Optimization of Flight Paths for Racing Quadrotors Using Sequential Convex Programming

Mike Timmerman and Cameron Hilman

{mtimmerm, chilman}@stanford.edu

June 5, 2024

Abstract

Racing quadrotors are high-speed drones designed for competitive racing through complex courses. Achieving optimal flight paths is crucial to minimize lap times. Traditional trajectory generation methods rely on polynomial methods which lack time optimality and do not exploit the full actuator potential required for high-speed maneuvering. In this paper, we present an approach using Sequential Convex Programming (SCP) and review an approach which introduces Complementary Progress Constraints (CPC) to optimize the flight paths. Traditional trajectory generation methods rely on polynomial methods which lack time optimality and do not exploit the full actuator potential required for high-speed maneuvering. In this paper, we present an approach using Sequential Convex Programming (SCP) and review an approach which introduces Complementary Progress Constraints (CPC) to optimize the flight paths. Traditional trajectory generation methods rely on polynomial methods which lack time optimality and do not exploit the full actuator potential required for high-speed maneuvering.

1 Introduction

The rapid advancements in drone technology have paved the way for various applications, among which high-speed agile quadcopter drone racing stands out as a particularly challenging and exciting field. The primary objective of this project is to develop a trajectory optimization method tailored for drone racing, where the quadcopter must navigate through a series of rings as swiftly and accurately as possible. This scenario epitomizes a classic optimization problem, where the goal is to determine the most efficient path that adheres to the physical constraints and dynamic capabilities of the drone.

Trajectory optimization in this context involves generating a path that meets predefined waypoints while optimizing certain performance criteria. Various optimization techniques can be employed to ensure that the generated trajectory is dynamically feasible and respects the drone's operational limits. The optimization process is driven by a cost function, which can be designed to reflect different objectives, such as minimizing flight time, energy consumption, or maximizing stability.

In this project, we apply Sequential Convex Programming (SCP) to optimize the drone's flight trajectory. Initially, a naïve approach is employed where waypoints are assigned specific timestamps. Building upon this, an enhanced method is introduced based on previous work that simultaneously optimizes both the trajectory and the waypoint allocation using a novel concept known as complementary progress constraint. This dual optimization approach aims to provide a more efficient and effective trajectory for the drone.

To rigorously assess the performance of these optimization methods, we implement a dynamic model of the drone that captures its physical limitations. Additionally, a nonlinear control algorithm is developed to ensure precise tracking of the optimized trajectories. Through this comprehensive approach, we complete the following technical contributions:

- Developing a dynamic model of the drone that incorporates physical limitations.
- Implementing a nonlinear tracking controller to achieve high-accuracy trajectory tracking.
- Utilizing Sequential Convex Programming to optimize the flight trajectory over the flight duration.
- Introducing complementary progress constraints to optimize waypoint allocation.
- Comparing and contrasting the effectiveness of these two optimization approaches in improving trajectory generation.

A substantial body of literature exists on the subject of autonomous drone racing and trajectory optimization. One notable study by Yuyang Shen et al [2]. presents a method for generating time-optimal trajectories for a swarm of autonomous racing drones, emphasizing the challenges of navigating through predefined waypoints with high maneuverability and collision avoidance. Their work underscores the complexity of ensuring efficient navigation in drone racing, requiring sophisticated control and planning algorithms. The use of differential flatness theory in trajectory generation has been pivotal, though traditional polynomial-based methods often fall short of achieving time-optimal paths.

Key milestones in this field include advancements from the IROS autonomous drone racing competitions and the AlphaPilot Challenge, where notable speeds have been achieved, yet human pilots still outperform autonomous drones in some scenarios. A significant advancement was made by Scaramuzza's team with the introduction of the Complementary Progress Constraint (CPC) method [1], which allowed drones to reach speeds of up to 19 m/s, surpassing human capabilities. This project builds on these findings by comparing SCP and CPC approaches, contributing to the ongoing evolution of high-speed autonomous drone performance.

This paper provides a detailed examination of these contributions, highlighting the innovations in trajectory optimization techniques and their practical implications for high-speed drone racing. By comparing SCP with complementary progress constraints, we aim to identify the most effective strategy for achieving optimal drone performance on a racing track.

The paper is structured as follows: section 2 introduces the methodology, where the drone model is described, as well as the controller used by the drone to track the trajectories. It also includes a description of the baseline used to compare the two optimization approaches. Lastly, the section covers the optimization problem employing sequential convex programming, and how introducing complementary progress constraints can solve its shortcomings. The results are summarized in section 3, where the time optimality of trajectory generation methods are compared.

2 Approach

2.1 Drone Model

For the drone model, six-degree-of-freedom dynamics are considered, which consist of 12 states: position p, linear velocity v, attitude η, and angular velocity Ω, and 4 inputs: four rotational speeds of the propellers ω_i, $i \in \{1, \ldots, 4\}$. The angular velocity consists of the components roll rate p, pitch rate q and yaw rate r, and the attitude consists of the components roll ϕ, pitch θ and yaw ψ.

The drone's translational dynamics are derived from Newton's third law:

$$m\mathbf{a} = F_{ext} = D + T + G \tag{1}$$

where m is the drone's mass, $\mathbf{a}$ is the linear acceleration, and F_{ext} is the externally applied force. The linear acceleration is modeled to include contributions from atmospheric drag D, thrust force T generated by the propellers, and gravitational force G.

The rotational dynamics are derived from Euler's rotation equations:

$$I\dot{\Omega} + \Omega \times (I\Omega) = M_{ext} = M_{prop} \tag{2}$$

where I is the inertia tensor of the drone and M_{ext} is the externally applied moments. The moments are modeled as being induced by the rotating propellers.

2.2 Nonlinear Controller

The control design is based on incremental nonlinear dynamic inversion, which makes use of dynamic inversion to cancel out the nonlinear dynamics such that standard control techniques can be used to achieve desired performance. The control architecture is shown in Figure 1, and its cascaded structure is as follows: trajectory regularization relies on the reference position, velocity, and acceleration from the reference trajectory. Together with the current position, velocity, and acceleration, this provides a desired linear acceleration. The acceleration control module uses INDI to determine a desired attitude and control thrust from this reference with the current acceleration. The attitude control module in turn uses INDI to provide a desired control moment from this reference, current attitude, and current angular velocity. The control thrust and moment are converted into propeller angular speeds using control allocation. Further details of these modules are not covered and will instead refer to [2]. We will focus on the reference trajectory module which supplies the controller with a trajectory consisting of a reference position, velocity, and acceleration. Hence, the trajectory optimization methods should be able to supply these three vectors.

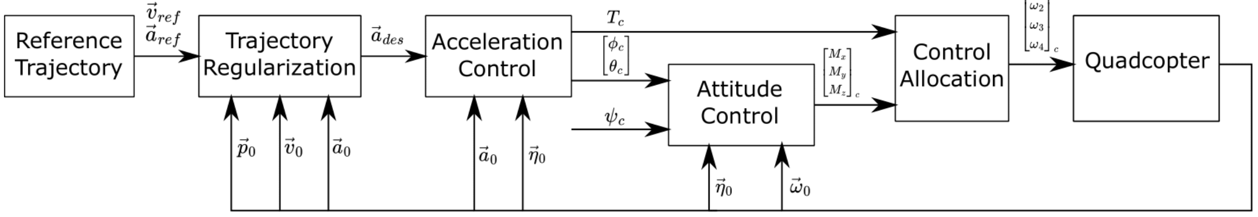
Reference Trajectory
$\vec{p}_{ref}$ $\vec{v}_{ref}$ $\vec{a}_{ref}$
Trajectory Regularization
$\vec{a}_{des}$
Acceleration Control
T_c
$\begin{bmatrix}\phi_c\\ \theta_c\end{bmatrix}$
ψ_c
Attitude Control
$\begin{bmatrix}M_x\\ M_y\\ M_z\end{bmatrix}_c$
Control Allocation
$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix}_c$
Quadcopter
$\vec{p}_0$ $\vec{v}_0$ $\vec{a}_0$
$\vec{a}_0$ $\vec{\eta}_0$
$\vec{\eta}_0$ $\vec{\omega}_0$

Figure 1: Control architecture diagram

2.3 Baseline Trajectory

A baseline trajectory generation method is established as an important measure of the performance of the trajectory optimization methods. This allows for quantifying the improvement over a simple non-optimal trajectory and establishing a lower bound. Additionally, the trajectory optimization methods require an initial guess of the trajectory, so this baseline can be used as a first approximation over which the optimization makes improvements.

The trajectory is constructed such that it passes through a given set of waypoints. For the baseline trajectory, these waypoints are time-stamped with a conservative reference velocity under which the trajectory would be tracked. These pairs of timestamps and waypoint coordinates are fit by a cubic spline with clamped boundary conditions between consecutive waypoints. These boundary conditions ensure continuity up to the second derivative. Therefore, the first derivative of this spline trajectory provides a velocity reference while the second derivative provides an acceleration reference. The set of waypoints is taken from [1] for which the baseline trajectory is shown in Figure 2. Note that the position is cubic, velocity is quadratic and acceleration is linear.

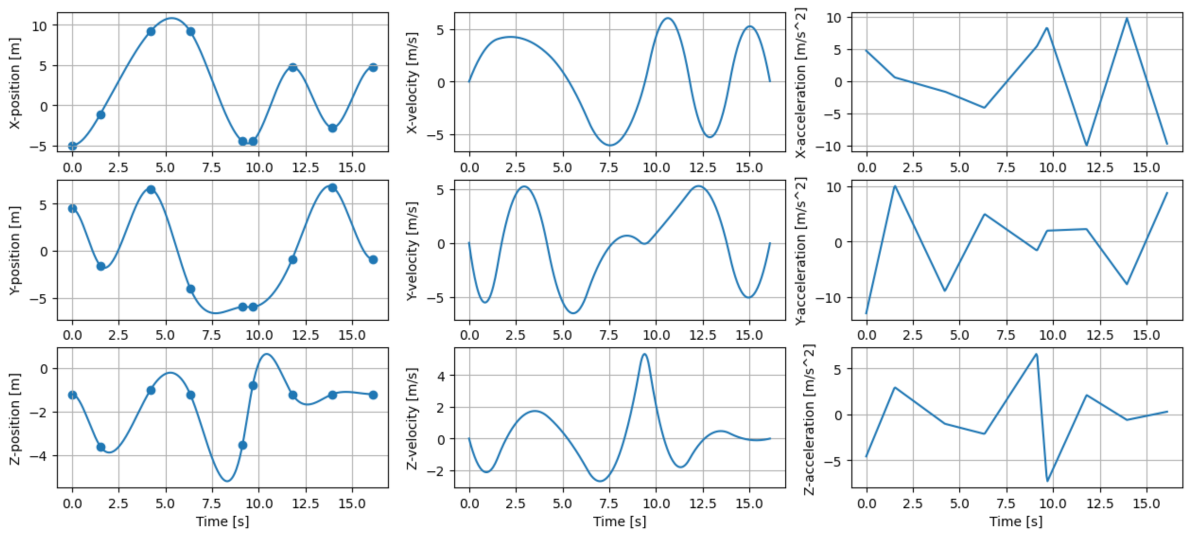

Figure 2: Baseline reference trajectory

2.4 Trajectory Optimization

For trajectory optimization, two methods are considered: a naïve Sequential Convex Programming (SCP) and an extended version that includes complementary progress constraints. SCP optimizes the path length, resulting in faster traversal at a given speed. Since the trajectory does not need to pass exactly through the waypoints, it can be adjusted to navigate around waypoints efficiently. Furthermore, it also results in a dynamically feasible path which enhances tracking performance. However, using a naïve approach, waypoints are assigned specific time steps to ensure in-order traversal, which restricts the variation of time between two waypoints. This limitation is addressed by introducing complementary progress constraints, which allow for the simultaneous optimization of the state and input trajectory, as well as the waypoint time allocation. This approach results in a more flexible and efficient trajectory optimization process.

2.4.1 Sequential Convex Programming

The key idea behind sequential convex programming is to convexify the dynamics around a nominal trajectory and solve the subsequent convex optimization problem. Therefore, an initial trajectory is required, which, however, does not need to be feasible. For this, the baseline trajectory is used. To set up the convex optimization problem in each iteration, a set of objectives and constraints are defined to achieve the desired goal.

Objective Function

The objective function is composed of several parts, each of which have a specific purpose. The full objective function is given as follows:

$$\begin{aligned} J(x_0) = (x_N - x_f)^T P(x_N - x_f) &+ \sum_{i=0}^{N-1} ||u_k||_R + ||p_{k+1} - p_k||_Q \\ &+ \sum_{i=1}^{N-2} ||v_{k+1} - 2v_k + v_{k-1}||_{\bar{Q}} + ||v_{k+2} - 3v_{k+1} + 3v_k - v_{k-1}||_{\bar{Q}} \end{aligned} \tag{3}$$

The terminal objective forces the drone to end near the final waypoint with zero terminal velocity. In the first summation, the input term prevents overly aggressive inputs and the position term optimizes path length. The second summation is included to smooth the velocity and acceleration trajectories. Since these are used as references in the tracking controller, it is important that these trajectories are sufficiently smooth to improve controller performance. The first term is a first derivative approximation and the second term is a second derivative approximation, hence including these terms limits the magnitude of these derivatives. The tuning matrices P, R, Q and $\bar{Q}$ are used to trade off between the different objectives.

Waypoint Tracking Constraints

For consecutive waypoint tracking, the following constraints are included:

$$(p_k - p_{w_j})^T (p_k - p_{w_j}) \leq \tau_j^2, \qquad \forall k \in \{0, \dots N\} \tag{4}$$

where p_k is the drone's position at time k and p_{w_j} is the waypoint position, where j indicates the index of the waypoint. τ_j is the slack variable to allow the trajectory to not have to exactly pass through each waypoint. Together, these constraints specify a time at which each waypoint should be reached. For the time index k, the timestamps from the baseline trajectory are used.

Dynamics Constraints

Dynamics constraints are necessary to ensure the generation of a physically feasible and realistic trajectory. Therefore, discrete-time dynamics are added as constraints on the states:

$$x_{k+1} = x_k + dt f_{RK1}(x_k, u_k), \qquad \forall k \in \{0, \dots N\} \tag{5}$$

where dt is the time step, x_k is the state, and u_k the input at time k. The dynamics are discretized using a first-order numerical integration scheme f_{RK1}.

Trust Region Constraints
Each iteration of SCP involves linearizing the dynamics. Therefore, in a given iteration, updates made to the trajectory are only valid in the linear region around the trajectory linearized in the current iteration. For this reason, constraints are added to prevent an update from deviating too far from this trajectory. These are trust region constraints that have the following form:

$$\begin{aligned} ||u_k - u_k^{(i)}||_\infty &< \rho_u, \qquad \forall k \in \{0, \dots N\} \\ ||x_k - x_k^{(i)}||_\infty &< \rho_x, \qquad \forall k \in \{0, \dots N\} \end{aligned} \tag{6}$$

where $x_k^{(i)}$ and $u_k^{(i)}$ are the states and inputs whose local region is linearized in the ith iteration.

State Constraints
Constraints are enforced upon some of the drone's states to encourage desired behaviour of the drone during the trajectory tracking. The state constraints can be summarized as follows:

$$\begin{aligned} p_{z_k} &\leq 0, \qquad \forall k \in \{0, \dots N\} \\ \phi, \theta &\leq \frac{\pi}{2}, \qquad \forall k \in \{0, \dots N\} \\ \Omega &\preceq \pi \end{aligned} \tag{7}$$

The first constraint introduces a physical constraint on the height of the trajectory, which should not go below the floor (negative z-axis defined upward). The second constraint puts limits on the roll and pitch angles to prevent overly aggressive behavior. The last constraint limits the components of the angular velocity for the same reason. These constraints were established from observations of resulting trajectories.

Time Optimality
Trajectory optimization using sequential convex programming relies on time-stamped waypoints, but the temporal dependence of the trajectory can still be changed. Essentially, these timestamps can be thought of as being normalized between 0 and 1 where the initial position corresponds to 0 and the final position corresponds to 1. The time in which the drone is set to track this trajectory can be rescaled until the drone is capable of hitting all waypoints. This, however, has two disadvantages:

- Rescaling of time traversal breaks the dynamic feasibility of the trajectories,

$$\begin{aligned} x_k &\neq x_k + dt v_k \\ v_k &\neq v_k + dt a_k \end{aligned}$$

- Proportion of time between waypoints cannot be changed,

$$\frac{t_{i+1} - t_i}{t_{j+1} - t_j} = const$$

These issues are addressed through introducing complementary progress constraints (CPC) [1].

2.4.2 Complementary Progress Constraint Method

Instead of satisfying the consecutive tracking of waypoints through timestamped constraints at each time the waypoints should be reached, the idea of complementary progress constraints can be used [1]. This method introduces a measure of progress and complements it with waypoint proximity. More specifically, this method formulates two factors that must complement each other, where one factor is the completion of a waypoint (progress) while the other factor is the local proximity to a waypoint. Intuitively, a waypoint can only be marked as completed when the drone is within a certain tolerance of the waypoint, allowing simultaneous optimization of the state, and input trajectory, and the waypoint time allocation. This is depicted in Figure 3, where the time rescaling used in SCP is shown on the left, while the right shows CPC in which waypoints can be freely assigned as part of the optimization problem.

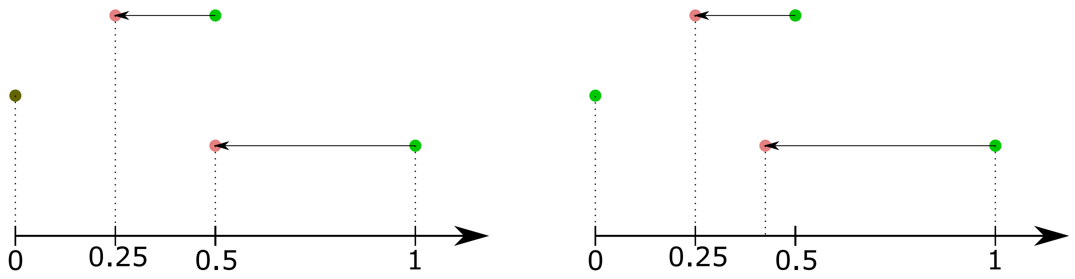

Figure 3: Time optimal waypoint assignment for the naive SCP approach and the CPC approach.

3 Experiments

This section presents an analysis of the two methods described in this paper. A total of three trajectory generation methods are compared on how quickly a drone can traverse the generated trajectory through a specified set of waypoints that represent the racing track. The drone model parameters were taken from [1], which corresponds to a typical racing drone.

The convex optimization problem within the sequential convex programming is configured with the weight matrices $P = \mathbb{I}_{12}$, $Q = \mathbb{I}_3$, $R = 1 \times 10^{-2}\mathbb{I}_4$ and $\bar{Q} = 5 \times 10^{2}\mathbb{I}_3$. The trust region constraints are set to be $\rho_x = 1.0$ and $\rho_u = 0.5$. The waypoint satisfaction constraints have a relaxation variable $r_j = 0.75$. The configuration of the trajectory optimization using CPC is left as default.

In order to obtain the lap time for a given trajectory, a drone is simulated to track the trajectory in minimal time. Hence, the *lap time for a trajectory is taken as the minimal time in which the drone is capable of tracking the trajectory.* The positional reference trajectories generated by each method are given in Figure 4 over time. Correspondingly, since each method reaches the waypoints at different times, the waypoints are also indicated for each method respectively. Additionally, Table 1 summarizes the lap time and path length for each trajectory.

The results would indicate that the trajectory found through CPC has the fastest lap time. However, this is the lap time obtained from the optimization assigned time. Namely, the controller was not able to track the resulting trajectory even after rescaling the time over which the trajectory spans. This is likely due to the optimization technique exhausting the physical capabilities of the drone completely, resulting in an over-aggressive trajectory, which is extremely hard to track. On the other hand, the drone is capable of tracking the SCP trajectory in a minimal time which is almost 22% faster than the baseline spline trajectory. Additionally, it is interesting to note that in terms of path length, the SCP trajectory achieves the shortest path. This is due to the SCP optimization objective containing a term which optimizes over path length as a substitute of optimizing over time.

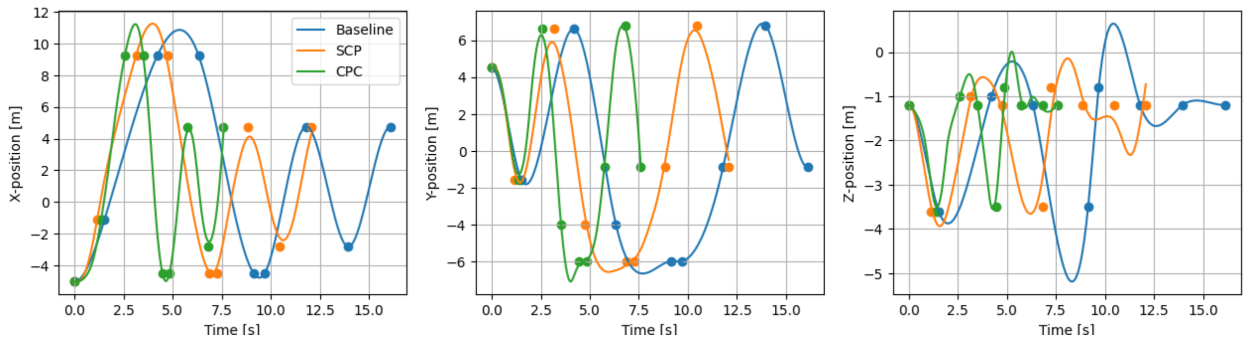

Figure 4: Position trajectories using cubic spline interpolation (blue), Sequential Convex Programming (orange) and Complementary Progress Constraints (green).

	Baseline	SCP	CPC
Lap Time $[s]$	15.5	12.1	7.6*
Path Length $[m]$	86.65	79.53	81.86

Table 1: Summary of path lengths and lap times for the different trajectories. *this does not represent the actual lap time of the drone since it was not capable of tracking the trajectory, but rather the optimization assigned time.

4 Conclusion

This study explores trajectory optimization techniques for high-speed quadcopter drone racing, comparing Sequential Convex Programming (SCP) and Complementary Progress Constraints (CPC). Building on previous work, we implemented a dynamic drone model and a nonlinear tracking controller to optimize flight paths. While CPC showed potential for the shortest lap times, its trajectories were often too aggressive for practical tracking. SCP, however, provided more consistently trackable paths, achieving a 22% improvement in lap time over a baseline method. Our findings highlight the need to balance optimization with the drone's physical capabilities, suggesting SCP as a more reliable approach for practical applications. For future work, different control methods can be considered, such as model predictive control, to improve tracking of aggressive trajectories. Additionally, waypoint loops may be incorporated by including attitude constraints along with position constraints for each waypoint.

GitHub and Presentation Links

GitHub:
`https://github.com/MikeTimmerman-ae/Drone_Trajectory_Optimization`

Presentation Video:
`https://drive.google.com/file/d/1QbYBA1vVwesKcVCzk7ondgnZxKBAAmMq/view`

References

[1] Philipp Foehn, Angel Romero, and Davide Scaramuzza. Time-optimal planning for quadrotor waypoint flight. *Science Robotics*, 6(56):eabh1221, 2021.

[2] Ezra Tal and Sertac Karaman. Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness. *IEEE Transactions on Control Systems Technology*, 29(3):1203–1218, 2021.