# A Comparative Study of Autonomous Quadrotor Race Trajectory Generation Methods and Controllers

Sofya Akhetova, Alex Choi, Barkın Sarıgöl

***Abstract*— The value of the quadrotor racing industry is projected to quintuple in the next 10 years, partly due to the emergence of autonomous drones. This raises the need to design race trajectories and controllers to reach waypoints and avoid obstacles while balancing the need to minimize race time, computational time, and maximize robustness to wind disturbance and initial condition offsets. In this paper, iLQR and various SQP cost functions were used to generate offline trajectories for a 6-DOF 12-state quadrotor. Additionally, TV-LQR and different MPC methods were used on generated trajectories to test robustness. We then conducted a comparative study on the above methods and found that there is an inherent trade-off between minimum time and robustness. Linear MPC controllers proved to be more robust than LQR and achieved a reasonable computational time when following the generated trajectories.**

## I. Introduction

Drone racing is projected to grow quintuple in the next decade, resulting in competitions with million dollar prize pools [1] [2]. This motivates the creation of control algorithms that can traverse multiple way-points in minimal time while being robust to perturbations in initial conditions and winds.

Constructing a minimum time open-loop trajectory can restrict the amount of control effort remaining for the stabilization of the system. As a result, they can become fragile in real-world conditions where wind and initial state offsets are unavoidable. This highlights the importance of designing not just an optimal trajectory, but also a controller that can robustly track it under realistic disturbances.

We explored two core components: trajectory optimization and trajectory stabilization. For trajectory optimization, we used iLQR to jointly generate and track time-optimal trajectories. As it runs very quickly, we can wrap it inside a minimum-time search which can be adjusted to favor robustness over aggressiveness. We also used Sequential Quadratic Programming (SQP) to get minimal time trajectory by shortening the optimizer's time budget until it failed to reach the next goal. We tested SQP with various cost formulations like fixed-time budget, variable-time budget, weighted time, and weighted next goal. For stabilization, we evaluated several controllers to track these offline-generated paths, including time-varying LQR, linear MPC, and nonlinear MPC (with and without warm starts). Each controller was tested for its trajectory tracking accuracy and online computational performance under wind and initial condition disturbances.

### A. Related work

Models for 6-DOF quadrotor systems emerged in the academic literature with work such as [3], and became a standard benchmark for testing control systems, state estimation and trajectory optimization [4] [5]. The quadrotor model is a 12 state, 4 input system with a drag model [6]. Wind is precomputed and modeled as a spatially-varying temporally-constant field. In a large portion of the literature, the temporally-constant field is superposed with temporally-varying mean-zero gusts modeled using the Dryden or von Karman spectra. For simplicity, temporal variation in this work is neglected. Approximating wind as the mean prevailing wind over time is also a common approach and is used in literature such as [7].

Many methods have been proposed for solving the trajectory generation problem under multiple waypoints. This includes methods such as generating piecewise polynomial segments [5], mixed integer quadratic programming [8], and sampling based algorithms [9]. These typically result in optimal open loop trajectories solving minimum time.

To stabilize around the generated trajectories, multiple MPC implementation strategies are explored based theory taken from [10].Though the formulations are detailed in Section III, readers interested in further details are referred to [10].

### B. Statement of contributions

By comparing a large variety of controllers and trajectory generation methods in a realistic quadrotor model, race trajectory, and disturbances, we provide a unified assessment of quadrotor guidance and control. The breadth of the trajectory design and controller design assessment allows one to make an informed choice on autonomous quadrotor race strategy. Additionally, to the best of the authors' knowledge, we introduce a novel method of time minimization in an iLQR formulation.

## II. Problem formulation

There are two problems we try to solve. First, the minimum time trajectory optimization attempts to solve the following discrete time optimization problem.

$$\begin{aligned} \min_{x[\cdot],u[\cdot]} \quad & t_f \\ \text{subject to} \quad & \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \\ & \mathbf{u}_k^{\text{lo}} \leq \mathbf{u}_k \leq \mathbf{u}_k^{hi} \quad \forall k \in 1,\ldots,N-1 \\ & h(x[\cdot]) > \mathbf{0} \end{aligned}$$

Where $t_f$ is the final time, $f(\mathbf{x}_k, \mathbf{u}_k)$ describes the dynamics of the system, and $h(x[\cdot]) \in \mathbb{N}_0^G$ is a function that returns the number of time indices spent in the radius of each waypoint. The problem is complicated by the fact that the dynamics are nonlinear, the function $h$ is integer valued and non-convex, and that optimizing over a changing length trajectory is itself an integer-like non-convex problem. Our techniques will not solve this problem directly, but use techniques that solve either a heuristic problem that is similar, or solve many optimization problems to approach a local solution.

Once the nominal open loop trajectories, $(\bar{x}[\cdot], \bar{u}[\cdot])$, are defined, we then solve an additional tracking problem of the following form.

$$\begin{aligned} \min_{u[\cdot]} \quad & c(x[\cdot], u[\cdot], \bar{x}[\cdot], \bar{u}[\cdot]) \\ \text{subject to} \quad & \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \\ & \mathbf{u}_k^{\text{lo}} \leq \mathbf{u}_k \leq \mathbf{u}_k^{hi} \quad \forall k \in 1, \ldots, N-1 \end{aligned}$$

Where $c(\cdot)$ is some cost function that minimizes some distance between the nominal trajectory and the current state.

### A. Model

We use a standard 12-state 6-DOF quadrotor model, defined below as a control affine system. The state vector is composed of three positions, three Euler angles, and their derivatives. Instead of individual motor torques, the system utilizes net vertical thrust and the three body torques (for pitch, roll, and yaw) as the control inputs. The model includes two coordinate systems: a fixed inertial coordinate system (with axes North, East, Down), and a body reference system centered on the quadrotor (with axes longitudinal, lateral, and down). The angles between these coordinate systems are described using ZYX Euler angles. While the model is nonlinear, a small angle approximation is used to equate the angular velocities in inertial and body coordinates. A full description of the dynamics can be found in appendix A.

These dynamics are described in continuous time, but simulated using discrete time methods.

$$\begin{gathered} \mathbf{q} = \begin{bmatrix} x & y & z & \psi & \theta & \phi \end{bmatrix}^\intercal \\ \dot{\mathbf{q}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & p & q & r \end{bmatrix}^\intercal \\ \mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} \\ \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u} \end{gathered}$$

Quadcopter drag is the sum of frame drag and the interaction between relative wind and propellers, often modeled using Blade Element Momentum (BEM) theory [7]. In this paper, we neglect the propeller interaction and model the drag only as frame drag with force pointing in the direction of relative wind (taking into account the wind field) and magnitude given by:

$$F_{drag} = \frac{1}{2}\rho V_{rel}^2 C_d S$$

The quadrotor also faces control limits. Since the propellers only spin in one direction, the thrust limits are given by $0 \leq T \leq 2g$ N. The pitch, yaw and rolling moment limits are given by $-0.5 \leq P \leq 0.5, -0.5 \leq Y \leq 0.5$ and $-0.5 \leq R \leq 0.5$ Nm.

### B. Course

We define the racecourse as a series of waypoints which must be achieved in a particular order. Each waypoint defines a spherical region of space with a radius of 1m where the center of the quadrotor must intersect. The quadrotor may approach the way-points from any direction, and there are no state constraints other than position which must be met.

The particular course used for the rest of the paper involves a zig-zag pattern with sharp turns and changes in elevation. This layout provides a non-trivial control sequence that is likely to push against the control input limits.

### C. Disturbances and Perturbations

During a race, quadrotors face wind disturbances and perturbations from a pre-planned trajectory. So, the trajectory and controller design must be robust to such effects.

Wind disturbances impact the drone through drag by changing the relative wind; the difference between quadrotor ground speed and local wind vector.

Perturbations from a trajectory that is planned offline are treated in two ways. Firstly, a quadrotor may start the race at a slightly different location than planned. To model this, we add an offset to the first 3 states (x,y,z positions) of the quadrotor at $t_0$. Secondly, for multitude of different reasons, the quadrotor may end up at a different state than the preplanned trajectory at any time instance throughout the race. To model this, we add an offset to all states of the quadrotor at $t_0$, effectively treating the trajectory as the tail problem of a larger trajectory.

## III. Proposed Solution

The problem is addressed in two steps. First, open loop trajectories are generated offline, optimizing for different combinations of final time and robustness. Second, controllers are designed to stabilize the system to these nominal trajectories.

### A. Trajectory Optimization

Two techniques are explored for the generation of nominal minimum time trajectories, those being iterative LQR (iLQR) and sequential quadratic programming (SQP). iLQR provides both a nominal trajectory and control policy for stabilization and can be compared in isolation. SQP only generates an open loop trajectory, and so will have its robustness compared using trajectory stabilization methods.

*1) iLQR:* iLQR generates trajectories by iteratively solving an approximated LQR problem on some initial nominal trajectory. The policy is shot forwards in time to generate a new trajectory to solve around. First, we describe the process for generating a (not minimum time) trajectory for a single way-point, then generalize the procedure to handle multiple way-points and minimum time.

We start with some initial trajectory, $(\bar{x}^{(0)}[\cdot], \bar{u}^{(0)}[\cdot])$, which we define as a linear interpolation between the initial

state/control and the goal state/control. Note, the trajectory of variables over time is notated as $x[\cdot]$, and the $k$th index is notated as $x_k$. Iterations of trajectories and states will be notated with a superscript.

$$\bar{x}_k^{(0)} = x_0 + \frac{x_g - x_0}{N}k$$
$$\bar{u}_k^{(0)} = u_0 + \frac{u_g - u_0}{N}k$$

This implies that the first trajectory is almost guaranteed to be dynamically infeasible, but it provides a reasonable initial trajectory for linearization. We define the deviation coordinates of this trajectory as $\Delta x_k = x_k - \bar{x}_k$ which evolves with approximately linear dynamics, $\Delta x_{k+1} \approx A_k^{(0)}\Delta x_k + B_k^{(0)}\Delta u_k$, with no constant term.

We then define another coordinate system that tracks the deviation from the goal as $\delta x_k = x_k - x_g = \Delta x_k + \bar{x}_k - x_g = \Delta x_k + \hat{x}_k$. This can be used to define a cost function that penalizes deviation from the final goal state.

$$c(\delta x[\cdot], \delta u[\cdot]) = \delta x_N^\intercal Q_N \delta x_N + \sum_{k=0}^{N-1} \delta x_k^\intercal Q \delta x_k + \delta u_k^\intercal R \delta u_k$$

When written with respect to $\Delta x$, this results in an LQR tracking problem whose cost term includes quadratic, linear, and constant terms. In general, the optimal cost-to-go, $J^*(x,t)$, and the optimal control, $\Delta u^*(x,t)$ can be solved as follows.

$$\begin{aligned}
J^*(\Delta x, k) &= \Delta x^\intercal \mathbf{P}_k \Delta x + 2\Delta x^\intercal \mathbf{p}_k + p_k \\
\Delta u^*(\Delta x, t) &= \mathbf{K}_k \Delta x + \mathbf{k}_k \\
\mathbf{P}_k &= Q_{xx} + Q_{ux}^\intercal \mathbf{K}_k + \mathbf{K}_k^\intercal Q_{ux} + \mathbf{K}_k^\intercal Q_{uu} \mathbf{K}_k \\
\mathbf{p}_k &= q_x + Q_{ux}^\intercal \mathbf{k}_k + \mathbf{K}_k^\intercal q_u + \mathbf{K}_k^\intercal Q_{uu} \mathbf{k}_k \\
p_k &= c_0 + \mathbf{k}_k^\intercal Q_{uu} \mathbf{k}_k + 2\mathbf{k}_k^\intercal q_u \\
\mathbf{K}_k &= -Q_{uu}^{-1} Q_{ux} \\
\mathbf{k}_k &= -Q_{uu}^{-1} q_u \\
Q_{uu} &= R_k + B_k^\intercal \mathbf{P}_{k+1} B_k \\
Q_{xx} &= Q_k + A_k^\intercal \mathbf{P}_{k+1} A_k \\
Q_{ux} &= B_k^\intercal \mathbf{P}_{k+1} A_k \\
q_x &= A_k^\intercal \mathbf{p}_{k+1} + q_k \\
q_u &= B_k^\intercal \mathbf{p}_{k+1} + r_k \\
c_0 &= p_k + c_k \\
q_k &= Q_k \hat{x}_k \\
r_k &= R_k \hat{u}_k \\
c_k &= \hat{x}_k^\intercal Q_k \hat{x}_k + \hat{u}_k^\intercal R_k \hat{u}_k
\end{aligned}$$

Critically, there is no guarantee that $Q_{uu}$ is invertible. The term can be regularized by adding some $Q_{uu} \leftarrow Q_{uu} + \lambda I$ for some $\lambda > 0$ which makes all eigenvalues positive. However, because $Q_{uu}$ is calculated backwards in time, it is difficult to predict what value of $\lambda$ is necessary. It is tempting to include some arbitrarily large $\lambda$ to ensure invertibility, however this causes the gain matrices to be small. This severely limits any deviation from the nominal trajectory and results in slow convergence of the iLQR algorithm. Our implementation instead initializes the regularization term with a small value, $\lambda = 1$ and calculates $Q_{uu}$ backwards in time while checking for positive eigenvalues. If $Q_{uu}$ is ever singular, the regularization term is multiplied by a factor $\lambda \leftarrow 2\lambda$, and the backward time calculation is re-attempted.

Once all gain terms are calculated, the policy is simulated forward in time to generate new trajectories, $(\bar{x}^{(i+1)}[\cdot], \bar{u}^{(i+1)}[\cdot])$. During simulation, all control input constraints are enforced to ensure that the new proposed trajectory does not exceed any limits. The above procedure is repeated until the percent change of the total cost along of the trajectory is less than some tolerance, $1\%$.

*2) Minimum Time iLQR with Way-points:* The above formulation of iLQR includes a single way-point and has fixed final time. The former can be addressed by considering a time varying vector of goals. Let $x_g^1, x_g^2, \ldots, x_g^G$ represent the $G$ way-points, and let $\mathbf{x}_g$ be a length $N$ vector representing the desired goal at each point in time. That is,

$$\mathbf{x}_g = \left[x_g^1, \cdots, x_g^1, x_g^2, \cdots, x_g^2, \cdots, x_g^G, \cdots, x_g^G\right]^\intercal \in \mathbb{R}^{N \times n}$$

Each section of goals is of length $T_i$. Generating trajectories with $x_g = \mathbf{x}_{g,k}$ is equivalent to solving a sequence of trajectory optimizations where the goal changes at certain points in time.

At this point, typically a minimum-time optimal control problem can be solved via time bisection as seen in [11, pg. 145]. However, iLQR is sensitive to the initializing trajectory, and may not be able to find suitable trajectories even if solutions exist. Instead, we utilize the fast computation speed of iLQR to use a less efficient but higher stability search that results in faster open loop trajectories.

Start with some initial $\mathbf{x}_g^{(0)}$ that results in a valid trajectory (one that contacts all way-points). For the resulting open loop trajectory, we calculate a vector $h(x[\cdot], u[\cdot]) \in \mathbb{N}_0^G$ representing the number of time steps where the quadrotor is within range of each way-point. For valid trajectories, $h_i(x[\cdot], u[\cdot]) > 0 \ \forall i \in 1, \ldots, G$. The goal with the largest amount of time-in-goal can be found as $i_{\max} = \arg\max h(x[\cdot], u[\cdot])$. The goal vector, $\mathbf{x}_G$, is then shrunk such that $T_{i_{\max}} \leftarrow T_{i_{\max}} - 1$. The iLQR algorithm is run again with this new goal vector and can be initialized by using a truncated version of the previously calculated trajectory. This warm-starting results in even faster iLQR convergence, typically requiring fewer than five iterations to converge. The procedure is repeated until any further time truncation results in invalid trajectories.

This procedure results in a sequence of valid open loop trajectories that progressively have shorter final times. The final trajectory is typically aggressive and difficult to stabilize, indicated by how much of the trajectory is spent at the control limits. To find a sufficiently conservative trajectory that can be close loop stabilized, we iterate back through the sequence of trajectories until a satisfactory trajectory is found.

*3) SQP:* Our other trajectory optimization approach is based on Sequential Quadratic Programming (SQP), where the nonlinear dynamics of the drone are linearized around a nominal trajectory at each iteration. We solve a convex

Quadratic Program (QP) at each step using this linearization and update the nominal trajectory by simulating the nonlinear dynamics forward. This process is repeated for 15 SQP iterations to refine the linear model and improve the final result.

To find the fastest feasible trajectory, we wrap this SQP-based optimization inside a binary search over the time horizon. At each binary search step, we test whether a trajectory of a given duration can successfully reach all goals. We then shorten or lengthen the time budget depending on if that run was successful until the minimum feasible time is found. We can set this time budget in two ways: either by allocating a constant time budget to each goal, or by allocating a varying time budget for all goals and subtracting the time taken to reach each goal from given budget.

We tested various cost functions in SQP. The general structure is to penalize the Euclidean distance between the predicted drone positions and the goal centers over the time horizon. We consider four main variants of this cost:

$$\text{(1) Uniform cost:} \quad J = \sum_{t=0}^{T} \left\| x_t^{\text{pos}} - g_{\text{current}} \right\|_2$$

$$\text{(2) Time-weighted cost:} \quad J = \sum_{t=0}^{T} t \cdot \left\| x_t^{\text{pos}} - g_{\text{current}} \right\|_2$$

(3) Goal-weighted cost:

$$J = \sum_{t=0}^{T} \left( \left\| x_t^{\text{pos}} - g_{\text{current}} \right\|_2 + w_d \cdot \left\| x_t^{\text{pos}} - g_{\text{next}} \right\|_2 \right)$$

Here, $x_t^{\text{pos}} \in \mathbb{R}^3$ represents the simulated position at time $t$, $g_{\text{current}}$ is the current goal center, $g_{\text{next}}$ is the next goal center (if applicable), and $w_d = \frac{1}{5}$ is a fixed discount factor applied to future goals.

*4) Trajectory optimization using MPC:* We also tried running a Model Predictive Control (MPC) algorithm with a binary search over the allowed time budget.

At every time step, MPC solves an optimization problem over a fixed horizon of next 20 steps using a linearized version of the drone's dynamics. These dynamics are refined using SQP, where we linearize around the last trajectory and iterate 15 times to improve accuracy.

Since we perform 15 SQP iterations across a 20-step horizon, we are solving 300 optimization problems per step. Along with the binary search that invokes many of these MPC calls, the total computational time becomes about three hours, even while using cost function (1), for a less than 10 seconds of real-life run time. This run time makes trajectory optimization using MPC impractical for real-time use which is why we did not go forward with this method.

### B. Trajectory Stabilization

To be able to stabilize around a trajectory generated offline, we consider three methods; Time-Varying LQR, Linear MPC and Nonlinear MPC, with and without warm-start. Both MPC implementations have been adopted from [10]. In the cost functions of TV-LQR and MPC, we penalize the deviation of states $\Delta x$ and control inputs $\Delta u$ from the preplanned path:

$$J(\Delta x, \Delta u) = \left[ \sum_{k=0}^{N-1} \underbrace{\Delta x_k^T Q \Delta x_k + \Delta u_k^T R \Delta u_k}_{\text{Running cost}} \right] + \underbrace{\Delta x_N^T Q \Delta x_N}_{\text{Terminal cost}} \tag{1}$$

*1) TV LQR:* To be able to determine the control input, we solve the following for k=N-1,...,0 with $P_N = Q$:

$$P_k = Q + A_k^T P_{k+1} A_k - A_k^T P_{k+1} B_k (R + B_k^T P_{k+1} B_k)^{-1} B_k^T P_{k+1} A_k$$

This results in the gain $K_k$ for $k = 0, ..., N-1$:

$$K_k = -(R + B_k^T P_{k+1} B_k)^{-1} B_k^T P_{k+1} A_k, \qquad u_k^* = -K_k \Delta x_k + u_k^{ref} \tag{2}$$

When simulating the policy, $u_k^*$ is clipped according to the control limits.

*2) Linear MPC:* The linear MPC determines $\Delta x$ and $\Delta u$ that minimize the Equation (1), but without the terminal cost. Now, $N$ is the time horizon length instead of the length of the trajectory. One advantage of MPC over TV-LQR is the ability to implement control limits. And as such, we make use of this capability which leads to the following constraints:

$$\begin{aligned} \Delta x_0 &= x_0 - x_0^{ref} \\ \Delta x_{k+1} &= A_k \Delta x_k + B_k \Delta u_k \\ u_k^{ref} + \Delta u_k &\leq u_{limit} \text{ for } k = 0, .., N-1 \end{aligned}$$

The expected control inputs and states in the horizon $(k = 0 : N-1)$ become: $\hat{u} = u^{ref} + \Delta u$, $\hat{x} = x^{ref} + \Delta x$. Only the control input at $k = 0$ is applied: $u_0 = u_0^{ref} + \Delta u_0$. In this implementation, by linearizing about the nominal trajectory, we are able to determine $A_k$ and $B_k$ offline.

*3) Warm starting:* If we anticipate deviating significantly from the reference trajectory $(x^{ref}, u^{ref})$, we can choose to linearize the system about our expected trajectory in the horizon $(\hat{x}, \hat{u})$. So, $A_k$ and $B_k$ depend on the system's expected trajectory. Linearizing around the expected trajectory is referred to as warm-starting and leads to the need to linearize the system at every time step. In MPC, because the horizon shifts one time step forward at every time step, in order to generate $(\hat{x}, \hat{u})$ for the next time step, we append our solution to the MPC problem at the current time step $(k = 0 : N-1)$ with an expectation for k=N $(\hat{x}_N, \hat{u}_N)$ generated as follows:

$$\begin{aligned} \hat{x}_N &= f(\hat{x}_{N-1}, \hat{u}_{N-1}) \\ \hat{u}_N &= \hat{u}_{N-1} \end{aligned}$$

Then, when moving onto the next time step, we apply the following shifting procedure to shift our currently expected control and state horizon forward by one step:

$$\begin{aligned} \hat{x}_{k=0:N-1} &\leftarrow \hat{x}_{k=1:N} \\ \hat{u}_{k=0:N-1} &\leftarrow \hat{x}_{u=1:N} \end{aligned}$$

*4) Nonlinear MPC (NMPC):* The implementation of nonlinear MPC is slightly different and employs SQP. In each SQP iteration $j$ (where $0 \leq j \leq n$), we solve for $\delta u_j$ and $\delta x_j$ which are defined as the update to $(\hat{x}, \hat{u})$ at every SQP iteration. Hence, at the end of each SQP iteration, we update the state and control in the horizon as follows:

$$(\hat{x}_{j+1}, \hat{u}_{j+1}) = (\hat{x}_j, \hat{u}_j) + (\delta x_j, \delta u_j)$$

Based on $\hat{x}_j$ and $\hat{u}_j$, we define:

$$\Delta \hat{x}_j = \hat{x}_j - x^{ref}$$
$$\Delta \hat{u}_j = \hat{u}_j - u^{ref}$$

At every iteration $j$, we write the cost function (1) (without the terminal cost) as a function of $\delta x_j$ and $\delta u_j$ about $\Delta \hat{x}_j$ and $\Delta \hat{u}_j$ to solve for $\delta x_j$ and $\delta u_j$:

$$\underset{\delta x_j, \delta u_j}{\operatorname{argmin}} \sum_{k=0}^{N-1} 2(\Delta \hat{x}_j)^T Q \delta x_j + 2(\Delta \hat{u}_j)^T R \delta u_j + \tag{3}$$

$$\delta x_j^T Q \delta x_j + \delta u_j^T R \delta u_j \tag{4}$$

The constraints become:

$$\delta x_{j,k=0} = x_0 - \hat{x}_{j,k=0}$$
$$\delta x_{j,k+1} = A_{j,k} \delta x_{j,k} + B_{j,k} \delta u_{j,k} + r_{j,k}$$
$$\delta u_{j,k} + \hat{u}_{j,k} \leq u_k^{limit}$$

$A_{j,k}$ and $B_{j,k}$ is the linearization of the system at the $j$th iteration about $(\hat{x}_j, \hat{u}_j)$. Hence, this introduces the need to linearize at every SQP iteration. We now also have the term $r_{j,k} = f(\hat{x}_{j,k}, \hat{u}_{j,k}) - \hat{x}_{j,k+1}$. This is needed to ensure dynamic feasibility after performing multiple SQP iterations. When performing multiple SQP iterations, we no longer have the guarantee that: $f(\hat{x}_{j,k}, \hat{u}_{j,k}) = \hat{x}_{j,k+1}$. So, we introduce the term $r_{j,k}$ as a first order correction to the QP. $(\hat{x}_{j=0}, \hat{u}_{j=0})$ is obtained using warm-start. This means that $(\hat{x}_{j=n,k=N}, \hat{u}_{j=n,k=N})$ at this time step becomes $(\hat{x}_{j=0,k=N-1}, \hat{u}_{j=0,k=N-1})$ at the next time step. It should also be noted that NMPC becomes mathematically equivalent to linear MPC when only one SQP iteration is performed.

## IV. Experimental results

### A. Trajectory Optimization

The various methods of generating open loop trajectories were compared on the basis of final time. A trajectory is considered valid if it reaches all way-points. In figure 1, these final times are compared.

Figure 2 shows the projected XY vs. Z trajectories for various time allocation and cost function strategies. As expected, the "Constant time" approach results in the slowest trajectory (13s), as it can allocate an excessive—amount of time to each segment. This is reflected in the large, sweeping arcs the robot follows under this policy. The "Varying time" strategy yields better performance (8.3s) by tightening the overall time budget across the full trajectory, producing more direct paths, as seen in figure 2. Interestingly, both the "Time weighted" and "Future goals weighted" approaches performed slightly better (7.5s) than the uniform

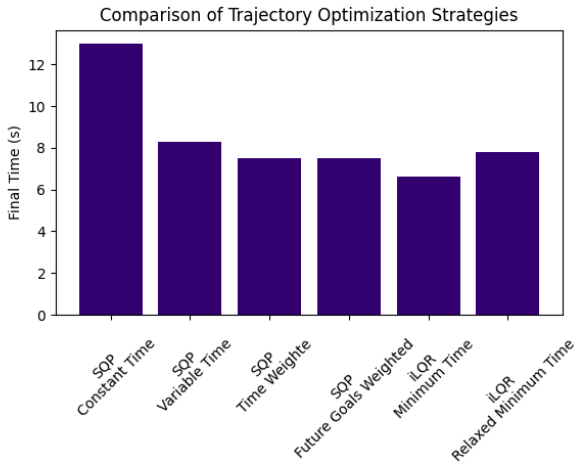



Fig. 1: A comparison of open loop trajectory length.

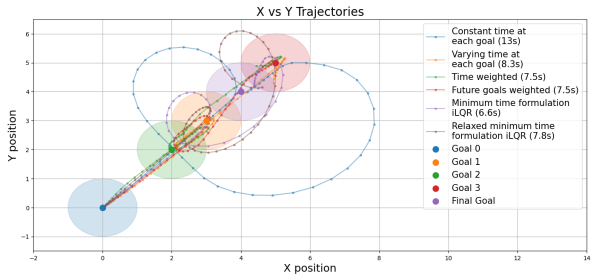
goal formulation. This was surprising, since we initially expected the norm-2 minimization to implicitly encourage faster trajectories through penalizing distance. However, it seems that assigning explicit weight to time and next goal can improve the path in specific scenarios. One of those situations might be where the next goal lies at a more sharp angle and putting more weight on time and next goal leads to a similar trajectory.

The time-truncation search used in the minimum time iLQR formulation resulted in faster open loop trajectories compared to those generated by SQP. This result does not necessarily come from the optimization method itself, but rather from the time-minimization technique. The iLQR time-truncation search enables the current goal to change even if the way-point has not been reached. For a quadrotor system where inertia plays a large role in the dynamics, this means momentum can carry the quadrotor to the way-point while the controller reorients the vehicle towards the next goal. This anticipation of the goal results in trajectories that glance off the surface of the way-point region. In contrast, the cost functions used in the SQP formulation only update the goal once the way-point has been reached.

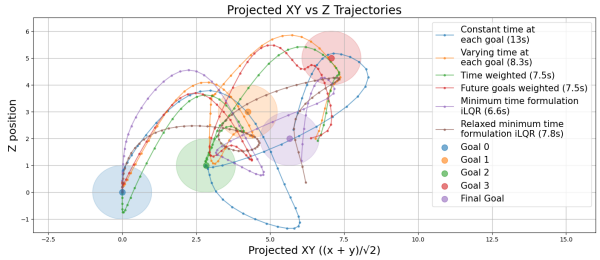
We also demonstrate the unstabilizability of the aggressive minimum time trajectory compared to relaxed trajectories. Figure 3 demonstrates the closed loop control failing to hit all way-points on an aggressively time-minimized open loop trajectory when the initial conditions are perturbed. Figure 4 shows the relaxed open loop trajectory that has sufficient closed loop performance.

### B. Trajectory Stabilization

The trajectory stabilization methods have been tested on the varying-time cost function. SQP-generated trajectory under both an initial condition offset of [0.1,....0.1] and a wind field to examine different controllers' robustness to these factors. Robustness has been quantified in terms of the 'mean trajectory following error (MTFE)', defined by the difference between the x,y,z position traversed by the




(a) X and Y axis




(b) Projected XY and Z axis view

Fig. 2: Comparison of all trajectories: (a) X and Y axis, and (b) projected XY and Z axis

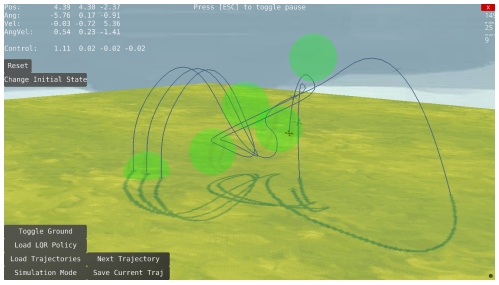


Fig. 3: Aggressive time minimized trajectories that can not be stabilized.

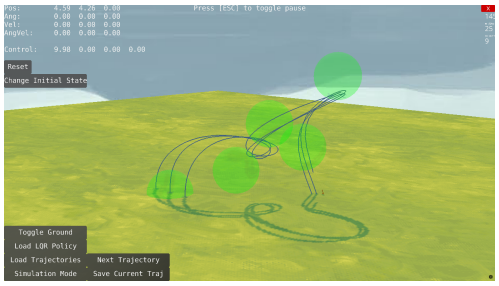


Fig. 4: Relaxed time minimized that can be stabilized.

quadrotor and the SQP-generated reference trajectory's x,y,z coordinates:

$$MTFE = \frac{1}{N}\sum_{i=1}^{N}(x_i - x_i^{ref})^2 + (y_i - y_i^{ref})^2 + (y_i - y_i^{ref})^2$$

where N is the length of the reference trajectory. The cost function for each controller was defined by $Q = Q_0 I$ and $R = I$. $Q_0$ was a scalar multiplier that was varied, taking on the values [1,10,100]. Varying $Q_0$ quantified the controller MTFE's sensitivity to the cost function. Values of $Q_0$ are indicated in the plots below:

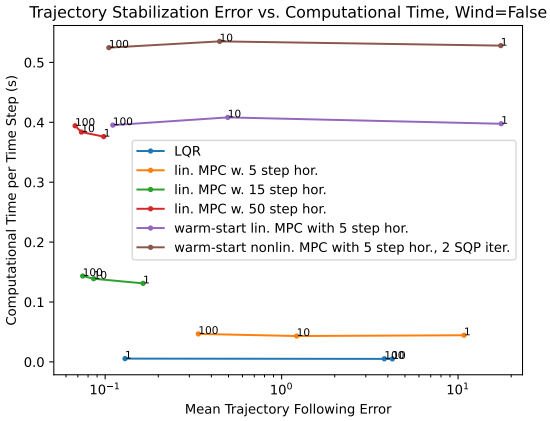



Fig. 5: MTFE versus Compute time under Initial Condition Offset for $Q_0 = [1, 10, 100]$

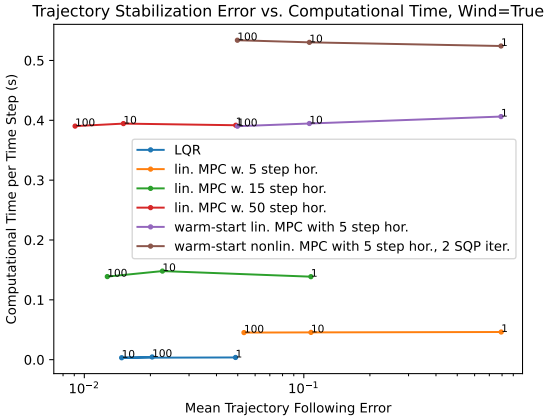



Fig. 6: MTFE versus Compute time in Windfield for $Q_0 = [1, 10, 100]$

The value of $Q_0$ that led to the lowest MTFE for the LQR controller was different in Figures 5 and 6. This shows that the optimal cost function depends on the specific realization of the disturbance. However, for all MPCs, $Q_0 = 100$ resulted in the lowest MTFE error and this makes sense because the MPC controllers already take into account the control limits during the optimization as hard constraints. So, a cost function that prioritizes state-following over control effort performs better.

From both Figures 5 and 6, it was possible to see that LQR's MTFE was lower than the linear MPC with 5 step horizon and higher than linear MPC with 15 step horizon. This means that MPC horizon must be sufficiently long to outperform LQR. We also see that in both figures, increasing the time-horizon significantly improves the performance of the MPC controller.

Additionally, it is evident that warm starting significantly increases the computational time. This is because warm starting requires linearization at every time step. In fact, the computational time of the linear MPC with 50 step horizon was similar to the computational time of the warm-started linear MPC with 5 steps. In both Figures 5 and 6, the 50-step horizon MPC outperforms the 5-step warm-started MPC significantly. This is probably due to the fact that linearizing the dynamics around the reference trajectory (rather than the expected trajectory) is already a good approximation of the nonlinear dynamics since the drone does not stray too far away from the reference trajectory. Because of similar reasons, the nonlinear warm-started MPC with 2 SQP iterations performs only slightly better than the warm-started linear MPC. Because the quadrotor trajectory does not deviate significantly compared to the reference trajectory, multiple SQP iterations do not lead to a significant improvement in tracking. In a quadrotor race, since the quadrotor must meet waypoints, significant deviations from the reference trajectory are not expected.

The key takeaway from Figures 5 and 6 is that given a sufficient time horizon, a simple linear MPC is highly robust and can be run in reasonable computational time. When more computational time is available, the control designer should opt to increase the linear MPC time horizon rather than implement a non-linear MPC or warm-start.

## V. Conclusions and future work

In this paper, we conducted a comprehensive assessment of a wide range of techniques that can be used to plan a quadrotor race trajectory and stabilize around it. These were tested on a realistic course that provided a non-trivial set of way-points that encouraged the control limits to be exceeded. The methods used to plan the race trajectory were uniform-cost SQP, time-weighted cost SQP, goal-weighted cost SQP, and iLQR. To stabilize around this trajectory, we considered linear and nonlinear MPC implementations as well as LQR.

The iLQR-based trajectory optimization was able to complete the race course faster than the SQP-based trajectory optimization. This was because the iLQR implementation started optimizing the controls to reach the next goal before the current goal is reached and made use of the momentum of the quadrotor to reach the current goal. Minimizing time more aggressively during the trajectory optimization made it harder for the LQR controller to meet the waypoints under initial condition disturbances. It was also observed that in a quadrotor race, the trajectory-optimization must be completed offline due to the large computational time needed for planning. This requires an additional controller to stabilize around the trajectory. Under both initial condition offsets and wind perturbation, a linear MPC controller provides a good balance between robustness and computational effort.

One avenue for extension would be to certify regions of attraction that can describe the subset of state space that is stabilizable by the controllers. This would provide a more rigorous metric for the robustness of a trajectory/controller pair. Techniques such as those found in [12] and [13] provide methods for calculating such regions of attraction via the direct computation of Lyapunov functions and probabilistic certification of these regions using neural networks. From there, it may be possible to parameterize the volume of the region of attraction against the final time of the open loop trajectory.

Another method for guaranteeing robustness may be to include the closed loop controller simulation in the open loop trajectory generation algorithm. This would increase the computation time of offline trajectory generation, but at the benefit of only generating robust trajectories.

Another extension would be to include obstacles and barriers to emulate race track elements. As shown in figure 7a and figure 7b, we have developed an obstacle avoidance algorithm through enforcing distance-based constraints within the SQP loop. Along with using a uniform cost function, each SQP iteration linearizes the drone's distance to the obstacle around the current trajectory, enforcing a constraint of the form $g(x) \geq 0$ along with a slack variable (for feasibility), where $g(x)$ represents the signed distance to the obstacle minus its radius. By generating trajectories on these more challenging and realistic racing scenarios, we can compare the various controllers' ability to avoid collisions while tracking. Control barrier functions may be implemented as well to ensure obstacle avoidance.

## References


[1] P. Research. (2024) Racing drones market size, share and trends 2024 to 2034. [Online]. Available: https://www.precedenceresearch.com/racing-drones-market

[2] D. R. C. League. (2024) A2rl and dcl announce autonomous drone race with $1 million prize pool. [Online]. Available: https://dronechampionsleague.com/a2rl-dcl-autonomous-championship/

[3] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, 2004, pp. 4393–4398 Vol.5.

[4] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, *Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2007-6461

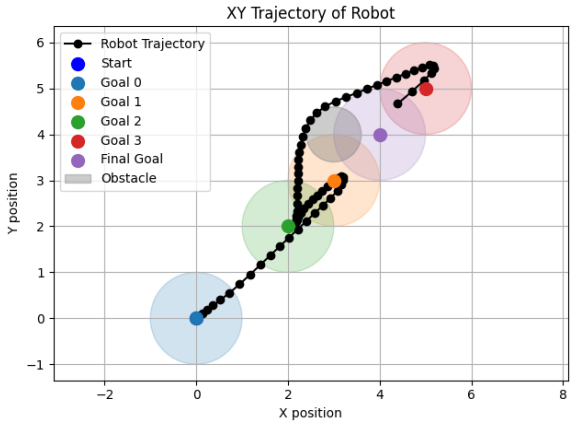
[5] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[6] F. Sabatino, "Quadrotor control: modeling, nonlinearcontrol design, and simulation," 2015.

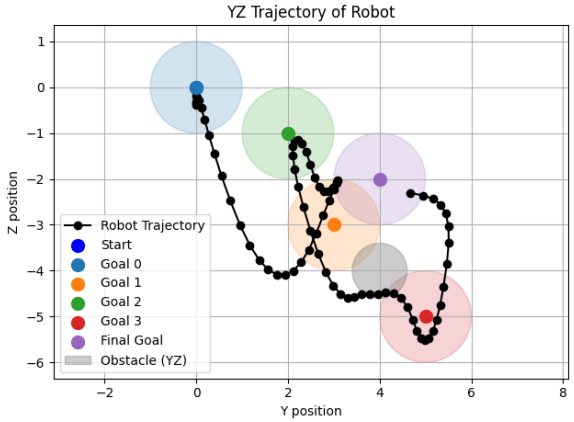
[7] J. Ware and N. Roy, "An analysis of wind field estimation and exploitation for quadrotor flight in the urban canopy layer," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1507–1514.

[8] J. Tordesillas, B. T. Lopez, and J. P. How, "Fastrap: Fast and safe trajectory planner for flights in unknown environments," *CoRR*, vol. abs/1903.03558, 2019. [Online]. Available: http://arxiv.org/abs/1903.03558

[9] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics," *CoRR*, vol. abs/1405.7421, 2014. [Online]. Available: http://arxiv.org/abs/1405.7421




(a) Trajectory in XY view




(b) Trajectory in YZ view

Fig. 7: SQP-based trajectory avoiding an obstacle: (a) top-down XY view, and (b) side YZ view.


[10] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. D. and, "From linear to nonlinear mpc: bridging the gap via the real-time iteration," *International Journal of Control*, vol. 93, no. 1, pp. 62–80, 2020. [Online]. Available: https://doi.org/10.1080/00207179.2016.1222553

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004. [Online]. Available: https://web.stanford.edu/ boyd/cvxbook/

[12] R. Tedrake, "Lqr-trees: Feedback motion planning on sparse randomized trees," in *Proceedings of the 5th Robotics: Science and Systems Conference (RSS)*, Seattle, USA, Jun. 2009, june 28–July 1, University of Washington.

[13] L. Yang, H. Dai, Z. Shi, C.-J. Hsieh, R. Tedrake, and H. Zhang, "Lyapunov-stable neural control for state and output feedback: A novel formulation," 2024. [Online]. Available: https://arxiv.org/abs/2404.07956


## APPENDICES

### A. State Model

TABLE I: Model Parameters and Variables

| Symbol | Description | Value | Unit |
|---|---|---|---|
| $x, y, z$ | Earth Position | - | m |
| $\phi, \theta, \psi$ | Earth to Body orientation | - | rad |
| $p, q, r$ | Body frame angular velocities | - | rad/s |
| $m$ | Total Mass | 1.0 | kg |
| $I_x, I_y, I_z$ | Moments of inertia | 1.0, 1.0, 1.0 | kg-m$^2$ |
| $g$ | Gravitational acceleration | 9.81 | m/s$^2$ |
| $C_d$ | Drag Coefficient | 0.2 | - |

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta} \\ q\cos\phi - r\sin\phi \\ p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\ F_{drag,x}/m \\ F_{drag,y}/m \\ g + F_{drag,z}/m \\ \frac{I_y - I_z}{I_x} qr \\ \frac{I_z - I_x}{I_y} pr \\ \frac{I_x - I_y}{I_z} pq \end{bmatrix}$$

$$\mathbf{G}(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{m}(\sin\phi\sin\psi - \cos\phi\cos\psi\sin\theta) & 0 & 0 & 0 \\ -\frac{1}{m}(\cos\phi\sin\psi + \cos\phi\sin\psi\sin\theta) & 0 & 0 & 0 \\ -\frac{1}{m}\cos\phi\cos\theta & 0 & 0 & 0 \\ 0 & 1/I_x & 0 & 0 \\ 0 & 0 & 1/I_y & 0 \\ 0 & 0 & 0 & 1/I_z \end{bmatrix}$$