

AutoTune: Controller Tuning for High-Speed Flight

Antonio Loquercio*, Alessandro Saviolo*, and Davide Scaramuzza

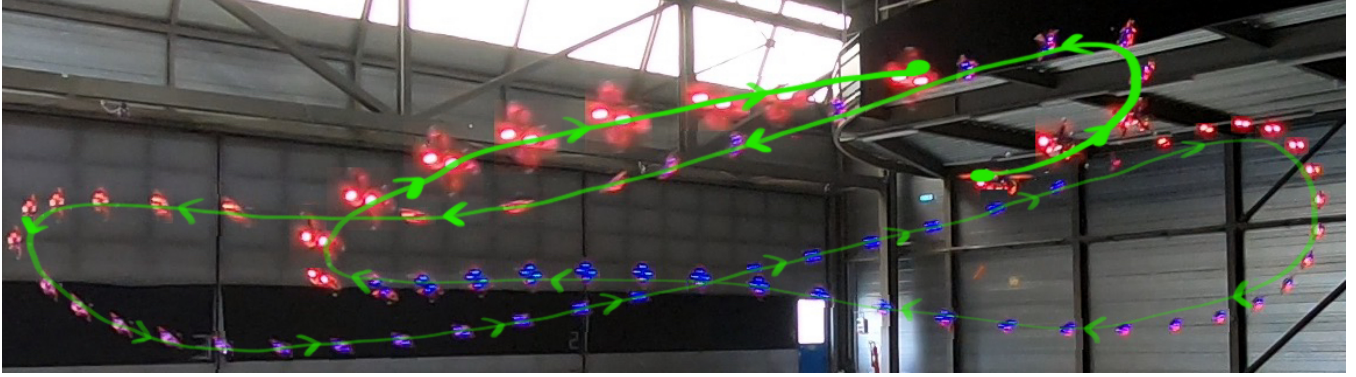


Fig. 1: A quadrotor flies a time-optimal trajectory with top speeds of 15 m s^{-1} . We automatically find a controller configuration that can fly such a high-speed maneuver with a novel sampling-based method called AutoTune. To get a better sense of the speed achieved by the quadrotor, please watch 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 show that AutoTune improves tracking error when flying a physical platform with respect to parameters tuned by a human expert.

SUPPLEMENTARY MATERIAL

A video showing qualitative results in the real world is available at <https://youtu.be/m2q-y7C01So>. Code can be found at 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. The

*The first two authors contributed equally. A. Loquercio and D. Scaramuzza are with the University of Zurich, Switzerland (<http://rpg.ifi.uzh.ch>). A. Saviolo is with the University of Padua, Italy. This work 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).

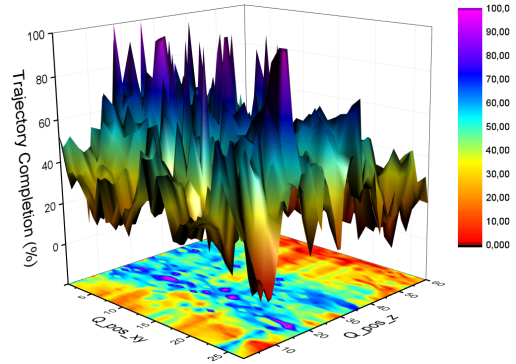


Fig. 2: Trajectory completion (%) as a function of two parameters of a model-predictive controller. The high speed and high angular accelerations required by time-optimal trajectories make the controller extremely sensitive to its parameters.

complex relationship between parameters and performance, empirically shown in Fig. 2, is caused by unavoidable factors such as noisy actuation, 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?

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 evaluations¹ 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

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

control, generally known as MIT rule [1], requires to express 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 [2], [3], [4]. 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 [5], [6], [7]. 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 [8].

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) [9] 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 these experiments, we find out two interesting characteristics of the relationship between controller parameters and flight performance during high-speed flight: (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. In the 2019 AirSim Game of Drones competition [10], our approach decreases lap-time by up to 25% with respect to the winner. 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. 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.
- 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 [1], [2], [3], [4]. 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 [11], [12], [13], 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.

Motivated by this difficulty, another family of approaches estimates the relationship between the controller's performance and its parameters directly from data [14], [6], [5]. 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 [6], [14], [7], and (ii) the function can be described by a parametric distribution, *e.g.* a Gaussian, as typical in inverse optimal control [5]. 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 [15], but this is generally not the case with time-optimal trajectories, which can be faster than the trajectories flown by the best human pilots [16].

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, these algorithms generally define the sampling distribution through heuristics [8]. Building such heuristics to properly fit the problem while minimizing the number of sampling iterations is a major challenge in itself, which limited the success of these methods in quadrotors applications to relatively simple problems, e.g. hovering [17]. Instead of heuristically defining the proposal distribution, we use Metropolis-Hastings [9] to sample the controller parameters. In addition, we propose a strategy specifically tailored to the problem of high-speed flight to reduce the sample complexity of the approach.

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:

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

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

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

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 [16], and then track the resulting trajectory τ_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 the system operates close to its physical limits [18]. Therefore, model predictive controllers (MPC) propose to solve a finite horizon version of the tracking problem in a receding horizon fashion:

$$\min_U \left[\sum_{k=t_0}^{t_0+t_h} \mathbf{x}[k]^\top \mathbf{Q} \mathbf{x}[k] + \mathbf{u}[k]^\top \mathbf{R} \mathbf{u}[k] \right] \quad (4)$$

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

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

where $x[k] = \tau_r[k] - \mathbf{s}[k]$ denotes the difference between the state of the platform and the corresponding reference

at time k , \mathbf{Q} and \mathbf{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 approach, however, requires to tune the controller parameters \mathbf{Q} , \mathbf{R} , and t_h to minimize a user-defined metric, e.g. the tracking error, over the entire trajectory². Tuning these parameters is challenging since it is not possible to analytically find the relationship between them and the long-horizon cost, as required for example by the traditional MIT rule [1]. This difficulty arises from the fact that the available model f is generally inaccurate and does not represent noise in the actuation and other disturbances, e.g. drag.

Since time-optimal trajectories push quadrotors to their physical limits, we track them with an MPC controller and tune the controller parameters through a specifically designed sampling-based technique. To improve performance, 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. 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 [19] 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 . The transition model $t(w_{t+1}|w_t)$ is generally a parametric distribution, e.g. a Gaussian. 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

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

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 $\frac{1}{\alpha}$, 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 $P(w)$.

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

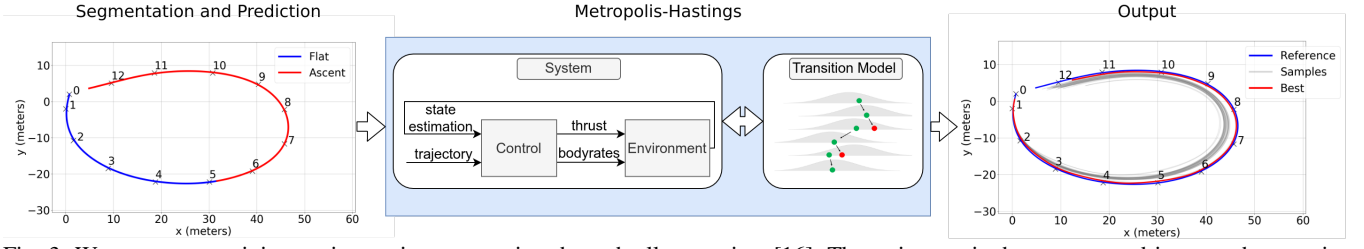


Fig. 3: We compute a minimum-time trajectory passing through all waypoints [16]. The trajectory is then segmented in parts that require different controller behaviors, and initial parameters for each segment are predicted with a regressor. The parameters are then jointly optimized with M-H sampling over multiple rollouts.

In this work, we use the M-H algorithm to find the parameters of a controller flying time-optimal trajectories. In this case, a sample w represents the MPC's parameters, and the score function is defined as:

$$d(w) = \exp(-m(w)), \quad (6)$$

where $m(w)$ is a metric measuring the performance (*e.g.* time) the controller accumulates over the entire trajectory. 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, a parameter configuration that enables to track a given trajectory accurately. Therefore, we continue the sampling procedure up to when we find a solution satisfying some user-defined performance metrics, *e.g.* in terms of tracking error or trajectory completion.

It is important to note that this setup makes the use of Metropolis-Hasting sampling equivalent to simulated annealing with constant temperature [20]. 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 [21] and a specifically designed heuristic to define the cooling function [20]. 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. However, in contrast to fixed-temperature simulated annealing, the use of Metropolis-Hastings additionally provides the possibility to approximate the distribution of controller parameters (*c.f.* Fig. 2), which can be 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 by more than 1.3 m or crashes. 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. As soon as we find

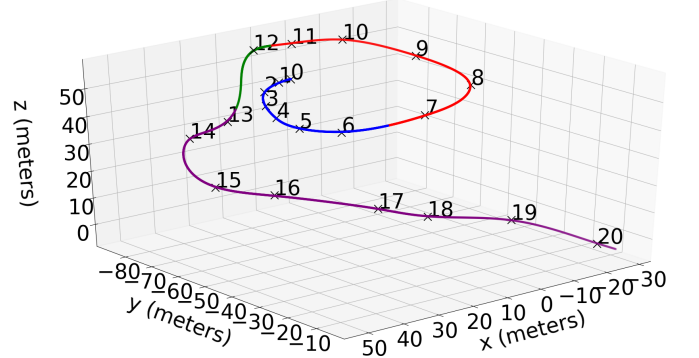


Fig. 4: Before starting the sampling, AutoTune segments a trajectory according to the gradient of z . The above maneuver was split into flat (blue), ascent (red), drop (green, steep descent), and descent (purple).

a controller configuration able to track the entire trajectory without crashing or missing any waypoint by more than 1.3 m, we stop the sampling procedure.

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 particularly 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. The resulting segments are then clustered such that the minimum segment duration is

Track	Max Vel [m s ⁻¹]	Random Search	Bayesian Optimization					AutoTune (Ours)
			Mat	LocalPer	RQ	SQ	Per	
Circle	20	100	100	100	100	100	100	100
Circle	34	65	80	65	60	60	30	100
Drop	20	40	100	50	50	50	40	100
Flip	16	80	80	80	80	80	80	100
Spiral	53	0	10	0	0	10	0	100
Qualifier	20	50	75	60	60	60	30	100
Final	22	10	40	40	30	30	25	100

TABLE I: Comparison of AutoTune with the baselines. All approaches have a maximum budget of 200 iterations to tune the controller. Random search and Bayesian optimization perform well on easy maneuvers, but 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.

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 [22] 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 iterations (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 [10], and Flightmare [23]. We test AutoTune on six trajectories selected to evaluate controller performance under strong accelerations and high-speed on all axes.

For comparison, we use two baselines for controller tuning. A naive one (*Random Sampling*) which randomly samples parameters independently and uniformly on all axis with a variance of 5. Besides, we compare to the strong baseline of *Bayesian Optimization* [7] with multiple choices of the Gaussian kernel. All baselines start tuning from the same point as ours: the trajectory is divided into parts and the regressor predicts initial parameters. All approaches have a budget of 200 iterations to complete the task. Note that the traditional tuning methods based on gradient-based optimization [13], [12], [1] 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:

$$TC = \frac{\sum_{i \in \text{waypoints}} \mathbb{1}[i]}{\sum_{i \in \text{waypoints}} 1}, \quad (7)$$

where the indicator function for waypoint i is

$$\mathbb{1}[i] = \begin{cases} 1 & \text{if drone at 1.3m distance from } i \text{ at } t_r(i), \\ 0 & \text{otherwise,} \end{cases}$$

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 1.3m 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 is consistently the best across all maneuvers. For easy maneuvers, e.g. the slow circle or the flip, almost

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.

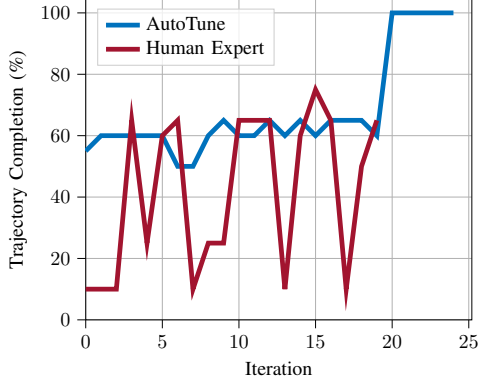


Fig. 5: Comparison between AutoTune and a human expert in tuning the controller for flying the Qualifier track (Fig. 4) in simulation. After 20 attempts, the human expert does not find the controller parameters to successfully fly the entire maneuvers and terminates tuning. Conversely, in 21 iterations AutoTune finds the parameters to fly the entire track.

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 strong Bayesian Optimization baseline widens, reaching up to 90% in the Spiral track. One of the main reasons for this performance gap is the fact that the Bayesian baseline has difficulties fitting the optimization function. Another interesting result of Table I is that Bayesian Optimization is strongly sensitive to the choice of the kernel.

Fig. 5 shows the comparison between the tuning performance of AutoTune and a human expert over time. The results show that the task of tuning parameters by hand is extremely challenging for humans, given the non-intuitive relationship between parameters and performance. Indeed, while the human expert could not tune the controller to fly the entire trajectory, AutoTune finds a feasible controller configuration after only 21 iterations.

To validate the importance of tuning parameters for flying faster trajectories, we compare our approach to the top three methods in the 2019 AirSim Game of Drones competition. 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 s, while the winner only reaches the goal in 30.11 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 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

In this section, we show that 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 [16]. This trajectory is used to tune the controller in the Flightmare simulator. The resulting controller is then evaluated on a physical platform, and its tracking performance is compared with parameters tuned by a human expert. 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, the controller parameters found by our approach 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 is to explicitly tune the controller on multiple motion primitives, in a procedure similar to domain randomization [24]. At the cost of a larger sample complexity, such procedure could make the parameters not only robust to changes in track layout, but also favour simulation-to-reality transfer.

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

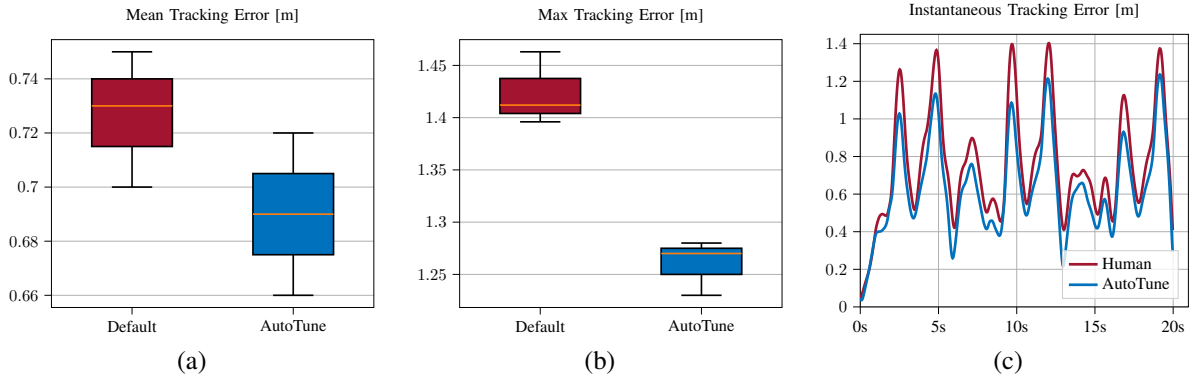


Fig. 6: Results in the real world. After tuning the parameters in simulation, we evaluate the best configuration found by AutoTune on a physical platform. We compare the performance with the parameters tuned by a human. We perform three runs for each parameter set. We also report the error as a function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

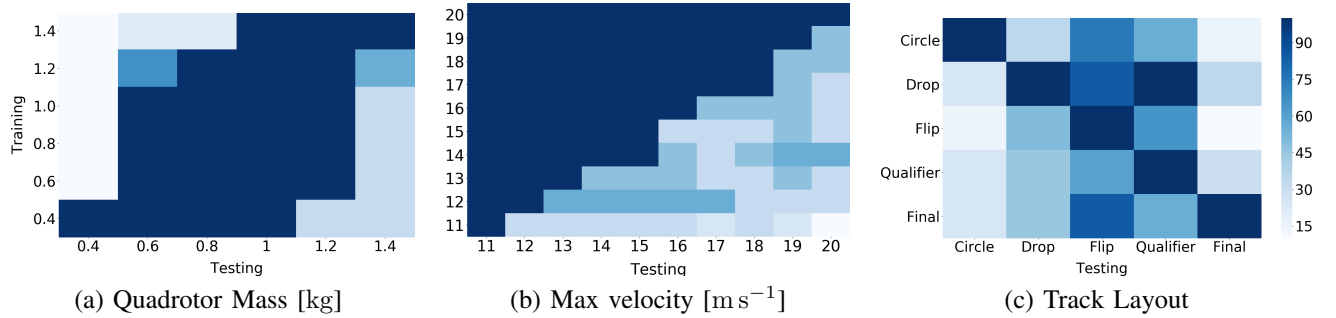


Fig. 7: Performance analysis when parameters used for tuning (Training) is different to the one used during execution (Testing). Overall, the parameters are robust to imperfect identification of the mass, and the faster the maneuver, the more sensitive the controller is to the parameters. However, the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

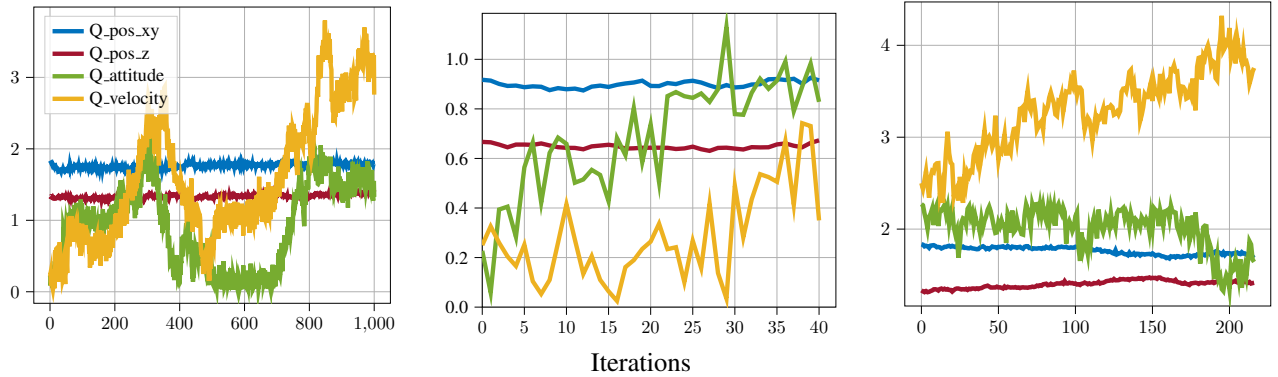


Fig. 8: Path in parameter space for different initial conditions. The parameters' values are divided by their maximum value achieved over all runs. AutoTune converges to a configuration with 100% trajectory completion for all initial conditions. Initialization strongly affects the converge speed. All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

completion for all initial conditions. However, while some require as little as 50 iterations, 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 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 [4], [6], [6], 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 function between performance and controller configurations.

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

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

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

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 iterations 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 iterations. 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 of controller parameters 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 a minimum-time trajectory at the limits of handling. In such cases, the relation between the parameters of a finite-horizon controller and the flight performance over the entire trajectory (measured, for example, in terms of trajectory completion or tracking error) is non-convex, not injective, and multi-modal. This complex relation, caused by noisy actuation, disturbances, or imperfect modeling, makes tuning challenging even to human experts. In this paper, we propose a sampling-based approach specifically tailored to the task of high-speed flight. This approach achieves very good tuning performance both in simulation and in the physical world.

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 [6], [7], [4], such techniques are generally suited to hovering or low-speed flight, and they are generally too conservative for time-optimal flight. Similar to previous work on agile flight [25], 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. Such considerations will potentially open the door to faster flight on physical quadrotors than what is possible today [26].

REFERENCES

[1] K. J. Åström, “Theory and applications of adaptive control - A survey,” *Autom.*, vol. 19, no. 5, 1983.
[2] M. J. Grimble, “Implicit and explicit lqg self-tuning controllers,” *Automatica*, vol. 20, no. 5, pp. 661–669, 1984.

[3] 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.
[4] 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.
[5] M. Menner and M. N. Zeilinger, “Maximum likelihood methods for inverse learning of optimal controllers,” *IFAC World Congress*, 2020.
[6] 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.
[7] 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.
[8] Y. Davidor, “Genetic algorithms and robotics: a heuristic strategy for optimization,” *World Scientific*, 1991.
[9] 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.
[10] 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,” *PLMR Post Proceedings of the NeurIPS 2019 Competition Track*, 2020.
[11] S. Lupashin, A. Schöllig, M. Sherback, and R. D’Andrea, “A simple learning strategy for high-speed quadcopter multi-flips,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 1642–1648.
[12] A. Julkanusart 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. 004 865–004 870.
[13] L. Cedro and K. Wiczorkowski, “Optimizing pid controller gains to model the performance of a quadcopter,” *Transportation Research Procedia*, vol. 40, pp. 156–169, 2019, international Scientific Conference on Sustainable, Modern and Safe Transport.
[14] 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.
[15] D. Ramachandran and E. Amir, “Bayesian inverse reinforcement learning,” in *International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann Publishers Inc., 2007, p. 2586–2591.
[16] P. Foehn and D. Scaramuzza, “Cpc: Complementary progress constraints for time-optimal quadrotor trajectories,” *arXiv preprint arXiv:2007.06255*, 2020.
[17] 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.
[18] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
[19] W. K. Hastings, “Monte carlo sampling methods using markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.
[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, 1983.
[21] M. Fielding, “Simulated annealing with an optimal fixed temperature,” *SIAM J. Optim.*, vol. 11, no. 2, 2000.
[22] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.
[23] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, “Flightmare: A flexible quadrotor simulator,” *Conference on Robot Learning (CORL)*, 2020.
[24] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, “Deep drone racing: From simulation to reality with domain randomization,” *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, 2020.
[25] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, “Deep drone acrobatics,” *Robotics: Science and Systems (RSS)*, 2020.
[26] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, “Alphapilot: Autonomous drone racing,” *Robotics: Science and Systems (RSS)*, 2020.
[27] 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.

VII. SUPPLEMENTARY

A. Tracks

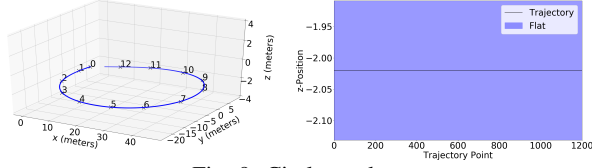


Fig. 9: Circle track.

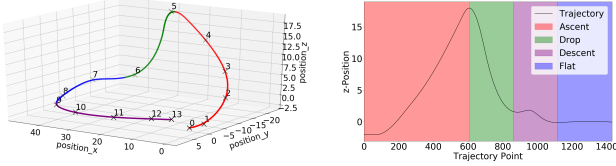


Fig. 10: Drop track.

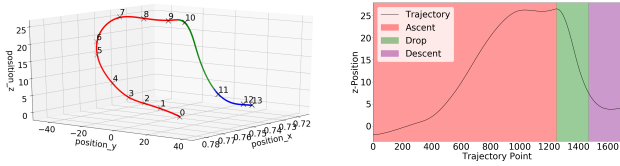


Fig. 11: Flip track.

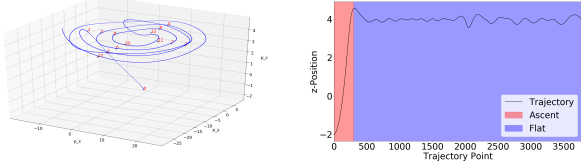


Fig. 12: Spiral track.

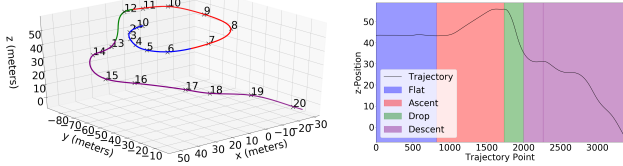


Fig. 13: Qualifier track.

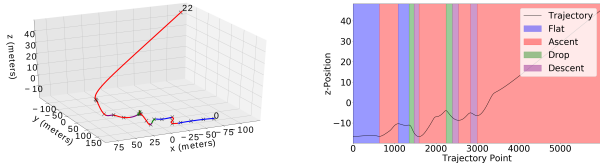


Fig. 14: Final track.

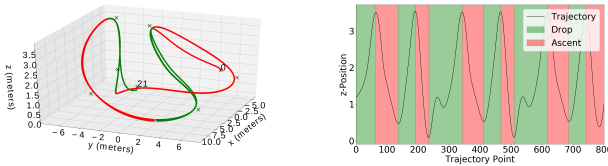


Fig. 15: SplitS track.

Specifically, we select the following maneuvers:

- Circle: 2D maneuver with a radius of $r = 16$ m. Consists of 12 waypoints at an average of 9 m distance.
- Drop: Ascent of 21 m in z over 50 m length followed by a steep descent of 16 m in height. Ends with a semi-circle with radius $r = 16$ m. Consists of 12 waypoints at an average of 10 m distance.

- Flip: Ascent of 25 m in z over 117 m length, with an half-loop in z of 14 m radius. Ends with a steep descent of $r = 20$ m over 28 m length. Consists of 12 waypoints at an average of 13 m distance.
- Spiral: Ascent of 6 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 $r = 6$ 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 m distance.
- Qualifier: Track used for the qualification round of the 2019 AirSim Game of Drones competition. Made of an ascending circle of 160 m, a 23 m drop in height over 25 m length and a long straight descent of 148 m length. Consists of 20 waypoints at an average of 16 m distance.
- Final: Track used for the final round of the 2019 AirSim Game of Drones competition. Made of a 209 m long segment with a hairpin and loop, followed by a 40 m ascent over approximately 296 m. Consists of 22 waypoints at an average of 22 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 $8 - 15 \text{ m s}^{-1}$. The SplitS trajectory consists of a sequence of ascents and drops of 3 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. 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 [27], without perception constraints, and the time-optimal planner de-

scribed by Foehn [16]. 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 ($Q_{\text{pos_xy}}$, $Q_{\text{pos_z}}$, Q_{attitude} , Q_{velocity}) and the horizon length, while we fix the MPC input costs (R_{thrust} , $R_{\text{pitchroll}}$, 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. 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.