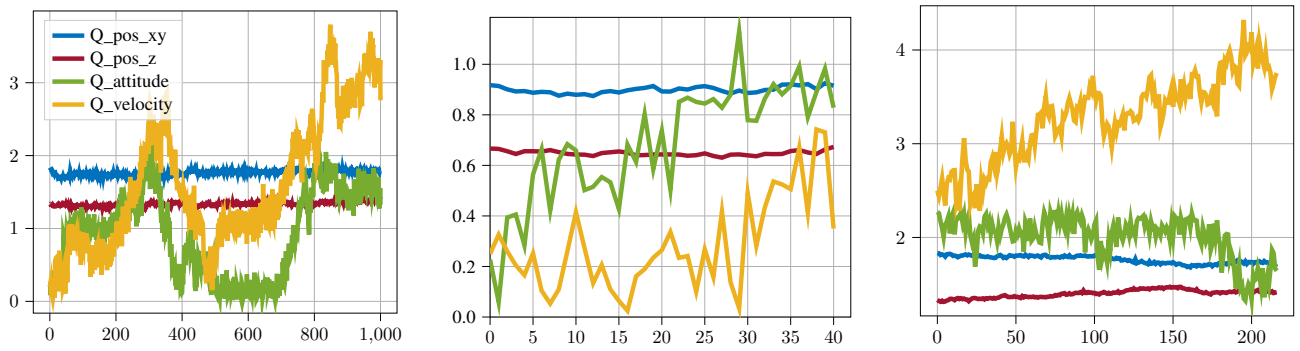


Fig. 8: Robustness to different initial conditions. The parameters' values are divided by their maximum value achieved over all runs. AutoTune converges to a configuration with 100% trajectory completion for all initial conditions. Initialization strongly affects the converge speed. All runs converge to a different optimum, demonstrating the multi-modal characteristic of the optimization function.

to a different local optimum for each initial condition. This behavior empirically shows that the relationship between parameters and performance at high-speed is *multi-modal*, *i.e.* different controller parameters have the same performance. These characteristics of the optimization function represent a challenge for gradient-based and Bayesian methods [6], [8], [8], which tend to converge to the mean between different optima. Conversely, since Metropolis-Hastings sampling can approximate any probability distribution under relatively mild assumptions, our approach does not suffer from the multi-modal nature of the optimization function.

	Trajectory Completion [%]	samples
AutoTune	**100**	**21**
– Regressor	100	172
– Segmentation	65	300

TABLE III: Ablation Study of the system's component on the Qualifier trajectory.

F. Ablation Studies

AutoTune is based on several components to reduce the sample complexity of Metropolis-Hastings sampling. We now validate our design with an ablation study. In particular, we ablate the following components: (i) the use of a regressor for

predicting an initial controller to initialize sampling, (ii) the segmentation of the trajectory in different parts. The results in Table III show that all components are important, but some have a larger impact than others. The initial guess produced by the regressor drastically reduces the number of samples to convergence, making the sampler find a solution in 88% less time. However, the most important contribution comes from the trajectory segmentation. Without this component, the sampler cannot find parameters to complete more than 65% of the trajectory in less than 300 samples. This is because global parameters do not allow the controller to dynamically adapt to different parts of the trajectory.

VI. Discussion and Conclusions

This paper shows the importance of an automated tuning procedure to fly high-speed maneuvers. While the effect of tuning is less prominent at low speeds, it acquires a fundamental role when the quadrotor flies at the limits of handling. In such cases, the relation between the parameters and flight performance (measured, for example, in terms of trajectory completion or tracking error) is non-convex, not injective, and multi-modal. In this paper, we propose a sampling-based approach specifically tailored to the task of high-speed flight.

One limitation of the proposed approach is that it does not consider closed-loop stability during optimization. While prior work proposed a series of techniques to guarantee stability during tuning [8], [9], [6], such techniques either require a very accurate model of the platform or very conservative parameters exploration strategies. These makes them suited for tasks like hovering or low-speed flight but not to high-speed flight, where model mismatch makes the parameter's optimization landscape very complex. Similar to previous work on agile flight [32], we have addressed this problem by tuning the controller exclusively in simulation and directly using the tuned controller on a physical platform. However, such a strategy strongly depends on the quality of the simulation environment. Therefore, combining existing techniques for safe tuning with our approach, to either tune from scratch or only finetune the controller on the physical platform, is a very exciting venue for future work.

References

[1] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, 2021.

[2] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for near-time-optimal quadrotor flight," *arXiv preprint arXiv:2108.13205*, 2021.

[3] K. J. Åström, "Theory and applications of adaptive control - A survey," *Autom.*, vol. 19, no. 5, 1983.

[4] M. J. Grimble, "Implicit and explicit lqg self-tuning controllers," *Automatica*, vol. 20, no. 5, pp. 661–669, 1984.

[5] K. Åström, T. Hägglund, C. Hang, and W. Ho, "Automatic tuning and adaptation for pid controllers: a survey," *Control Engineering Practice*, vol. 1, no. 4, pp. 699–714, 1993.

[6] M. A. M. Basri, A. R. Husain, and K. A. Danapalasingam, "Intelligent adaptive backstepping control for mimo uncertain non-linear quadrotor helicopter systems," *Transactions of the Institute of Measurement and Control*, vol. 37, no. 3, pp. 345–361, 2015.

[7] M. Menner and M. N. Zeilinger, "Maximum likelihood methods for inverse learning of optimal controllers," *IFAC World Congress*, 2020.

[8] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[9] S. S. M. Alonso, P. Hennig, J. Bohg and S. Trimpe, "Automatic lqr tuning based on gaussian process global optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, May 2016, pp. 270–277.

[10] Y. Davidor, "Genetic algorithms and robotics: a heuristic strategy for optimization," *World Scientific*, 1991.

[11] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[12] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 1642–1648.

[13] A. Julkananusart and I. Nilkhamhang, "Quadrotor tuning for attitude control based on double-loop pid controller using fictitious reference iterative tuning (frit)," in *IEEE Industrial Electronics Society*, 2015, pp. 4865–4870.

[14] L. Cedro and K. Wieczorkowski, "Optimizing pid controller gains to model the performance of a quadcopter," *Transportation Research Procedia*, vol. 40, pp. 156–169, 2019.

[15] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *IEEE Robotics and Automation Letters*, 2021.

[16] S. Trimpe, A. Millane, S. Doessegger, and R. D'Andrea, "A self-tuning lqr approach demonstrated on an inverted pendulum," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 281–11 287, 2014.

[17] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *International Joint Conference on Artifical Intelligence (IJCAI)*. Morgan Kaufmann Publishers Inc., 2007, p. 2586–2591.

[18] X. Xiao, B. Liu, G. Warnell, J. Fink, and P. Stone, "Appld: Adaptive planner parameter learning from demonstration," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4541–4547, 2020.

[19] Z. Wang, X. Xiao, G. Warnell, and P. Stone, "Apple: Adaptive planner parameter learning from evaluative feedback," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, p. 7744–7749, Oct 2021.

[20] K. Khuwaja, N.-u.-Z. Lighari, I. C. Tarca, and R. C. Tarca, "Pid controller tuning optimization with genetic algorithms for a quadcopter," *Recent Innovations in Mechatronics*, vol. 5, no. 1., pp. 1–7., 2018.

[21] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, pp. 1–8, 2021.

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

[23] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight," *arXiv*, 2021.

[24] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, 1983.

[26] M. Fielding, "Simulated annealing with an optimal fixed temperature," *SIAM J. Optim.*, vol. 11, no. 2, 2000.

[27] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[28] R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, and A. Kapoor, "Airsim drone racing lab," *NeurIPS 2019 Competition Track*, 2020.

[29] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," *Conference on Robot Learning (CORL)*, 2020.

[30] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[31] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[32] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," *Robotics: Science and Systems (RSS)*, 2020.

[33] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-aware model predictive control for quadrotors," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

[34] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection," *Knowledge and Information Systems*, 2017.

[35] M. Diehl, H. J. Ferreau, and N. Haverbeke, *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*. Springer Berlin Heidelberg, 2009, pp. 391–417.

VII. SUPPLEMENTARY

A. Tracks

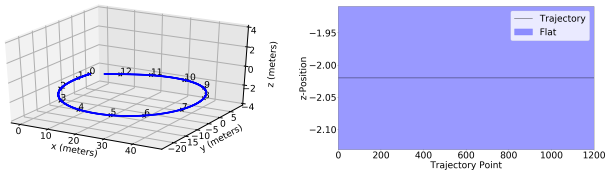


Fig. 9: Circle track.

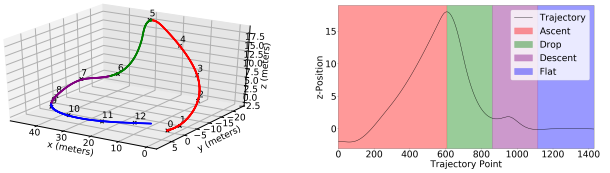


Fig. 10: Drop track.

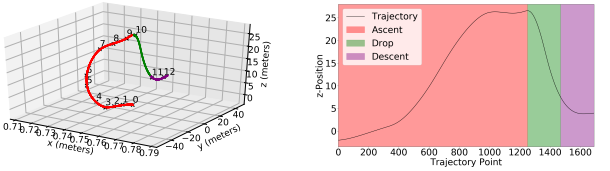


Fig. 11: Flip track.

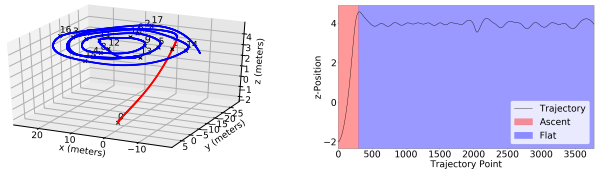


Fig. 12: Spiral track.

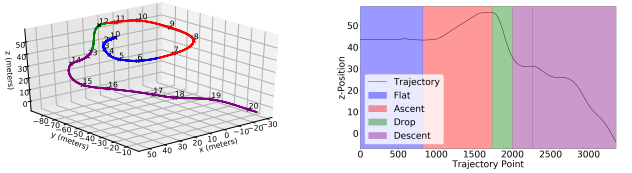


Fig. 13: Qualifier track.

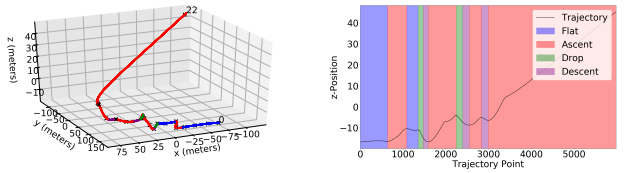


Fig. 14: Final track.

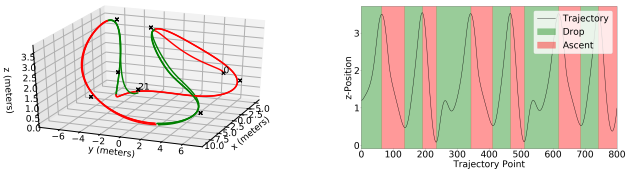


Fig. 15: SplitS track.

Specifically, we select the following maneuvers:

- Circle: 2D maneuver with a radius of $r = 16\,\mathrm{m}$. Consists of 12 waypoints at an average of $9\,\mathrm{m}$ distance.
- Drop: Ascent of $21\,\mathrm{m}$ in z over $50\,\mathrm{m}$ length followed by a steep descent of $16\,\mathrm{m}$ in height. Ends with a semi-circle with radius $r = 16\,\mathrm{m}$. Consists of 12 waypoints at an average of $10\,\mathrm{m}$ distance.
- Flip: Ascent of $25\,\mathrm{m}$ in z over $117\,\mathrm{m}$ length, with an half-loop in z of $14\,\mathrm{m}$ radius. Ends with a steep descent of $r = 20\,\mathrm{m}$ over $28\,\mathrm{m}$ length. Consists of 12 waypoints at an average of $13\,\mathrm{m}$ distance.
- Spiral: Ascent of $6\,\mathrm{m}$ followed by a spiral made of three planar circles of 30, 22, and 16 meters in the clockwise direction followed by a half a circle with radius $r = 6\,\mathrm{m}$. Then the drone follows two planar circles of 30 and 38 meters in a counter-clockwise direction. Consists of 17 waypoints at an average of $13\,\mathrm{m}$ distance.
- Qualifier: Track used for the qualification round of the 2019 AirSim Game of Drones competition. Made of an ascending circle of $160\,\mathrm{m}$, a $23\,\mathrm{m}$ drop in height over $25\,\mathrm{m}$ length and a long straight descent of $148\,\mathrm{m}$ length. Consists of 20 waypoints at an average of $16\,\mathrm{m}$ distance.
- Final: Track used for the final round of the 2019 AirSim Game of Drones competition. Made of a $209\,\mathrm{m}$ long segment with a hairpin and loop, followed by a $40\,\mathrm{m}$ ascent over approximately $296\,\mathrm{m}$. Consists of 22 waypoints at an average of $22\,\mathrm{m}$ distance.

Eventually, in the real world we fly an aggressive trajectory, that we call SplitS, composed of 21 waypoints with abrupt changes in the flight direction and speeds in the range $8 - 15\mathrm{m\,s^{-1}}$. The SplitS trajectory consists of a sequence of ascents and drops of $3\,\mathrm{m}$ in height repeated three times. The trajectories of these maneuvers are illustrated in Section VII-A. All maneuvers start and end in the hover condition.

B. Training Data and Feature Selection

The training data consists of 14 time-optimal trajectories, each annotated with a set of parameters that allows the controller to successfully track the trajectory. In total, the training trajectories are divided into 73 segments: 13 flat, 22 ascent, 23 descent, and 15 steep. For each segment type, we train a Gradient Boosting regressor on the relative data with default parameters. We extract five features from each segment, namely the number of points in the segment, the slope of the line connecting the first and last point of the segment, as well as their difference in height, and the mean velocity and acceleration. These features were selected with a cross-validation procedure over a wider set of features. Specifically, during the selection process we considered the following features: the number of points in the segment, the slope of the line connecting the first and last point of the segment, as well as their difference in height, velocity and acceleration, and the minimum, the mean and the maximum velocity and acceleration. We evaluated each feature combination by predicting the controller initial parameters on the qualifier track and then by computing the root-mean-square error (RMSE) of the predicted parameters with respect to the optimal parameters in the training data. Finally, we selected the feature combination that minimized the RMSE.

C. Implementation Choices

We use the MPC formulation of Falanga [33], without perception constraints, and the time-optimal planner described by Foehn [1]. We use the AirSim SimpleFlight quadrotor

configuration. The quadrotor weights 1.0 kg and can generate thrust to weight ratio up to 4.179. We choose the MPC control frequency to be 200 Hz with horizon step 0.05. Rather than optimize over the entire optimization landscape, we fix some parameters of the controller to obtain a reduced, but representative, subspace. In particular, we tune the MPC state costs (Q_pos_xy, Q_pos_z, Q_attitude, Q_velocity) and the horizon length, while we fix the MPC input costs (R_thrust, R_pitchroll, R_yaw) to 1.0, which we experimentally saw to be not very important to performance. Moreover, we bound the inputs to match the limits of the simulator, by fixing the maximum pitch-roll to 10.0 rad/s, the maximum yaw to 3.0 rad/s and the maximum thrust to 20.0 N. The same drone specifics are also used by the time-optimal planner.

We use a Gaussian distribution as proposal function for Metropolis-Hastings algorithm. The distribution is centered on the last set of parameters accepted and has variance 5, which we empirically found to be effective for our task. At each iteration, the M-H algorithm proposes a new set of parameters by sampling them from the Gaussian distribution. When updating the parameters of a segment, the variance of the preceding segments is reduced by 25%. This maintains global optimality but makes the controller close to the previous solution in the preceding segments. We use $e^{-2\sqrt{t}}$, where t is the lap time with added penalty for missed waypoints, as cost function for the M-H algorithm.

D. Choosing Trajectory Segmentation Conditions

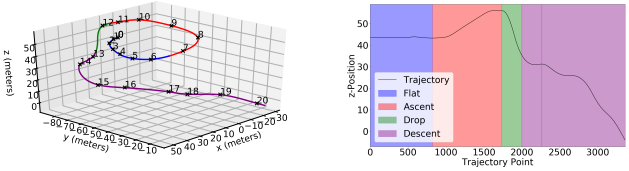
Trajectory segmentation is fundamental for successfully tracking high-speed maneuvers. We split the trajectory into multiple segments according to the height gradient of the reference. Specifically, we assign each point to the class *flat*, *ascent*, or *descent* if the difference in height $g_z = z[k] - z[k+1]$ with its successor is $|g_z| < 1m$. The resulting segments are then clustered such that the minimum segment duration is 2 seconds. Eventually, all the descent and ascent segments with a slope higher than $45°$ are recursively split into two equal parts, where the first is assigned to the class *steep* and the second remains assigned to the original class.

Since the segmentation conditions are crucial to the correct functioning of the proposed algorithm, we study its effects on the tracking performance. Table IV reports the trajectory completion performance of the proposed algorithm on the Qualifier track. Each entry of the table is illustrated in Fig. 16. The reported tracking performance is the best trajectory completion found by the proposed algorithm after 100 iterations.

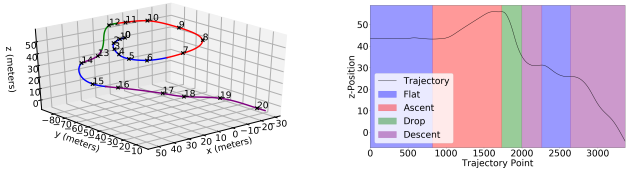
The results demonstrate the importance of choosing the right segmentation conditions. By employing high thresholds on height difference and/or minimum duration, the trajectory segmentation method generates only one global segment, thus not allowing different controller behaviors along the track. On the other hand, if the chosen thresholds are too little, then the proposed algorithm cannot find a suitable solution in the limited number of iterations. Therefore, in future work we aim to build a parameter-free automatic segmentation procedure with change-point detection techniques generally used for time series [34].

Height Difference [m]	Min Duration [s]	Trajectory Completion [%]
1	2	100
10	2	100
20	2	65
30	2	65
1	1	60
1	5	65

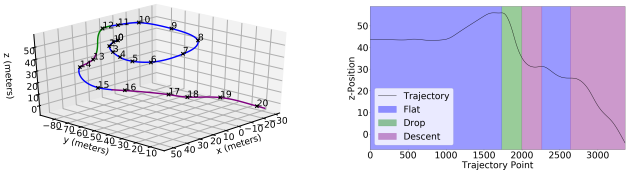
TABLE IV: Effects of the trajectory segmentation conditions on the tracking performance. The track considered is Qualifier and the metric is the trajectory completion. We assign each point to the class *flat*, *ascent*, or *descent* if the difference in height $g_z = z[k] - z[k+1]$ with its successor is $|g_z|$ <*Height Difference*. The resulting segments are then clustered such that the minimum segment duration is *Min Duration* seconds.



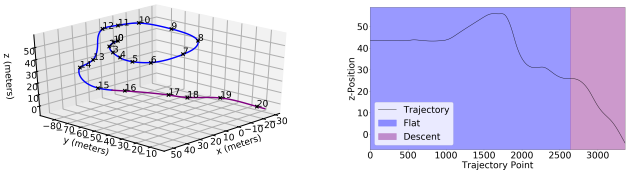
Height Difference $1m$, Min Duration $2s$.



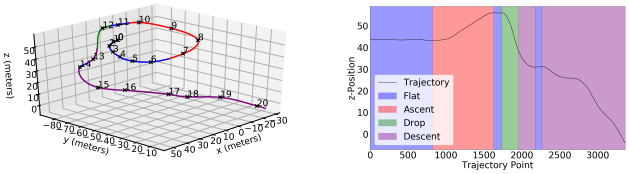
Height Difference $10m$, Min Duration $2s$.



Height Difference $20m$, Min Duration $2s$.



Height Difference $30m$, Min Duration $2s$.



Height Difference $1m$, Min Duration $1s$.

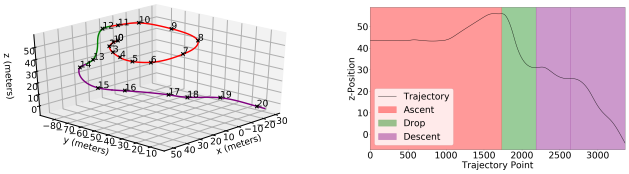


Fig. 16: Height Difference $1m$, Min Duration $5s$.

E. Controller Stability

To increase the stability of the controller between segments, we perform nonlinear MPC optimization with a real-time iteration (RTI) scheme [35]. The fact that only a single optimization iteration is done for each timestamp allows a smooth transition of the commanded actions while switching segments. A qualitative analysis is illustrated by Figure 17. The controller's outputs remain smooth over time during switching for any pair of segments.

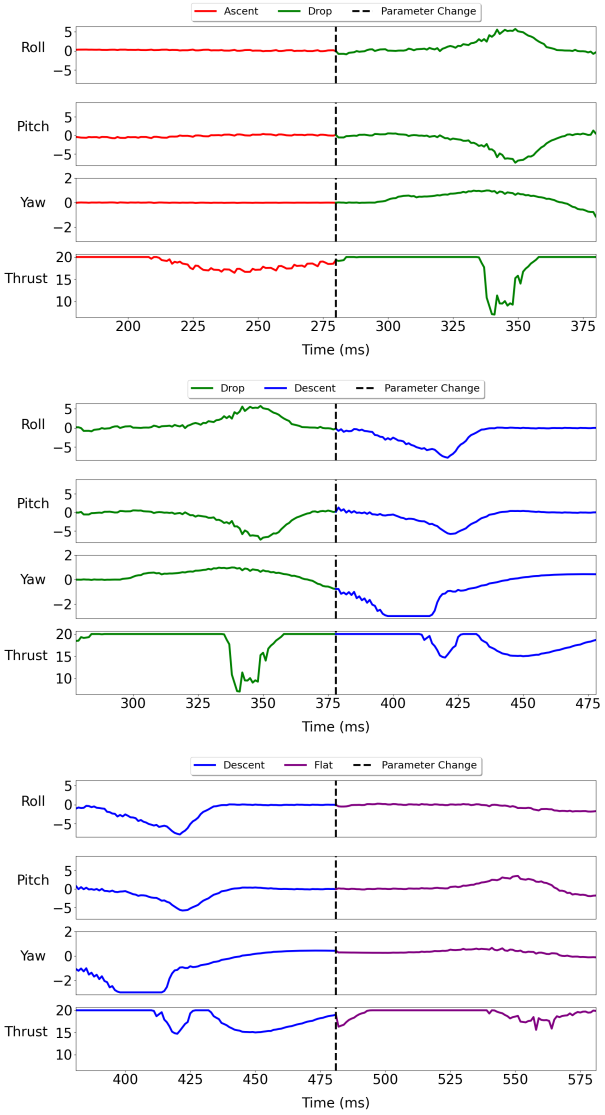


Fig. 17: Effects of parameter change on the stability of the controller on the Drop track. The controller's outputs remain smooth over time during switching for any pair of segments.