Trajectory Optimization for Multirotors

Parker C. Lusk and Jesus Tordesillas
(equally contributed)

***Abstract*— Trajectory optimization is performed for aggressively maneuvering multirotors. Leveraging the differential flatness of multirotors, we formulate the trajectory optimization problem as a mixed-integer quadratic program solved using Gurobi. We implement four different aggressive trajectories: flip in roll, flip in pitch, flip in roll with translation and half-flip with translation. The controller used to track the trajectory consists of an outer loop (with a feedforward term) to track the position and velocity, and an inner loop to track attitude. This framework is extensively tested in simulation and on real hardware, achieving flights through a 90° gate at $8\,\mathrm{m/s}$. Finally, we study and implement an alternative method to explicitly take into account actuator constraints.**

SUPPLEMENTAL MATERIAL

A video of the simulation and hardware testing can be found at https://youtu.be/WnfzMO0pXsc. The code used in this project is available at https://github.com/jtorde/uav_trajectory_optimizer_gurobi.

I. INTRODUCTION AND RELATED WORK

Multirotors are popular aerial platforms for guidance, navigation, and control research. In this paper, we focus on the guidance of these small unmanned aircraft systems (sUAS) via trajectory optimization.

Different approaches have been proposed in the literature to solve the trajectory optimization problem for multirotors. One method uses the full nonlinear dynamics of multirotors, and solve it as an standard optimal control problem, which can be solved via sparse nonlinear programming and hp-adaptive Gaussian quadrature collocation methods [1].

However, most state-of-the-art planners [2]–[7] leverage the differential flatness of multirotors, which gives a one-to-one algebraic mapping between the "flat" output states, and the full system states and inputs [8]. Using this property, trajectory generation can be formulated as a convex optimization problem. Assuming a simple n^{th}-order integrator model, the squared norm of a derivative of the position is minimized to find a dynamically-feasible smooth trajectory [4], [9]. Tracking of the trajectory is then achieved using a low-level tracking controller [6].

The goal of this project is to implement both in simulation and in hardware (see Fig. 1) the trajectory optimization problem required to obtain different aggressive maneuvers. In particular, we will focus on flips through a gate. Moreover, we also study an alternative formulation proposed in [10] to explicitly take into account actuator constraints.

The authors are with the Aerospace Controls Laboratory, MIT, 77 Massachusetts Ave., Cambridge, MA, USA {plusk, jtorde}@mit.edu

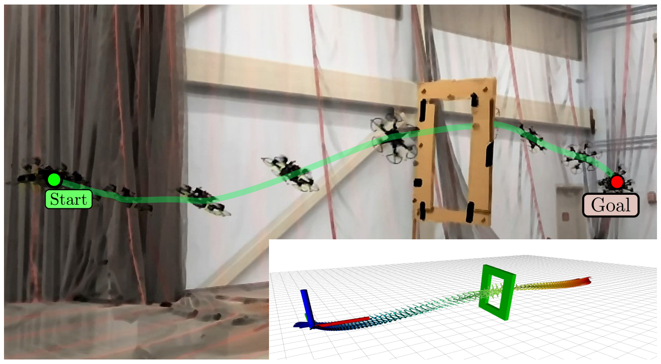


Fig. 1: A hexarotor following the optimized trajectory at $8\,\mathrm{m/s}$ through a gate rotated 90°. Both the hexarotor and the cardboard gate have reflective markers used for the motion capture systems.

In summary, this work presents the following contributions:

- A formulation of the trajectory optimization as a mixed-integer quadratic program (MIQP) and implementation in Python using Gurobi.
- Simulation experiments showing four different aggressive trajectories: *flip in roll*, *flip in roll with translation*, *flip in pitch*, and *half-flip with translation*.
- Hardware experiments for the *half-flip with translation* trajectory, with speeds up to $8\,\mathrm{m/s}$.
- Implementation of an alternative formulation to take into account the actuator constraints.

The rest of this paper is organized as follows. The trajectory optimization problem is posed in Section II. A brief discussion of the tracking controller is found in Section III. Simulation results are given in Section IV-A and hardware results in Section IV-B. In Section V, an alternative formulation is discussed. Finally, the conclusion is given in Section VI.

II. TRAJECTORY GENERATION

Using a triple-integrator model, and assuming a piece-wise constant jerk, the acceleration will be piece-wise linear, the velocity will be a spline of parabolas and the position will be a spline of cubic polynomials. Hence, by minimizing jerk we are minimizing the control cost (or, in in other words, maximizing the smoothness of the trajectory). The design of a sample trajectory with N intervals is shown in Fig. 2. Apart from these dynamic constraints, we have to impose continuity constraints and state/inputs bounds. Finally, to obtain a desired trajectory, we impose several attitude and position constraints that will be detailed later on. In summary,

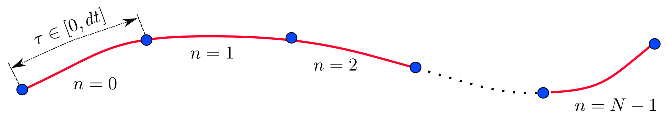


Fig. 2: Each interval $n \in \{0, N-1\}$ of the trajectory is a third degree polynomial, with a total time of dt per interval. $\tau \in [0, dt]$ denotes a local reference of the time inside an interval.

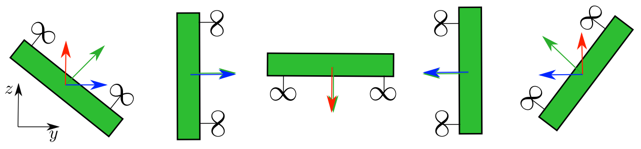


Fig. 3: Orientations imposed to perform a flip in roll. In red, $\boldsymbol{a}_{n,z} + g$, and in blue, $\boldsymbol{a}_{n,y}$

the mathematical program is given by

$$\begin{aligned} \min_{j_{0:N-1}} \; & \sum_{n=0}^{N-1} \|\mathbf{j}_n\|^2 \qquad (1) \\ \text{s.t. } & \mathbf{x}_0(0) = \mathbf{x}_{\text{init}} \\ & \mathbf{x}_{N-1}(dt) = \mathbf{x}_{\text{final}} \\ & \boldsymbol{x}_n(\tau) = \boldsymbol{a}_n\tau^3 + \boldsymbol{b}_n\tau^2 + \boldsymbol{c}_n\tau + \boldsymbol{d}_n \; \forall n, \forall \tau \in [0, dt] \\ & \boldsymbol{v}_n(\tau) = \dot{\boldsymbol{x}}_n(\tau) \; \forall n, \forall \tau \in [0, dt] \\ & \boldsymbol{a}_n(\tau) = \dot{\boldsymbol{v}}_n(\tau) \; \forall n, \forall \tau \in [0, dt] \\ & \mathbf{j}_n = 6\boldsymbol{a}_n(0) \; \forall n \\ & \mathbf{x}_{n+1}(0) = \mathbf{x}_n(dt) \quad n \in \{0, N-2\} \\ & \|\boldsymbol{v}_n(0)\|_\infty \le v_{\max}, \qquad \forall n \\ & \|\boldsymbol{a}_n(0)\|_\infty \le a_{\max} \qquad \forall n \\ & \|\boldsymbol{j}_n\|_\infty \le j_{\max} \qquad \forall n \\ & \text{attitude and position constraints} \end{aligned}$$

In the above optimization problem, dt is the time allocated per interval (see Fig. 2). To ensure that the problem remains convex, we fix this value and use a line search on dt, taking the first solution for which the problem converges.

In the remainder of this section, we outline the trajectory design of four different trajectories.

A. Flip in Roll

In this scenario we consider a gate above the position of the vehicle, and we force the multirotor to do a flip in roll while traversing the gate. The five different stages of the roll maneuver are depicted in Fig. 3. The stages are encoded as the following constraints, where $\{b_{ni}\}_{i=0}^{3}$ are binary decision variables:

$$\begin{aligned} b_{n0} = 1 &\implies \boldsymbol{a}_{n,y} = \boldsymbol{a}_{n,z} + g \; \forall n \\ b_{n1} = 1 &\implies \begin{cases} \boldsymbol{a}_{n,z} + g = 0 \; \forall n \\ \boldsymbol{a}_{n,y} \ge \epsilon_y \; \forall n \end{cases} \\ &\begin{cases} \boldsymbol{a}_{N/2,z} + g \le -\epsilon_z \\ \boldsymbol{a}_{N/2,y} = 0 \\ \boldsymbol{x}_{N/2} = \boldsymbol{x}_{Gate} \end{cases} \\ b_{n2} = 1 &\implies \begin{cases} \boldsymbol{a}_{n,z} + g = 0 \; \forall n \\ \boldsymbol{a}_{n,y} \le -\epsilon_y \; \forall n \end{cases} \\ b_{n3} = 1 &\implies \boldsymbol{a}_{n,y} = -(\boldsymbol{a}_{n,z} + g) \; \forall n \\ &\textstyle\sum_{n=0}^{N-1} b_{ni} = 1 \; \forall i \\ &\textstyle\sum_{i=0}^{3} b_{ni} \le 1 \; \forall n \end{aligned}$$

All the variables in these constraints are evaluated at dt, which is omitted to simplify the notation. In these constraints, g denotes the absolute value of the gravity ($9.81\,\mathrm{m/s}$), and the accelerations a_y and a_z are expressed in the world frame. ϵ_y denotes a strictly positive quantity (but not necessarily small) chosen by the user. The idea behind ϵ_y is to enforce a coupling between the accelerations a_y and a_z to prevent the vehicle from flipping with $a_y(t) = 0 \; \forall t$. Without this coupling, the multirotor could perfectly flip by simply achieving $a_z < -9.81\,\mathrm{m/s}$, which is feasible for this decoupled triple-integrator model, but clearly infeasible in the real multirotor: a flip in roll requires $a_y(t) \neq 0$ for some interval of time (i.e., the normal of the plane of the motors have to be not parallel to the world axis z for some interval of time, see Fig. 3).

Note that with the above binary decision variables, we are allowing the solver to choose in which specific interval of the trajectory each specific orientation is achieved. Moreover, we are not forcing any specific order in these constraints. Therefore, the solver can choose to do a positive-roll flip or negative-roll flip (depending on the initial velocity of the vehicle, for instance).

B. Flip in Roll with translation

Using the same constraints as above, we can also achieve a translation while doing the flip if we impose that $\boldsymbol{x}_{\text{init},x} \neq \boldsymbol{x}_{\text{final},x}$. In this case the gate is placed in the middle between the starting and the end point.

C. Flip in Pitch

In this scenario, the gate is again above the initial position of the multirotor, but this time we force a flip in pitch using the following constraints:

$$\begin{aligned} b_{n0} = 1 &\implies \boldsymbol{a}_{n,x} = \boldsymbol{a}_{n,z} + g \; \forall n \\ b_{n1} = 1 &\implies \begin{cases} \boldsymbol{a}_{n,z} + g = 0 \; \forall n \\ \boldsymbol{a}_{n,x} \ge \epsilon_x \; \forall n \end{cases} \\ &\begin{cases} \boldsymbol{a}_{N/2,z} + g \le -\epsilon_z \\ \boldsymbol{a}_{N/2,x} = 0 \\ \boldsymbol{x}_{N/2} = \boldsymbol{x}_{Gate} \end{cases} \\ b_{n2} = 1 &\implies \begin{cases} \boldsymbol{a}_{n,z} + g = 0 \; \forall n \\ \boldsymbol{a}_{n,x} \le -\epsilon_x \; \forall n \end{cases} \\ b_{n3} = 1 &\implies \boldsymbol{a}_{n,x} = -(\boldsymbol{a}_{n,z} + g) \; \forall n \\ &\textstyle\sum_{n=0}^{N-1} b_{ni} = 1 \; \forall i \\ &\textstyle\sum_{i=0}^{3} b_{ni} \le 1 \; \forall n \end{aligned}$$

Note that these constraints are the same as the ones previously defined in Section II-A, but $\boldsymbol{a}_{n,y}$ is replaced by $\boldsymbol{a}_{n,x}$, and ϵ_y by ϵ_x.

D. Half-flip with translation

We now consider a scenario in which the vehicle has to pass through a gate tilted by 90°. To achieve this, we impose the following constraints:

$$\begin{aligned} &\boldsymbol{a}_{n,z} + g \ge 0 \; \forall n \\ &\begin{cases} \boldsymbol{a}_{N/2,z} + g = 0 \\ \boldsymbol{a}_{N/2,y} \ge \epsilon_y \\ \boldsymbol{x}_{N/2} = \boldsymbol{x}_{Gate} \end{cases} \end{aligned}$$

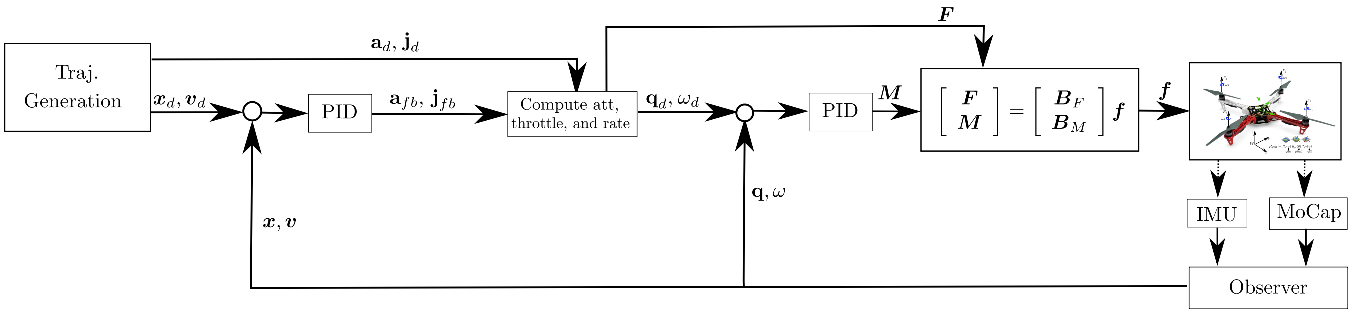

Traj. Generation
$\boldsymbol{x}_d, \boldsymbol{v}_d$
$\mathbf{a}_d, \mathbf{j}_d$
PID
$\mathbf{a}_{fb}, \mathbf{j}_{fb}$
Compute att, throttle, and rate
$\boldsymbol{F}$
$\mathbf{q}_d, \omega_d$
PID
$\boldsymbol{M}$
$\begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{M} \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_F \\ \boldsymbol{B}_M \end{bmatrix} \boldsymbol{f}$
$\boldsymbol{f}$
$\mathbf{q}, \omega$
$\boldsymbol{x}, \boldsymbol{v}$
IMU
MoCap
Observer

Fig. 4: System architecture. The low-level autopilot is capable of tracking desired attitude and body rates, which are generated by the outer loop. The outer loop tracks position and velocity, using acceleration and jerk as feed-forward commands. All of these inputs are generated from the trajectory optimization described in this paper.

In the first constraint we simply impose that the relative acceleration in z has to be always positive (to ensure that the multirotor does not perform a complete flip). In the second set of the constraints, we impose that the vehicle's roll has to be the same as the orientation of the gate. Note that no binary variables are used in this case, since the constraint at $N/2$ is enough to generate the desired trajectory.

For any tilted angle $\alpha \neq 90°$ of the gate, the constraints would be as follows:

$$\begin{cases} \boldsymbol{a}_{n,z} + g \geq 0 \;\forall n \\ (\boldsymbol{a}_{N/2,z} + g)tan(\alpha) = \boldsymbol{a}_{N/2,y} \\ \boldsymbol{a}_{N/2,y} \geq \epsilon_y \\ \boldsymbol{x}_{N/2} = \boldsymbol{x}_{Gate} \end{cases}$$

III. TRAJECTORY TRACKING

To track this trajectory, we use the control diagram shown in Figure 4. In this control scheme, we distinguish fwour main blocks:

- **Trajectory Generator**: Here the optimization problem explained in Section II is solved, producing the desired position, velocity, acceleration, and jerk.
- **Outer Loop**: Using a PID loop, feedback acceleration is computed to track position and velocity. Combining this feedback acceleration with the desired acceleration from the trajectory generator as a feed-forward term, the desired attitude is computed. The total commanded acceleration is also used to calculate the desired thrust.
- **Inner Loop**: Given the desired jerk from the trajectory generator and feedback jerk numerically differentiated from feedback acceleration, the desired body rates $\boldsymbol{\omega}_{\text{des}}$ are computed. Then, given the desired attitude from the outer loop, this block calculates the moments $\boldsymbol{M}$ using a quaternion-based PID controller expressed as

$$\boldsymbol{M} = \text{sign}(q_w) K_p \vec{\boldsymbol{q}}_e + K_d(\boldsymbol{\omega}_{\text{des}} - \boldsymbol{\omega}), \quad (2)$$

  where $\boldsymbol{q}_e = \boldsymbol{q}^* \otimes \boldsymbol{q}_{\text{des}}$ and $\vec{\boldsymbol{q}}_e = \begin{bmatrix} q_{e,x} & q_{e,y} & q_{e,z} \end{bmatrix}^\top$ is the vector (imaginary) part of the quaternion. Using the motor allocation matrix, the actuator commands $\boldsymbol{f}$ are recovered from the desired thrust and moments.
- **Observer**: Position, velocity, attitude, and IMU biases are estimated by fusing propagated IMU measurements with an external motion capture system via an extended Kalman filter.

IV. RESULTS

We solve the optimization problem using Gurobi [11], a commercial solver that solves convex problems (LPs, QPs, and QCPs), and that also allows the use of binary variables (i.e., it also solves MILPs, MIQPs, and MIQCPs). We use the Python interface to encode the optimization problem. Additionally, we leverage ROS for communication between the different algorithm components, which enables both simulation and hardware testing with minimal changes.

A. Simulation

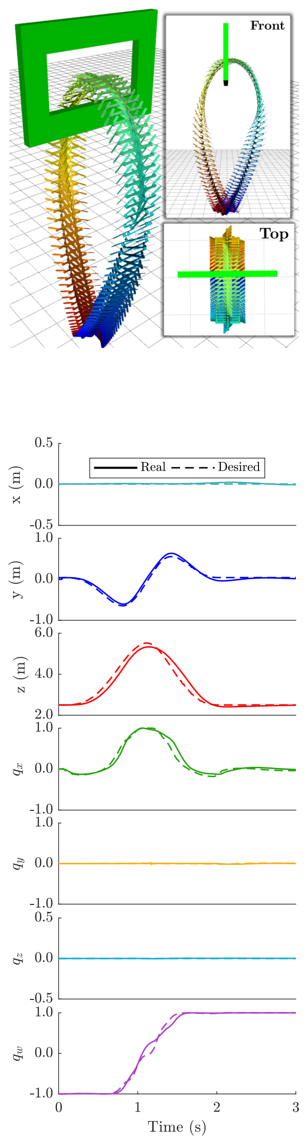
For the simulation, we first generate the trajectory with the aforementioned optimization problem, and then track it using the controller diagrammed in Figure 4. The full nonlinear dynamics of the multirotor are simulated in C++ by numerically integrating the associated differential equations using a 4^{th}-order Runge–Kutta solver. We visualize the trajectory using RViz and Gazebo.

The 3D plot of the trajectory, together with the position and orientation tracking errors of all the trajectories explained previously are shown in Figure 5. As it can be seen, the multirotor always passes through the gate and a reasonably low tracking error in position is achieved for all of the trajectories. We also have low attitude tracking error in the *flip in roll* and *flip in pitch* trajectories, although this error is higher for the *flip in roll with translation* and *half-flip with translation* trajectories. One of the reasons for this is that the controller struggles more when it has to do a flip (or a half-flip) at the same time as it is translating: specially in situations in which the plane of the motors is parallel to the direction of the translation, the multirotor instantaneously "loses control" on that direction. We also believe that more tuning of the inner and outer loop controllers of the simulation would help to reduce this error.

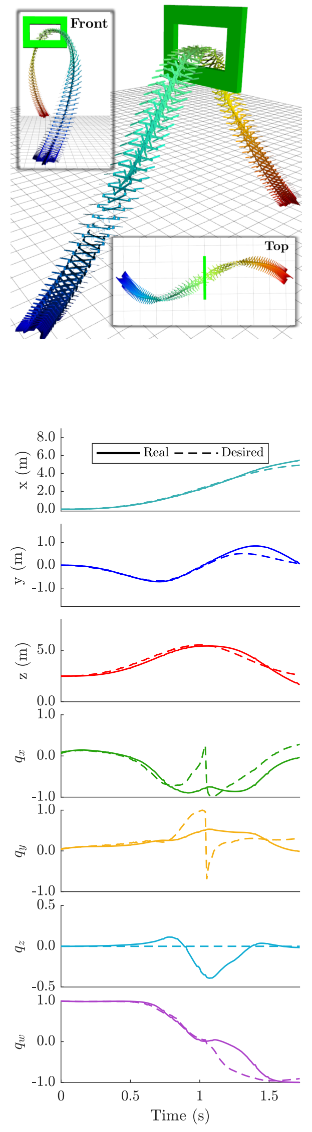
Refer to the supplemental material for a video of the simulation.

B. Hardware

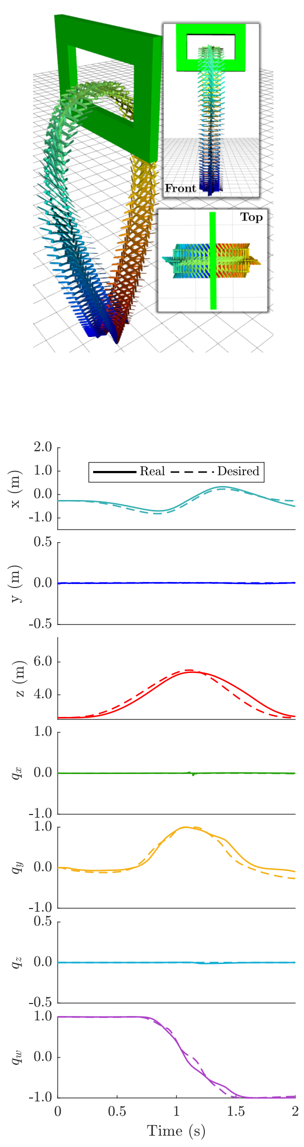
We test the trajectory generation algorithm in hardware using a hexarotor in the Aerospace Controls Laboratory flight space. The flight space (shown in Figure 1) is equipped with two brands of motion capture systems: VICON for the half where the vehicle starts and OptiTrack for the half where the vehicle ends. This requires combining the two



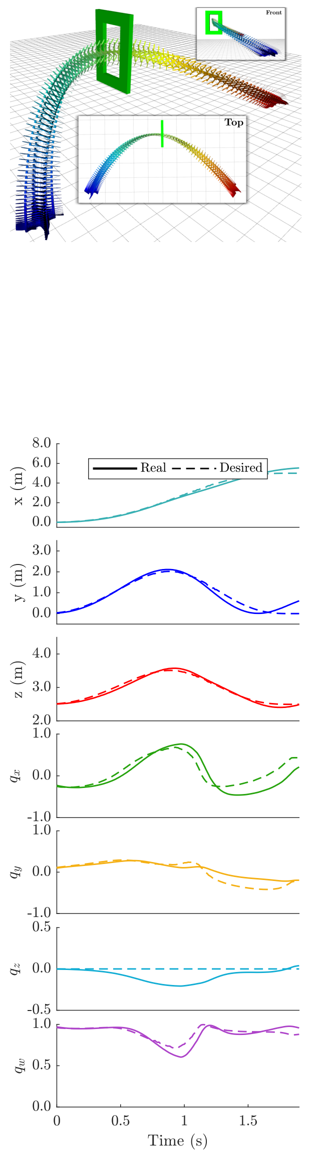
(a) Flip in roll



(b) Flip in roll with translation



(c) Flip in pitch



(d) Half-flip with translation

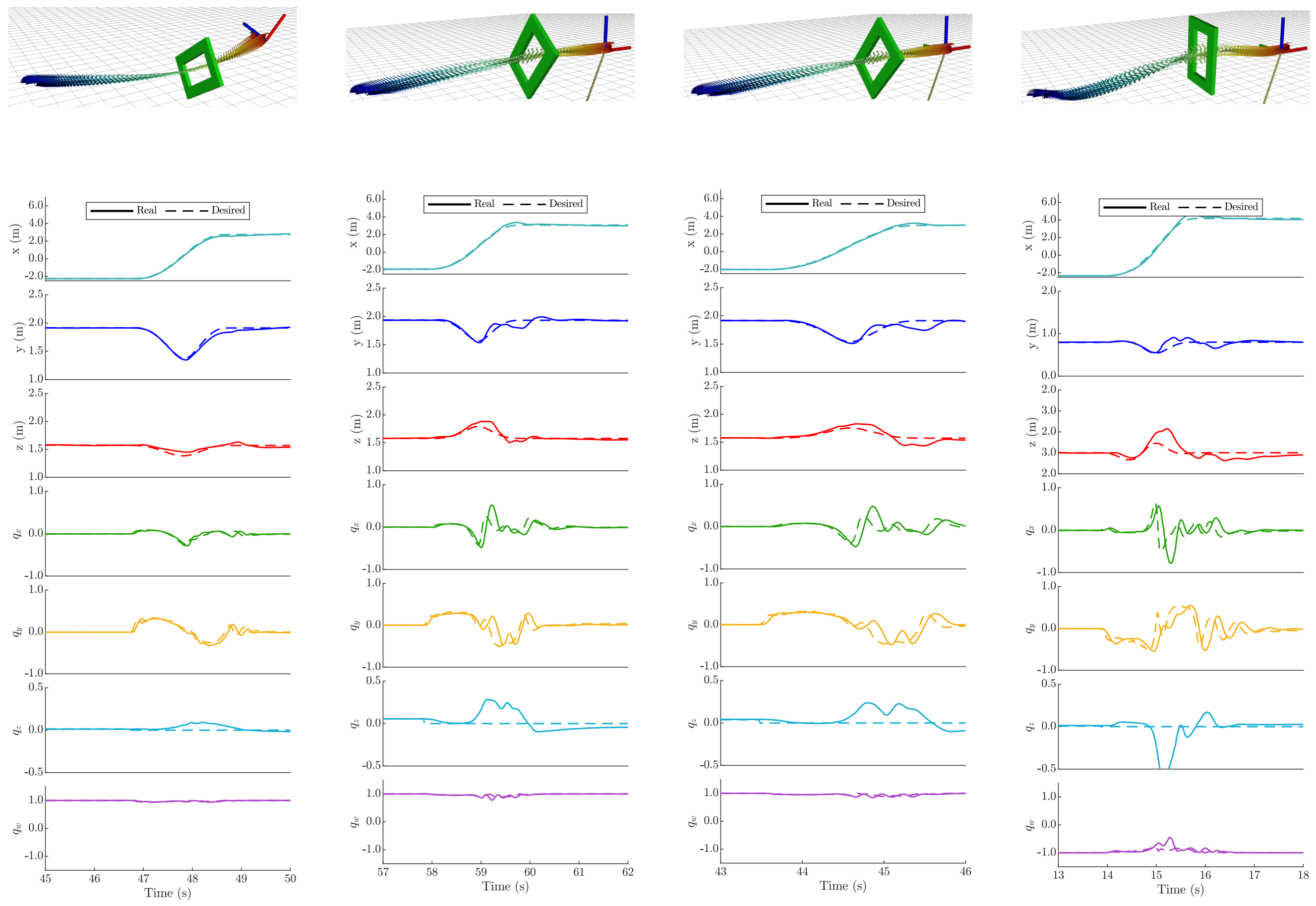
Fig. 5: Simulation results of the various trajectories discussed in Section II. The vehicle starts at the blue marker and ends at the red marker. The tracking performance of each trajectory is also shown.

systems with a static frame transformation; however, due to the warping of camera systems (and in particular, at the edges of two separate systems), using a static transformation is not accurate for all points between the systems.

Specifically, we test the *half-flip with translation* trajectory through a gate (see Fig. 1) at various orientations and opening sizes. The hexarotor (shown in Fig. 7) is equipped with a Snapdragon Flight board which runs the outer and inner loop control. The trajectory optimization piece is performed off-board at the beginning of each test.

The results of these tests are found in Fig. 6. In the error plots, we can see that the trajectories are generally tracked well, but there is more error than in simulation. We think of four possible causes for this, starting with the most flagrant:

- In the formulation presented so far, we are not taking explicitly into account actuator constraints of individual motors. After analyzing the logs of the flights, we noticed that in some maneuvers motors were saturating. This nonlinear phenomenon is clearly undesirable and could be remedied by lowering the state bounds in the optimization problem (1). An alternative approach to solve this is presented in Section V.
- The flip happens at the intersection between the VICON and OptiTrack camera systems. Due to the error in fitting a static transform between the two systems, a state jump is introduced as the multirotor flies between these two systems. Further, we have observed that OptiTrack is in general more finicky than VICON, requiring frequent calibration and more thoughtful placement of reflective markers.
- A higher-order model (i.e., minimizing snap) would help reduce the jerk and create a more mild trajectory.



(a) 30° and wide (b) 45° and wide (c) 45° and narrow (d) 90° and narrow

Fig. 6: Hardware flight results of the *half-flip with translation* trajectory. The orientation and opening size of the gate is varied. The tracking performance of each trajectory is also shown. The maximum speed achieved is about $8\,\mathrm{m/s}$.

- As with any controller, more time tuning the gains on the outer and inner loop would help reject the specific disturbances caused by the designed maneuver.

Refer to the supplemental material for a video of the flight tests.

V. Alternative Formulation

In contrast to a numerical optimization for minimum-jerk trajectory generation, Cutler and How [10] propose a method that utilizes 9th order polynomials to achieve a closed-form solution to minimum-snap trajectories. Actuator constraints are respected by iteratively solving this closed-form expression while minimizing the time of the total trajectory. We implemented this approach in MATLAB. To track the trajectory, we also implemented the quaternion-based attitude controller proposed in [10], which relies on the differential flatness of multirotors.

A. Dynamic Model

The nonlinear dynamics of a multirotor are modeled as a rigid body in $\mathbb{R}^3 \times \mathrm{SU}(2)$. We use the unit quaternion attitude representation to capture the nonlinear nature of rotations. Thus, the equations of motion (in an ENU inertial frame, FLU body frame) are given by

$$\begin{bmatrix} 0 \\ \ddot{\mathbf{r}}^i \end{bmatrix} = \frac{1}{m}\mathbf{q}^* \otimes \begin{bmatrix} 0 \\ \mathbf{F}^b \end{bmatrix} \otimes \mathbf{q} - \begin{bmatrix} 0 \\ \mathbf{g}^i \end{bmatrix} \tag{3}$$

$$\dot{\boldsymbol{\omega}}^b = \mathbf{J}^{-1}\left[\mathbf{M}^b - \boldsymbol{\omega}^b \times \mathbf{J}\boldsymbol{\omega}^b\right], \tag{4}$$

where $\mathbf{F}^b = \begin{bmatrix} 0 & 0 & f_{\text{total}} \end{bmatrix}^\top$, f_{total} is the thrust, aligned with the z-axis due to motor placement, and $\mathbf{M}^b$ are the moments induced on the body by the actuators.

B. Trajectory Generation

A piecewise-smooth trajectory is generated for the multirotor to follow. As is popular in other work [9], a polynomial representation of the trajectory position is used, which makes specifying constraints on the derivatives of position straightforward. For a 9th-order polynomial with n waypoints ($n-1$ segments), the position trajectory is defined as

$$\mathbf{r}_d(t) = \begin{cases} \sum_{i=0}^{9} \alpha_{i,1} t^i & 0 \le t < t_1 \\ \sum_{i=0}^{9} \alpha_{i,2} t^i & t_1 \le t < t_2 \\ \vdots & \vdots \\ \sum_{i=0}^{9} \alpha_{i,n-1} t^i & t_{n-2} \le t \le t_{n-1} \end{cases} .$$

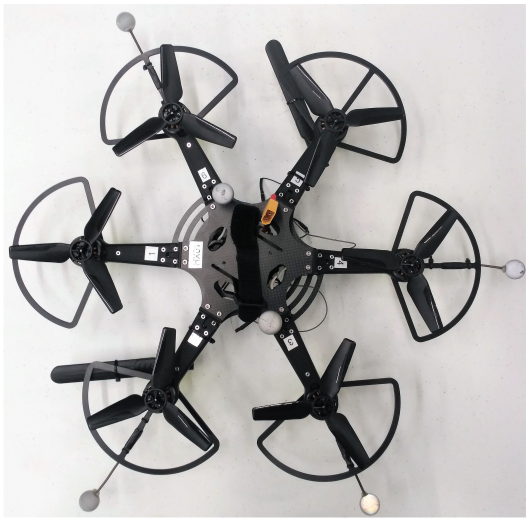


Fig. 7: Custom hexarotor used in the hardware experiments. It is equipped with the Qualcomm® Snapdragon Flight, which is where the controller and the observer run.

With a desired initial and final position, fixed times $\{t_i\}_{i=1}^{n-1}$ and continuity constraints on the derivatives of $\mathbf{r}_d$, a linear system can be constructed to solve for the coefficients $\alpha_{i,j}$.

C. Actuator Constraints

The primary factor in the magnitude of the actuator commands is the segment endpoint times $\{t_i\}_{i=1}^{n-1}$. By performing a line search on the endpoint times, actuator constraints can be enforced. The mapping between the optimized trajectory $\mathbf{r}_d$ and actuator commands is found via differential flatness, as outlined in [9], [12].

D. Results

To test our implementation of [10], we generate a two-segment trajectory from $\mathbf{x}(0) = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^\top$, through $\mathbf{x}(t_1) = \begin{bmatrix}\times & \times & -2\end{bmatrix}^\top$, to $\mathbf{x}(t_2) = \begin{bmatrix}6 & 0 & 0\end{bmatrix}^\top$, as shown in Fig. 8. Note that at time t_1 we only impose a constraint on the z-position, allowing the other positions to be free. The times are initially set to $t_0 = 0$, $t_1 = 0.5$, $t_2 = 2$.

The tracking of a minimum-time, actuator-constrained trajectory is shown in Fig. 9. The trajectory is optimized after 12 iterations. If the optimization is stopped early at 6 iterations, the actuator constraints are violated, as shown in Fig. 10. Note that on a real system, the tracking would not be as good since the actuators would saturate.

VI. CONCLUSION AND FUTURE WORK

This work presented a trajectory optimization framework to obtain aggressive trajectories for multirotors. By using the differential flatness of multirotors, a MIQP was formulated to obtain a desired trajectory, and then a controller was used to track it. This framework was successfully tested both in simulation and in hardware experiments, achieving aggressive flights through a gate with speeds up to $8\,\mathrm{m/s}$. Finally, an alternative formulation to explicitly take into account actuator constraints was studied and implemented in simulation.

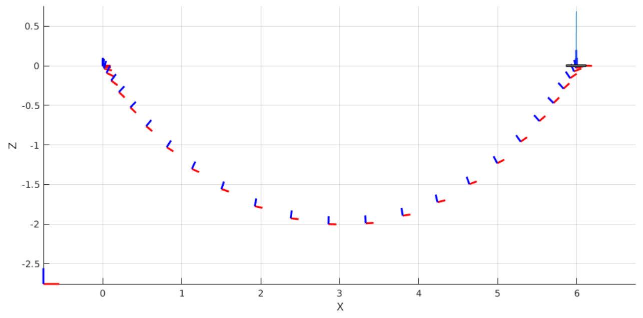


Fig. 8: A trajectory generated using the MATLAB implementation of the alternative method. Constraints are chosen so that the z-position of the multirotor moves through -2. A line search is performed on time for a minimum-time trajectory that respects actuator constraints.

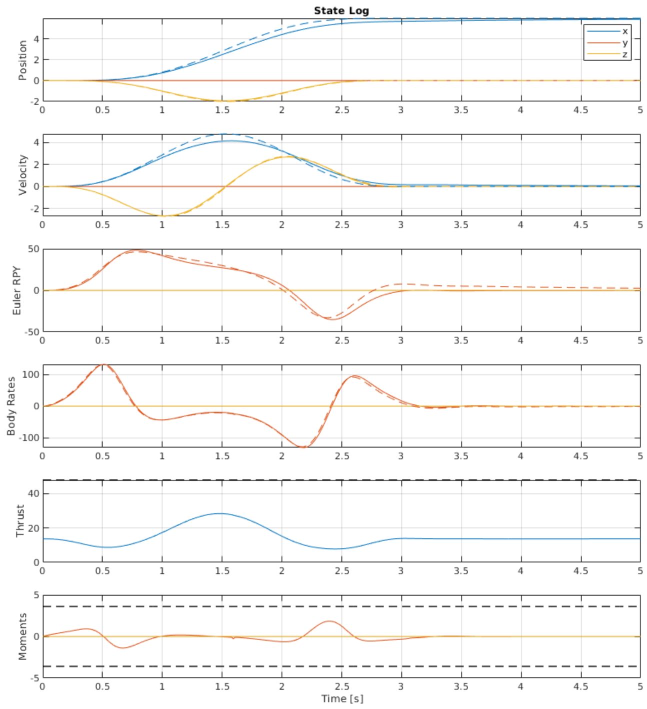


Fig. 9: Tracking of the optimized trajectory after 12 iterations. Final times are chosen to be $t_0 = 0$, $t_1 = 1.6$, $t_2 = 3.1$. The actuator constraints are mapped into forces and moments and shown in dashed black lines.

Given the promising results obtained both in simulation and in hardware experiments, there are several possible future directions. On one hand, in the proposed approach we have assumed that the controller is able to maintain the multirotor near the desired trajectory, but strictly speaking, there are no performance guarantees. Therefore, we think one interesting future work is to apply *tube-based MPC* [13], [14] to guarantee that the vehicle is inside a tube around a nominal trajectory. This strategy is similar to the idea of funnels and LQR-trees [15]. Moreover, for the hardware experiments we have used the accurate full-state information (position, velocity, orientation, and angular rates) provided by the motion capture systems. Hence, one possible future

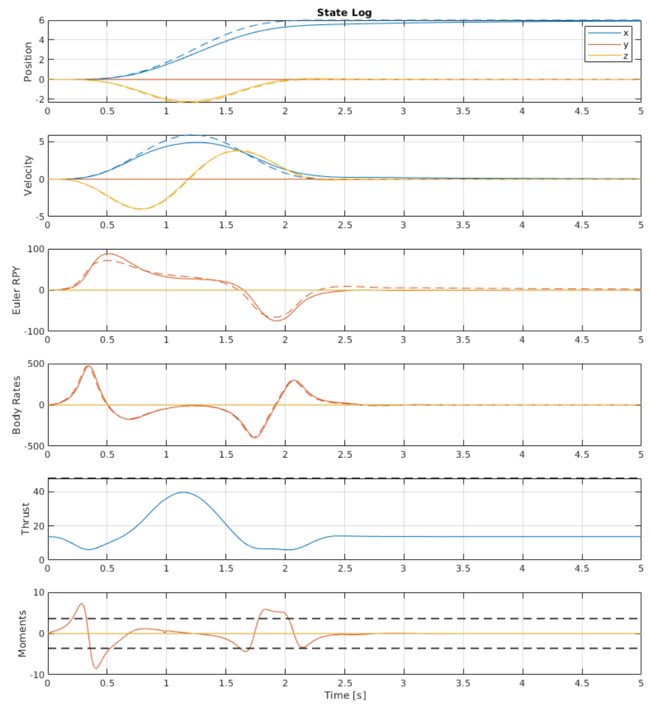


Fig. 10: Tracking of the optimized trajectory stopped early after 6 iterations. Final times are $t_0 = 0$, $t_1 = 1$, $t_2 = 2.5$. The actuator constraints are violated due to attitude commands.

work is to use on-board vision-based state estimation to overcome this limitation. The planner would then generate a more (or less) conservative trajectory by taking into account the uncertainty of the state estimation.

ACKNOWLEDGMENT

Thanks to Russ Tedrake, Wei Gao, and Yunzhu Li for valuable discussions and suggestions related to this project and for the Underactuated Robotics course.

REFERENCES

[1] M. A. Patterson and A. V. Rao, “GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, p. 1, 2014.

[2] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, “Continuous-time trajectory optimization for online uav replanning,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5332–5339, IEEE, 2016.

[3] M. Burri, H. Oleynikova, , M. W. Achtelik, and R. Siegwart, “Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments,” in *Intelligent Robots and Systems (IROS 2015), 2015 IEEE/RSJ International Conference on*, Sept 2015.

[4] C. Richter, A. Bry, and N. Roy, “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,” in *Robotics Research*, pp. 649–666, Springer, 2016.

[5] A. Bry, C. Richter, A. Bachrach, and N. Roy, “Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments,” *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 969–1002, 2015.

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

[7] J. Tordesillas, B. T. Lopez, and J. P. How, “Fastrap: Fast and safe trajectory planner for flights in unknown environments,” *arXiv preprint arXiv:1903.03558*, 2019.

[8] M. J. Van Nieuwstadt and R. M. Murray, “Real-time trajectory generation for differentially flat systems,” *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 8, no. 11, pp. 995–1020, 1998.

[9] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2520–2525, IEEE, 2011.

[10] M. Cutler and J. How, “Actuator constrained trajectory generation and control for variable-pitch quadrotors,” in *AIAA Guidance, Navigation, and Control Conference*, 2012.

[11] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2018.

[12] M. Faessler, A. Franchi, and D. Scaramuzza, “Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories,” *IEEE Robotics and Automation Letters*, vol. 3, pp. 620–626, April 2018.

[13] D. Q. Mayne, E. C. Kerrigan, E. Van Wyk, and P. Falugi, “Tube-based robust nonlinear model predictive control,” *International Journal of Robust and Nonlinear Control*, vol. 21, no. 11, pp. 1341–1353, 2011.

[14] P. Falugi and D. Q. Mayne, “Getting robustness against unstructured uncertainty: a tube-based mpc approach,” *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1290–1295, 2014.

[15] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, “Lqr-trees: Feedback motion planning via sums-of-squares verification,” *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.