Airways: Optimization-Based Planning of Quadrotor Trajectories according to High-Level User Goals

Christoph Gebhardt[†], Benjamin Hepp[†], Tobias Nägeli, Stefan Stevšić, Otmar Hilliges
Departement of Computer Science, AIT Lab
ETH Zürich

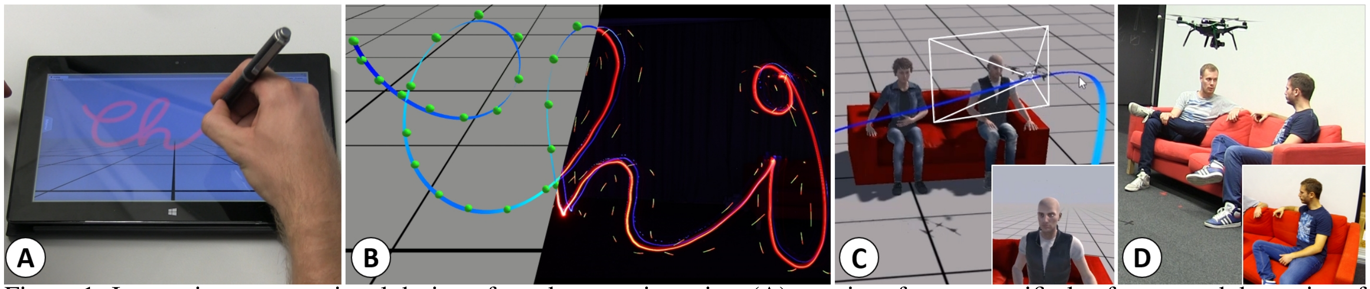

Figure 1: Interactive computational design of quadrotor trajectories: (A) user interface to specifiy keyframes and dynamics of quadrotor flight. (B) An optimization algorithm generates feasible trajectories and (C) a 3D preview allows the user to quickly iterate on them. (D) The final motion plan can be flown by real quadrotors. The tool enables the implementation of a number of compelling use cases such as (B) robotic light-painting, aerial racing and (D) aerial videography.

ABSTRACT

In this paper we propose a computational design tool that allows end-users to create advanced quadrotor trajectories with a variety of application scenarios in mind. Our algorithm allows novice users to create quadrotor based use-cases without requiring deep knowledge in either quadrotor control or the underlying constraints of the target domain. To achieve this goal we propose an optimization-based method that generates feasible trajectories which can be flown in the real world. Furthermore, the method incorporates high-level human objectives into the planning of flight trajectories. An easy to use 3D design tool allows for quick specification and editing of trajectories as well as for intuitive exploration of the resulting solution space. We demonstrate the utility of our approach in several real-world application scenarios, including aerial-videography, robotic light-painting and drone racing.

Author Keywords
robotics; quadrotors; computational design; videography

ACM Classification Keywords
I.2.9 Robotics: Autonomous vehicles; Operator interfaces; H.5.2 User Interfaces;

[†]The first two authors contributed equally to this work.

INTRODUCTION

In recent years micro-aerial vehicles (MAVs), in particular Quadrotors, have seen a rapid increase in popularity both in research and the consumer mainstream. While the underlying mechatronics and control aspects are complex, the recent emergence of simple to use hardware and easy programmable software platforms has opened the door to widespread adoption and enthusiasts have embraced MAVs such as the AR.Drone or DJI Phantom in many compelling scenarios including aerial photo- and videography. Furthermore, the HCI community has begun to explore these drones in interactive systems such as sports assistants [11, 25, 28] or display [30] of content.

Clearly there is a desire to use such platforms in a variety of application scenarios. Current SDKs already give novices access to manual or waypoint based control of MAVs, shielding them from the underlying complexities. However, this simplicity comes at the cost of flexibility. For instance, flying a smooth, spline-like trajectory or aggressive flight maneuvers, for example to create an aerial light show (e.g., [2, 3]), is tedious or impossible with waypoint based navigation. These limits exist because manufacturers place hard thresholds on the dynamics to ensure flight stability for inexperienced pilots.

More importantly, state-of-the-art technologies offer only very limited support for users who want to employ MAVs to reach a certain high-level goal. This is maybe best illustrated by the most successful application area – that of aerial videography. What a few years ago was limited to professional camera crews, requiring cost-intensive equipment like a helicopter, can now, in principle, be done by end-users with a MAV and an action camera. However, producing high-quality aerial footage is not an easy task – it demands attention to the creative aspects of videography such as frame composition and camera motion (cf. [22]). In the case of airborne cameras, an operator needs to fly smoothly, accurately and safely around a camera target. Furthermore, the target has to be framed properly alongside further creative considerations. Thus this is a difficult task and typically requires at least two experienced operators – one pilot and a camera man (cf. [5]). Our method tackles this problem by enabling a single novice user to fly challenging trajectories and still create aesthetically pleasing aerial footage.

Overview & Contribution

Embracing the above challenges we propose a computational method that enables novice end-users to create quadrotor use-cases without requiring expertise in *either* low-level quadrotor control *or* specific knowledge in the target domain. The core contribution of our method is an optimization-based solution that generates *feasible trajectories* for flying robots while taking *high-level user goals* such as visually pleasing video shots, optimal racing trajectories or aesthetically pleasing MAV motion into consideration. Furthermore, we develop an easy-to-use tool that allows for straightforward specification of flight trajectories and high-level constraints. Our approach guides the users in exploring the resulting design space via a 3D user interface and allows for quick iteration until finding a solution which fits best with the user's intentions.

We demonstrate the flexibility of our approach in three real-world scenarios including aesthetically pleasing aerial-videos, robotic light-painting and drone racing.

RELATED WORK

MAVs in HCI

With MAVs becoming consumerized the HCI community has begun to explore this design space. FollowMe [27] is a MAV that follows a user and detects simple gestures via a depth camera, whereas others have proposed using head motion for MAV control [10], while [30] propose a simple, remote controlled flying projection platform. Several setups have been proposed that turn such MAVs into flying, personal companions. For example, to act as jogging partner [25] or general purpose sports coach [11], or as an actuated and programmable piece of sports equipment [28].

Commercially available drones, targeted at the consumer market, shield the user form low-level flight aspects and provide simple manual control (e.g. using smartphones as controller) or waypoint based programmatic navigation as well as GPS based person following. This dramatically lowers the entry barrier for novices but also limits the ceiling of achievable robotic behavior. Our approach also aims for simplicity but gives more power to the users, enabling even novices to design and implement complex flight trajectories, concentrating on the high-level goals of the application domain.

Video Stabilization & Camera Path Planning

Improving the visual quality of end-user produced content is a goal we share with post production video stabilization. Inspired by early work which formulates the problem and discusses the aesthetics of cinematography [8] a number of approaches employ computer vision methods to estimate the original, jerky camera path. Based on this a new, smooth path is computed to generate stabilized video [9, 18] and even time-lapse footage [14] from the source material. Camera path planning has also been studied extensively in the context of virtual environments using constraint based [4, 33] or probabilistic [16] methods. However, these methods are not limited by real-world physics and hence can produce arbitrary camera trajectories and viewpoints. Our approach differs from the above as we propose a *forward* method that gives the user full control over the creative aspect of camera planning while simultaneously optimizing for physical feasibility of the flight path and cinematographic objectives. A 3D simulation lets the user explore the design space before flying the actual trajectory and hence helps in understanding the trade-offs to consider.

Computational Design

Sharing the goal of unlocking areas that previously required significant domain knowledge to novice users, the HCI and graphics communities have proposed several methods that give novice users control over aesthetic considerations while achieving functionality. Recent examples include digitally designed gliders [32] and kites [21] with optimized aerodynamic properties. At the core of these approaches are sophisticated simulations or analytical models of the problem domain that carefully balance accuracy and rapid responses to ensure interactivity while maintaining guarantees (e.g., physical stability). We build on domain knowledge from the robotics and MAV literature and propose an interactive design tool for complex MAV behavior usable by non-experts.

Robotic Behavior and Trajectory Generation

Automating the design of robotic systems based on high-level functional specifications is a long-standing goal in robotics, graphics and HCI. Focussing on robot behavior only, simple direct touch and tangible UIs [34], and sketch based interfaces to program robotic systems [17, 29] have been proposed. Visual markers have been used to control robots explicitly, for example as kitchen aides [31], or implicitly [13, 35], to schedule tasks for robots in-situ which are then collected and executed asynchronously.

Generating flight trajectories for MAVs is well-studied in robotics. In particular, the control aspects of aggressive and acrobatic flight is an active area of research (e.g., [19]). Mellinger et al.'s work on generating minimum snap trajectories [24] is the most related to ours. While they specify a trajectory as a piecewise polynomial spline between keyframes, we discretize the trajectory into small piecewise linear steps. The result is a more intuitive formulation of the optimization problem, making the incorporation of additional constraints and objectives much easier. We extend the approach in [24] by optimizing trajectories for flyability *and* for high-level human objectives. We place the users in the loop and provide easy-to-use tools to design quadrotor trajectories according to high-level objectives.

Joubert et al. [12] share a similar goal in proposing a design tool that allows novice users to specify a camera trajectory, simulate the result, and execute the motion plan. In contrast to our work, their method does not automate feasibility checking but delegates the correction of violations to the user. Furthermore, our method allows to treat keyframes as soft instead

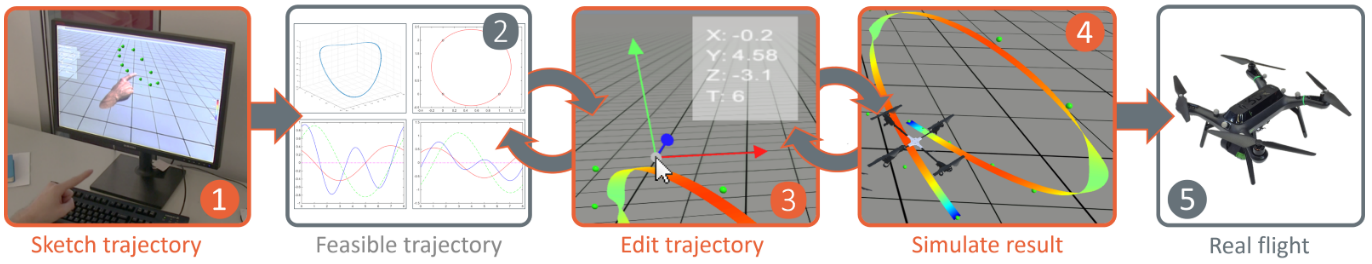

Figure 2: System workflow schematically. (1) User sketches keyframes. (2) An optimization method generates a feasible trajectory. (3+4) The user can quickly iterate over the trajectory and explore the solution space of feasible trajectories via a physics simulation or a rendered preview (see Fig. 3, D). (5) Final trajectory can be flown with a real MAV.

of hard constraints, allowing to trade off feasibility against keyframe matching. We also incorporate a larger number of high-level user constraints, such as additional cinematographic goals and collision-free trajectories, into the algorithm – requiring a different formulation of the optimization problem. Finally, we demonstrate the gain in generality in our approach in the additional use cases of light writing and aerial racing.

SYSTEM OVERVIEW

We propose an end-to-end system that allows users to generate motion plans for quadrotors that are 'flyable' and adhere to high-level human-specified objectives for a variety of application scenarios. Fig. 2 illustrates the design process.

Using for example a LeapMotion controller the user specifies keyframes, each consisting of a position and a time (1). The optimization algorithm generates an initial 'flyable' trajectory from these inputs, i.e., one that lies within the physical capabilities of the underlying quadrotor hardware (2). The method aims to find a solution that goes through all specified keyframes, however the user may now adjust both a keyframe's position and timing as well as other parameters such as the overall flight time, the optimization's objective (e.g., minimization of velocity) or the extend to which the generated trajectory should follow the optimization's objective versus the position of the specified keyframes (3). This can result in trajectories that do not directly meet the user inputs but are the best trade-off between the potentially conflicting use-case specific constraints. A built-in physics simulation allows the user to virtually fly the quadrotor and thus provides a better understanding of the expected real-world behavior and enables rapid iteration of trajectories (4). This tool already enables the design of various flight-maneuvers for example designing an aerial race-course or a light-show (please see video for an illustration of the design process and results).

However, the goal of our work is to enable more complex end-user scenarios as well. To this end we have extended our method to also integrate high-level aesthetic constraints that are not necessarily directly associated with the basics of quadrotor control. Fig. 3 illustrates how our tool can be used to plan aerial videography shots. In this case, the user designs an initial camera trajectory around one or several targets. In addition to the keyframes the user specifies targets which shall be captured by the on-board camera (Fig. 3, B). Our algorithm generates both a quadrotor trajectory and a gimbal trajectory within the physical bounds. To acquire visually pleasing footage our method incorporates cinematographic constraints such as smooth camera and target motion, smooth transitions between multiple targets and reduction of perspective distortions. Furthermore, the algorithm takes obstacle information into account and automatically routes the trajectory through free-space (see Fig. 3, C). It would also be straightforward to integrate other constraints such as limits of the coverage of a tracking system or government flight regulations.

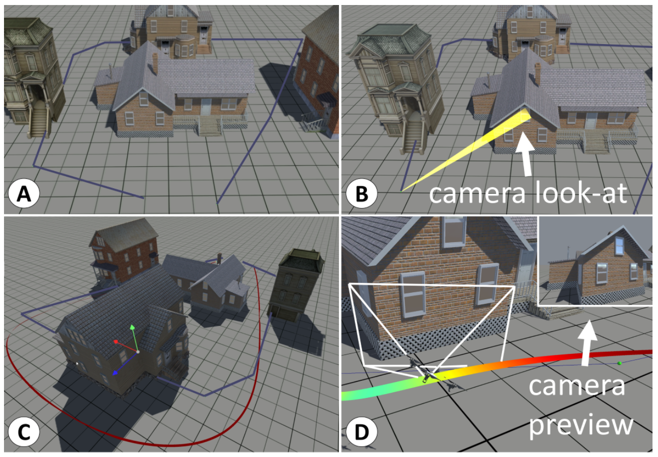

Figure 3: Planning of aerial video shots. (A) User specifies sparse keyframe positions connected by straight lines (purple). (B) For each keyframe the user also specifies a desired camera target (yellow). (C) We generate a smooth and collision free motion plan alongside gimbal control inputs (dark red). (D) Virtual preview allows for rapid prototyping showing the current camera frustum and a camera preview.

In order to better understand the implications of the camera planning our tool allows the user to virtually fly the shot by dragging the virtual quadrotor along the trajectory (Fig. 3, D). For each point in time the tool renders the scene as it would be captured in reality. The user can then edit the plan and iterate over different alternatives. Once satisfied the generated trajectories for quadrotor and gimbal can be deployed as a reference to be followed by a real quadrotor.

METHOD

So far we have discussed the proposed design tool at a high-level and focused on how the user accomplishes certain tasks.

We now introduce the underlying method we use to generate trajectories. To be able to reason about flight plans computationally, a model of the quadrotor and its dynamics are needed. This is a complex and challenging topic and we refer the reader to the Appendix for the full non-linear model that is needed to control the position and dynamics of the robot during flight (please see also [20]). The full model directly relates the inputs of a quadrotor to its dynamics – this however makes trajectory generation a challenging problem and integrating such a highly non-linear model into an optimization scheme is complicated, incurs high computational cost and negates convergence guarantees [24]. However, for most application scenarios considered here a full non-linear treatment is not necessary as demonstrated by our results. In particular if the goal is to generate trajectories only (i.e., position and velocities) rather than the full control inputs as in [24]. Therefore, we present a linear approximative model of the quadrotor and detail the optimization-based algorithm based on it.

Approximate Quadrotor Model for Trajectory Generation

When generating a trajectory we want to ensure that it can be followed by a quadrotor, i.e. a flight plan where each specified position and velocity can be reached within the time limits without exceeding the limits of the qudrotors inputs.

Therefore, we chose to approximate the quadrotor as a rigid body, described by its moment of inertia only along the world frame z-axis (i.e. we ignore pitch and roll of the quadrotor):

$$\begin{aligned} m\ddot{\mathbf{r}} &= \mathbf{F} + m\mathbf{g} \in \mathbb{R}^3 \qquad (1) \\ I_\psi \ddot{\psi} &= M_\psi \in \mathbb{R}, \end{aligned}$$

where $\mathbf{r}$ describes the center of mass of the quadrotor, ψ is the yaw angle of the quadrotor, m is the mass of the quadrotor, I_ψ is the moment of inertia about the body-frame z-axis, $\mathbf{u}_r$ is the the force acting on the quadrotor and M_ψ is the torque along the z-axis. This approximation allows to generate trajectories in the *flat output* space of the full quadrotor model (see Appendix for more details).

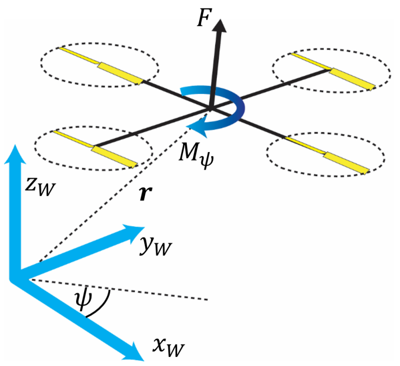

Figure 4: Our approximated quadrotor model with position $\mathbf{r}$, yaw angle ψ), world frame ($\mathbf{x}_W$, $\mathbf{y}_W$, $\mathbf{z}_W$), the moment acting on the quadrotor along the world frame z-axis M_{yaw} and the force F acting on the center of mass of the quadrotor.

In addition to the equations of motion we introduce bounds on the maximum achievable force and torque:

$$\mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max} \in \mathbb{R}^4, \qquad (2)$$

where $\mathbf{u} = [\mathbf{F}, M_\psi]^T$ is the input of the system.

With this model it is not possible to exploit the full dynamic agility of a quadrotor. As an example, consider the situation of accelerating straight upwards by rotating all motors at maximum speed. To now also rotate around the body-frame z-axis we would have to lower the speed of motors 2 and 4, reducing the total thrust of the quadrotor. Currently we do not incorporate this coupling between the translational and rotational dynamics into the bounds Eq. (2) of our approximate linear model Eq. (1). Therefore, to still ensure that a quadrotor can follow trajectories generated on base of this approximation conservative bounds are required. Nonetheless, our results and applications demonstrate that these bounds still allow the quadrotor's agility to be sufficiently rich for many use cases. We refer the interested reader to the Appendix for details on how to choose these bounds.

For trajectory generation we rewrite the approximate model as a first-order dynamical system and discretize it in time with a time-step Δt assuming a zero-order hold strategy, i.e. keeping the inputs constant in between stages:

$$\mathbf{x}_{i+1} = A_d \mathbf{x}_i + B_d \mathbf{u}_i + c_d, \qquad (3)$$

where $\mathbf{x}_i = [\mathbf{r}, \psi, \dot{\mathbf{r}}, \dot{\psi}]^T \in \mathbb{R}^4$ is the state and $\mathbf{u}_i$ is the input of the system at time $i\Delta t$. The matrix $A_d \in \mathbb{R}^{8x8}$ propagates the state $\mathbf{x}$ forward by one time-step, the matrix $B_d \in \mathbb{R}^{8x4}$ describes the effect of the input $\mathbf{u}$ on the state and the vector $c_d \in \mathbb{R}^8$ that of gravity after one time-step.

Trajectory Generation

With this approximate quadrotor model in place we can now discuss the optimization scheme to generate trajectories. The user specifies M keyframes describing a desired position k_j at a specific time-point $\eta(j)\Delta t$, where $\eta : \mathbb{N} \rightarrow \mathbb{N}$ maps the index of the keyframe to the corresponding time-point. In the case of mouse-based user input we assume constant time between consecutive positions. To compute a feasible trajectory over the whole time horizon $[0, t_f]$ we discretize time with a time-step Δt into N stages. The variables we optimize for are the quadrotor state x_i and the inputs u_i of the system Eq. (3) at each stage $i\Delta t$.

The first goal of our optimization scheme is then to follow the user inputs as closely as possible, expressed by the cost

$$E^k = \sum_{j=1}^{M} ||r_{\eta(j)} - k_j||^2. \qquad (4)$$

A small residual of E^k indicates a good match of the planned quadrotor position and the specified keyframe. The bounds

Eq. (2) together with Eq. (3) and Eq. (4) can then be formulated as a quadratic program

$$\begin{aligned} \underset{X}{\text{minimize}}\ & \frac{1}{2}X^T H X + f^T X \\ \text{subject to}\ & A_{ineq} X \leq b_{ineq} \\ \text{and}\ & A_{eq} X = b_{eq}, \end{aligned} \tag{5}$$

where X denotes the stacked state vectors x_i and inputs u_i for each time-point, H and f contain the quadratic and linear cost coefficients respectively which are defined by Eq. (4) , A_{ineq}, b_{ineq} comprise the linear inequality constraints of the inputs Eq. (2) and A_{eq}, b_{eq} are the linear equality constraints from our model Eq. (3) for each time-point $i \in 1, \dots, N$. This problem has a sparse structure and can be readily solved by most optimization software packages. However, this problem is ill-posed and the result for a particular set of keyframes might be counterintuitive at first. Since we only measure the match of quadrotor position at the keyframe times the state at other time-points is not constrained in any way except for the quadrotor dynamics. Therefore a straight path between two keyframe positions is as good as a zig-zag pattern if it is feasible. An example of this is shown in Fig. 5. To attain better results we have to further regularize the solution.

In many robotics application one goal is to minimize energy expenditure and this is often done by penalizing non-zero inputs or in other words attempting to reach desired positions with minimal wasted effort. For end-user applications, for example in the context of a racing game, one can also aim to attain smooth trajectories by penalizing higher derivatives of the quadrotor's position with respect to time such as acceleration (2nd) or jerk (3rd). We introduce the cost

$$E^d = \sum_{i=q}^{N} ||D^q \begin{bmatrix} x_i \\ \dots \\ x_{i-q} \end{bmatrix} ||^2, \tag{6}$$

where D^q is a finite-difference approximation of the q-th derivative from the last q states. Since the term jerk is not commonly known outside of engineering fields an intuition is to think of high values of jerk as a feeling of discomfort caused by too sudden motion. Humans tend to plan motion by minimizing the norm of jerk [7] and thus, minimizing jerk results in motion plans that appear pleasant to a human.

The combined cost $E = \lambda_k E^k + \lambda_d E^d$ with weights $\lambda_{k|d}$ is still a quadratic program and enables us to generate trajectories

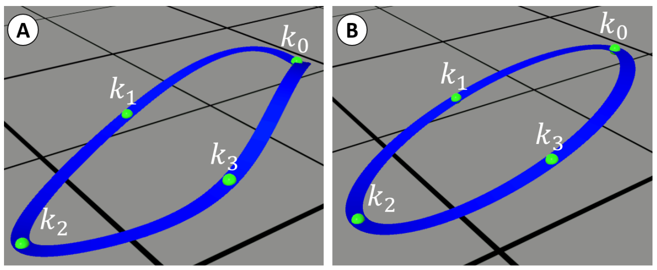

Figure 5: Same trajectory, optimized to only follow the keyframes (A) and to follow keyframes as well as minimizing snap on each stage of the optimization problem (B).

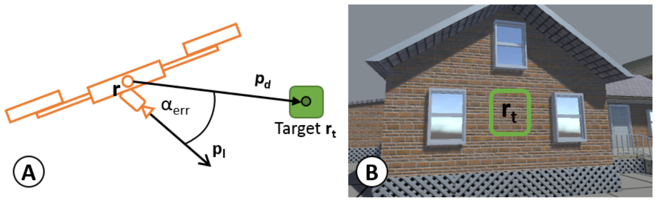

Figure 6: (A) camera direction $\mathbf{p_l}$ and distance $\mathbf{p_d}$. (B) effect of minimizing camera angle error α_{err} w.r.t. the target $\mathbf{r_t}$ in the center of the FOV of the camera.

that are feasible and that are optimal in the sense of Eq. (5). While still relatively basic in functionality this already enables a variety of use-cases such as aerial light-shows and racing-games as illustrated in the next section.

Optimizing for Human Objectives

With the basics in place we now turn our attention to including high-level human objectives into the optimization. As a running example we will consider the task of planning an aerial video-shot but we would like to emphasize that many other tasks such as 3D reconstruction or projector based augmented reality could be implemented in the same way.

We have already discussed how this process works from the user's perspective in the system overview. Here the user provides additional camera targets that should be recorded at a specific time (see Fig. 3). Furthermore, we assume that the quadrotor is equipped with a gimbal that we can control programmatically. From a cinematographic standpoint, the most pleasant viewing experience is conveyed by the use of either static cameras, panning ones mounted on tripods or cameras placed onto a dolly (cf. [22]). Changes between these shot types can be obtained by the introduction of a cut or jerk-free transitions, i.e. avoiding sudden changes in acceleration. Furthermore, it is desirable to introduce saliency constraints or in other words we want not only the camera path to be smooth but also want to keep the target motion within the image frame as steady as possible and constrain it's motion to smooth motion.

To achieve these high-level objectives, we include a target position for each stage into the optimization variable. Analogous to the quadrotor position we introduce a cost term E^t that measures the deviations of user-specified keytarget points from the target positions at the corresponding stages. We penalize higher temporal derivatives (acceleration and jerk) of the target position by including finite differences in the cost term $E^{t,d}$. To link the quadrotor and target trajectories we introduce a simple gimbal model:

$$\begin{gathered} \dot{\psi}_g = u_{g,\psi} \\ \dot{\phi}_g = u_{g,\phi} \\ [\psi_{g,min}, \phi_{g,min}]^T \leq [\psi_g, \phi_g]^T \leq [\psi_{g,max}, \phi_{g,max}]^T \\ \mathbf{u}_{g,min} \leq [u_{g,\psi}, u_{g,\phi}]^T \leq \mathbf{u}_{g,max}, \end{gathered}$$

where the inputs $u_{g,\psi}$, $u_{g,\phi}$ represent the angular velocities of the yaw ψ_g and pitch ϕ_g of the gimbal and both the inputs and the absolute angles are bounded according to the physical gimbal. The bounds specify the limits on the absolute angles

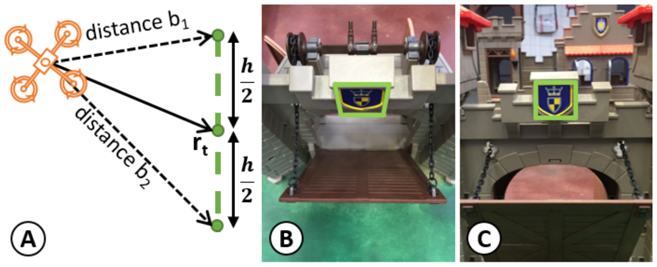

Figure 7: (A) Illustration of skewness error, where b_1 and b_2 are the distances to the upper and lower edge of the target bounding box. (B+C) Perspective without skewness correction (B) and with (C). Note that target is centered in both images.

and the angular velocities. To ensure a smooth motion of the gimbal we introduce a cost E^g on temporal finite differences of the yaw and pitch angles analogous to Eq. (6). We do not incorporate the attitude of the quadrotor into our gimbal model and therefore the bounds have to be chosen conservatively.

The angle between the current camera direction $\mathbf{p_l}$ and the direction of the target $\mathbf{p_d}$ is depicted in Fig. 6. The error is then computed by

$$\alpha_{err} = \cos^{-1}\left(\frac{\mathbf{p_d}\cdot\mathbf{p_l}}{|\mathbf{p_d}||\mathbf{p_l}|}\right) \tag{7}$$

$$\mathbf{p_d} = \mathbf{r_t} - \mathbf{r} \text{ and } \mathbf{p_l} = \begin{bmatrix}\cos\phi_g\cos(\psi+\psi_g)\\ \cos\phi_g\sin(\psi+\psi_g)\\ \sin\phi_g\end{bmatrix},$$

where $\mathbf{r}$ is the quadrotor position, $\mathbf{r_t}$ is the target position and ψ is the pitch angle of the quadrotor.

Deviations of the camera direction from the desired target are penalized by

$$E^c = \sum_{i=1}^{N}\left(\alpha^i_{err}\right)^2, \tag{8}$$

where α_{err^i} is the camera angle error at stage i. Here the separation of target trajectory from the camera direction might seem surprising but it gives more flexibility as the user can choose the weights of the importance of target keypoints and the camera direction separately.

The final aesthetic cost is related to perspective effects. Viewpoints that are to high or low relative to the recorded object of interest lead to skew and results in strong vanishing lines in the image. This is illustrated in Fig. 7. While this effect maybe desired in some situations (imagine an overhead shot) we allow the user to supress these types of distortions by optionally including a skewness cost E^s:

$$s_{err} = \frac{b_1}{b_2} - 1 = \frac{(\mathbf{p_d} + \tilde{\mathbf{h}}/2)\cdot\mathbf{p_d}}{(\mathbf{p_d} - \tilde{\mathbf{h}}/2)\cdot\mathbf{p_d}} - 1$$

$$\tilde{\mathbf{h}} = \begin{cases}\mathbf{h}, & \text{if } p_{d,3} >= p_{t,3}\\ -\mathbf{h}, & \text{else}\end{cases},$$

$$E^s = \sum_{i=1}^{N}\left(s^i_{err}\right)^2 \tag{9}$$

(10)

where $\mathbf{h}$ is a vertical vector pointing from the center of the target to the upper edge of the bounding box and s^i_{err} is the skewness error at stage i. In the computation we distinguish the case of a quadrotor flying above the target and the case of flying below a target.

Summing up the individual cost terms gives results in the final cost $E = \sum_{i=\{k,s,t,g,c\}}\lambda_i E^i$ Unfortunately α_{err} and s_{err} are non-linear in the variables of the motion plan and in consequence minimizing E can no longer be written as a quadratic program. We describe how we minimize E in the implementation section.

By penalizing snap of the quadrotor position and jerk of the camera motion the combined cost results in aesthetically pleasing footage (see the accompanying video). We can now generate a motion plan for a quadrotor that follows a target trajectory with the camera. To further support novice users we included an approximate collision-free scheme that can be used to keep a minimum distance from the target or stay at a safe distance from obstacles. Again we refer to the implementation section for details. Note that this only works for static objects where the position is known at the time of trajectory generation.

IMPLEMENTATION

In this section we describe how we implemented the different components of our system. We start with describing the iterative quadratic programming scheme, then explain the onboard controller and the quadrotor hardware and finally show how we realized the design tool.

Iterative Quadratic Programming

To solve the non-linear problem described above we resort to a scheme of iterative quadratic programming (IQP). The general idea is to linearly or quadratically approximate the problem around the current estimate of the solution. This approximate system is then solved and a better, consistent estimate of the solution is found. These iterative schemes usually converge within a few iterations despite the cost functions not being convex anymore. In our concrete implementation we start with an initial guess of the trajectory by interpolating the quadrotor positions and the camera targets between the keyframes. We also enforce all initial equality constraints to be fulfilled. As the proposed energies are usually non-convex a good initial guess is important to find a good solution. For each major iteration of our solver we build the H and f matrices of a quadratic program. This is done by quadratically approximating each of the cost terms around the trajectory X, note that this does not affect the quadratic terms in E. We also assemble the

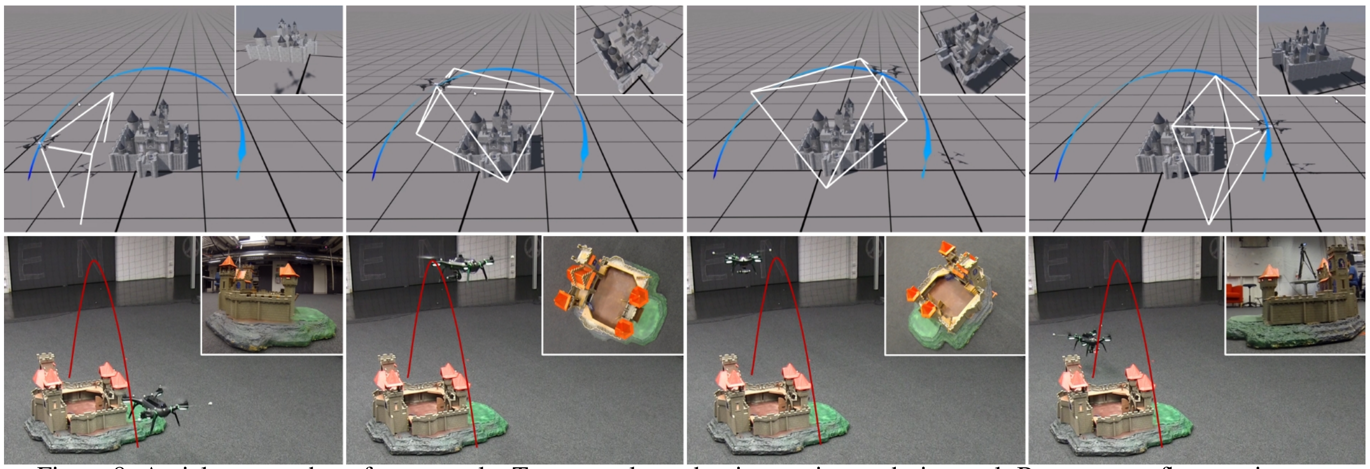
Figure 8: Aerial camera shot of a toy castle. Top row: planned trajectory in our design tool. Bottom row: flown trajectory.

bounds and equality and inequality constraints and linearize them analogously. The fully assembled system is a sparse quadratic program and can be solved by most optimization packages. The solution gives us a change dX of the current motion plan. We perform a line search with the step length α to find a new motion plan $X_{next} = X + \alpha dX$. We lower α until we find a X_{next} with a cost $C(X_{next}) < C(X)$. This step is necessary as the cost of the approximated quadratic program is only an approximation of the real residual. An empirically derived serves as termination criteria.

Obstacle Avoidance:
We approximate each obstacle as a static sphere with a radius o_r and introduce a non-convex constraint $||r \geq o_r||^2$. We linearize these constraints in each IQP iteration for each stage in time around the current trajectory X. Although we cannot guarantee global optimality of the resulting trajectories, this approach can be helpful for planning trajectories in scenes with known geometry and many objects. More advanced collision avoidance schemes, potentially taking dynamic targets into consideration (e.g., [1]), could be included in future work.

Algorithm Performance:
To evaluate the performance of our optimization scheme we measured the time necessary to generate different trajectories. The runtime of the algorithm depends on the flight duration, the number of keyframes and the constraints which are incorporated into the optimization problem. Typical run times (Intel Core i7 4GHz CPU, Matlab's quadprog solver) are $\sim$1 sec for pure QPs (e.g., the trajectory in Fig. 9 had a flight time of 30 sec and was generated in 1.4 sec) and tens of sec for IQPs (e.g., the trajectory in Fig. 8 had a flight time of 20 sec and was generated in 14 sec). Optimizing over a receding horizon which is shifted along the trajectory may be a fruitful strategy to speed up the algorithm. Another idea would be to split a trajectory in overlapping and reasonable constrained sub-trajectories and optimize them separately. Both approaches would negate the global optimality property of generated trajectories, requiring evaluation of real-world feasibility.

Onboard Control and Hardware:
Once we generate trajectory control inputs these can be transmitted to a real quadrotor. Our real-time control system builds on the PIXHAWK autopilot software [23]. Desired positions along the motion plan, camera look-at vectors and target trajectories are transmitted from a ground-station via the Robot Operating System (ROS). An LQR (Linear-quadratic regulator) running on a dedicated single-board computer computes the necessary forces and moments to track the motion plan. These forces are then translated into low-level rotor and gimbal speeds by further controllers running on a PX4 FMU. We created result figures using two different quadrotor platforms: the 3DR Solo and a custom-build Pixhawk-based platform.

Design Tool: The 3D trajectory design tool has been implemented as Unity 3D tool which allows for easy adaptation and integration of a variety of IO devices. A further advantage of this design decision is that it is easy to develop augmented reality applications such as mixing real and virtual quadrotors in a racing scenario. We have interfaced the design tool with our optimization algorithm implemented using the Matlab optimization toolbox. The source code for the optimization algorithm can be found in the supplementary materials as self-contained Matlab code.

RESULTS AND APPLICATION SCENARIOS
Despite having used camera planning as running example we note that our method is general and can be applied in many different application scenarios. In particular, the discrete nature of the proposed IQP scheme makes it straightforward to incorporate application specific constraints. In this section we want to illustrate a number of interactive usage scenarios which we have implemented using our method.

Light Painting
Quadrotors have already been used in entertainment settings, in particular to create spectacular aerial light shows (cf. [2]). However, creating such complex and coordinated flight patterns is not possible with consumer grade technologies and hence has not been accessible to the end-user. Our tool allows for straightforward end-user design of such creative scenarios.

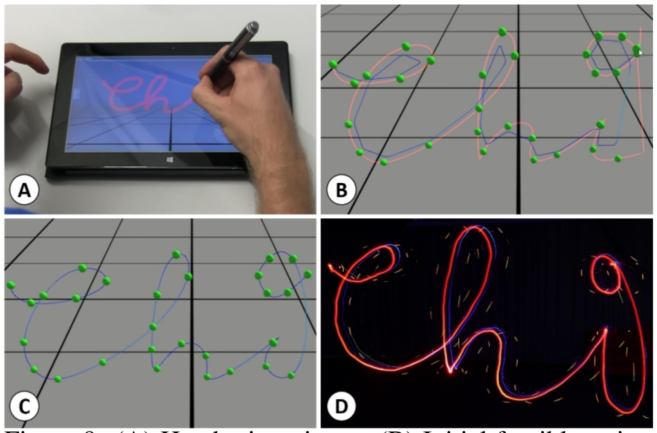

Figure 9: (A) Handwritten input. (B) Initial feasible trajectory can be overly smooth. (C) After iteration a feasible and visually pleasing trajecory is found. (D) Final result flown by MAV and captured via long-exposure photography.

One such example is illustrated in Fig. 9. Here the user provides input position constraints by writing or sketching the desired shape. Our method then generates a feasible trajectory which as a side-effect of minimizing snap also smooths the input strokes. However, the generated trajectory may not coincide with the desired output e.g. because it linearly interpolates the keyframes so that handwriting may not be legible anymore (see Fig. 9, B). The user can correct for this by changing the parameters of the optimization scheme (e.g., weights of the energies) or by adjusting keyframe positions and timings.

Once satisfied the trajectory can be flown by a real robot. In Fig. 9 we have mounted a bright LED to the robot and captured the flight path via long-exposure photography.

Racing

Another interesting application domain is that of aerial racing. First person drone racing is an emerging sport that requires a lot of expertise in manual quadrotor control. Our tool can bring this within reach of the end-user. As a proof of concept we have developed a simple aerial racing game. In this scenario a user can design a free-form race course, specifying length, curvature and other parameters as well as overall race-time.

For the race itself we implemented a semi-manual flight mode for which we changed the position controller, by remapping the feedforward term ($m\ddot{\mathbf{r}}_d$) of Eq. (14) to the joystick of a game controller. Thereby, the user can choose the direction and the strength of the feedforward-force allowing him to deviate with the quadrotor from the generated reference trajectory. Users can then, for instance, take a short cut in a curve or fly the trajectory with a higher velocity than generated by the optimization method. The score is calculated as a function of the deviation from the generated trajectory and the time needed to complete all laps. In other words, the player who managed to stay on the trajectory as fast as possible will win. We note that by manipulating the underlying controller, it would be possible to introduce further video game concepts such as player strength balancing into real-world quad racing. For example, allowing a player to temporarily race on a faster reference trajectory than his opponents.

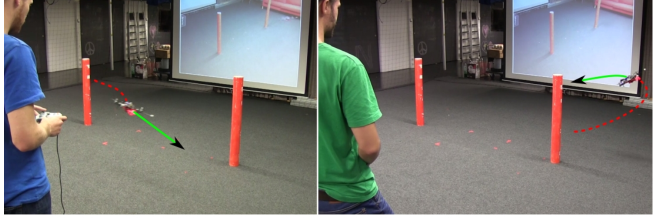
Figure 10: Two player aerial racing. User input is weighted with automatic control to adjust difficulty.

Aerial Videography

Our main results stem from the application scenario of aerial videography. We have already mentioned the technical details and how we incorporate cinematographic goals into our optimization scheme. Here we briefly summarize a number of interesting and challenging video-shots (best viewed in video).

Fig. 8 illustrates a shot where a quadrotor flies over a toy castle and at the same time records it. Here the gimbal has to smoothly track the target just as the quadrotor swoops over the object and turns around its own axis once reaching the highest point. Such a shot composition is difficult to achieve manually due to the complicated quadrotor-camera-target coordination.

Even with conventional cameras, composition of multi target shots is a very challenging task. Aerial-videography makes this even more difficult due to the many degrees of freedom and complex geometric dependencies requiring coordination for smooth, jerk-free transitions from one to the next target while airborne. In Fig. 11 we illustrate a sliding shot, transitioning between targets – the two actors – while the camera is moving from left to right and steadily rising in altitude. Throughout the entire trajectory the oerientations of quad and camera never remain constant, yet the camera targets are kept in focus and the transitions are smooth. Flying such a trajectory manually would only be possible with two operators, one for steering the camera, the other the quadrotor.

DISCUSSION AND FUTURE WORK

So far we presented a novel method to generate quadrotor trajectories subject to high-level goals and demonstrated its feasibility in different applications. In the remainder of this paper we are going to discuss the limitations of our approach and highlight interesting areas for future work.

Limitations

The optimization framework proposed in this paper has proven to be powerful and versatile however there are of course a number of limitations. First, our goal is to enable non-expert users to design arbitrary MAV use cases. While the method is generic and designed to be extensible it does require expertise and effort to formalize further objectives (that we have not treated so far) and to integrate them into the algorithm. We believe that our high-level design tool bridges the gap between the underlying optimization algorithm and end-user goals sufficiently well. Nonetheless it is an interesting future research

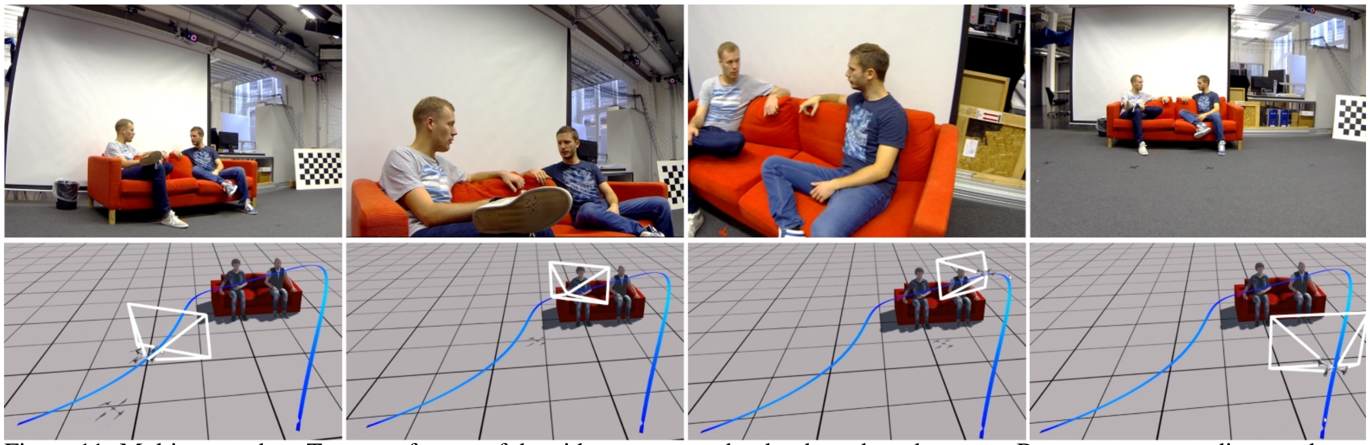
Figure 11: Multi target shot. Top row: frames of the video sequence shot by the onboard camera. Bottom row: according quadrotor positions shown in the preview of the design tool.

question how end-users could extend not only the use-cases we have demonstrated but also the optimization itself.

Currently all our application scenarios depend on a high precision indoor tracking system. This is a limiting factor as one would of course like to fly many of the examples outdoors using GPS sensing. To this end our method is generic and could be made to work with any localization system, in particular with GPS position data in outdoor scenarios. However, we have not implemented this and of course the localization accuracy would impact the exact results.

Future Work

The optimization-based design of quadrotor trajectories subject to high-level user constraints is a comprehensive research space and our work only started to cover it. The investigations we did so far raised a number of additional research problems. In our method, the time at which a certain keyframe is reached cannot be changed by the optimization scheme. To extend our algorithm, it would be interesting to formalize the optimization problem in a way that both, the keyframe's position and its time can be optimized.

Furthermore, the use case of aerial videography raised the question: what is an aesthetic aerial video sequence and is it possible to optimize for it? By incorporating human objectives into the optimization, our work already presents a starting point, nevertheless it would be interesting to see whether further concepts and rules of cinematography can be incorporated into the optimization problem.

Nonetheless, our method is generic and can be applied to further use cases. For example, it would be possible to extend the method to 3D scanning of buildings and other objects of interest. Here one could integrate objectives that capture e.g., reconstruction quality and surface coverage. Another possible example includes an advanced flying action camera for outdoor usage enabling users to trigger pre-defined trajectories on-the-fly. These scenarios would obviously require additional information such as the environment's 3D geometry for collision avoidance and accurate localization of drone and human in the outdoor case. Finally, it is not only drones that our method applies to. Most straightforward would be an extension to other actuated camera platforms such as dollies and robotic arms.

CONCLUSION

In summary we have proposed a user in the loop design tool for the creation of aerial robotic behavior. At its core lies an optimization-based algorithm that integrates low-level quadrotor control constraints and high-level human objectives. Therefore, we used a linear approximation of the quadrotor model enabling us to generate trajectories subject to the physical limits of a quadrotor. Stating the problem as discrete, additionally permits the easy incorporation of high-level constraints to support the user, for instance, in the creation of pleasing aerial footage. This allows users to concentrate on the creative and aesthetic aspects of the task at hand and requires little to no expertise in quadrotor control or the target domain. We have demonstrated the flexibility and utility of our approach in three different use cases including aerial videography, light painting and racing.

ACKNOWLEDGEMENTS

We thank Cécile Edwards-Rietmann for providing the video voiceover and Gábor Sörös for creating the aerial light-painting videos. We are also grateful for the valuable feedback from the associate chairs and external reviewers. This work was partially funded by the Swiss Science Foundations (UFO 200021L_153644) and Microsoft Research.

REFERENCES

1. Javier Alonso-Mora, Tobias Naegeli, Roland Siegwart, and Paul Beardsley. 2015. Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots* 39, 1 (2015), 101–121. DOI: http://dx.doi.org/10.1007/s10514-015-9429-0

2. ArsElectronica. 2012. Spaxels. (2012). http://www.aec.at/spaxels/.

3. Cannes Festival. 2012. New Directors' Showcase. Video. (2012). https://youtu.be/cseTX_rW3uM

4. Marc Christie, Rumesh Machap, Jean-Marie Normand, Patrick Olivier, and Jonathan Pickering. 2005. Virtual camera planning: A survey. In *Smart Graphics*. Springer, 40–52.

5. T.J. Diaz. 2015. Lights, drone... action. *Spectrum, IEEE* 52, 7 (July 2015), 36–41. DOI: http://dx.doi.org/10.1109/MSPEC.2015.7131693

6. Nadeem Faiz, Sunil K Agrawal, and Richard M Murray. 2001. Trajectory planning of differentially flat systems with dynamics and inequalities. *Journal of Guidance, Control, and Dynamics* 24, 2 (2001), 219–227.

7. Tamar Flash and Neville Hogan. 1985. The coordination of arm movements: an experimentally confirmed mathematical model. *The journal of Neuroscience* 5, 7 (1985), 1688–1703.

8. Michael L. Gleicher and Feng Liu. 2008. Re-cinematography: Improving the Camerawork of Casual Video. *ACM Trans. Multimedia Comput. Commun. Appl.* 5, 1, Article 2 (Oct. 2008), 28 pages. DOI: http://dx.doi.org/10.1145/1404880.1404882

9. M. Grundmann, V. Kwatra, and I. Essa. 2011. Auto-directed video stabilization with robust L1 optimal camera paths. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. 225–232. DOI: http://dx.doi.org/10.1109/CVPR.2011.5995525

10. Keita Higuchi and Jun Rekimoto. 2012. Flying Head: Head-synchronized Unmanned Aerial Vehicle Control for Flying Telepresence. In *SIGGRAPH Asia 2012 E-Tech (SA '12)*. ACM, New York, NY, USA, 12:1—-12:2. DOI: http://dx.doi.org/10.1145/2407707.2407719

11. Keita Higuchi, Tetsuro Shimada, and Jun Rekimoto. 2011. Flying Sports Assistant: External Visual Imagery Representation for Sports Training. In *Augmented Human International Conference (AH '11)*. ACM, 7:1—-7:4. DOI: http://dx.doi.org/10.1145/1959826.1959833

12. Niels Joubert, Mike Roberts, Anh Truong, Floraine Berthouzoz, and Pat Hanrahan. 2015. An Interactive Tool for Designing Quadrotor Camera Shots. *ACM Trans. Graph.* 34, 6, Article 238 (Oct. 2015), Article 238, 11 pages. DOI: http://dx.doi.org/10.1145/2816795.2818106

13. Jun Kato, Daisuke Sakamoto, and Takeo Igarashi. 2012. Phybots: A Toolkit for Making Robotic Things. In *Proceedings of the Designing Interactive Systems Conference (DIS '12)*. ACM, New York, NY, USA, 248–257. DOI: http://dx.doi.org/10.1145/2317956.2317996

14. Johannes Kopf, Michael F. Cohen, and Richard Szeliski. 2014. First-person Hyper-lapse Videos. *ACM Trans. Graph.* 33, 4, Article 78 (July 2014), 10 pages. DOI: http://dx.doi.org/10.1145/2601097.2601195

15. Taeyoung Lee, Melvin Leok, and N Harris McClamroch. 2013. Nonlinear robust tracking control of a quadrotor UAV on SE (3). *Asian Journal of Control* 15, 2 (2013), 391–408.

16. Tsai-Yen Li and Chung-Chiang Cheng. 2008. Real-Time Camera Planning for Navigation in Virtual Environments. In *Smart Graphics*, Andreas Butz, Brian Fisher, Antonio Krger, Patrick Olivier, and Marc Christie (Eds.). Lecture Notes in Computer Science, Vol. 5166. Springer Berlin Heidelberg, 118–129. DOI: http://dx.doi.org/10.1007/978-3-540-85412-8_11

17. Kexi Liu, Daisuke Sakamoto, Masahiko Inami, and Takeo Igarashi. 2011. Roboshop: Multi-layered Sketching Interface for Robot Housework Assignment and Management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 647–656. DOI: http://dx.doi.org/10.1145/1978942.1979035

18. Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. 2013. Bundled Camera Paths for Video Stabilization. *ACM Trans. Graph.* 32, 4, Article 78 (July 2013), 10 pages. DOI: http://dx.doi.org/10.1145/2461912.2461995

19. Sergei Lupashin and Raffaello DAndrea. 2012. Adaptive fast open-loop maneuvers for quadrocopters. *Autonomous Robots* 33, 1-2 (April 2012), 89–102. http://link.springer.com/10.1007/s10514-012-9289-9

20. Robert Mahony, Vijay Kumar, and Peter Corke. 2012. Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor. *IEEE Robotics & Automation Magazine* 19, 3 (Sept. 2012), 20–32. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6289431

21. Tobias Martin, Nobuyuki Umetani, and Bernd Bickel. 2015. OmniAD: Data-driven Omni-directional Aerodynamics. *ACM Trans. Graph.* 34, 4, Article 113 (July 2015), 12 pages. DOI: http://dx.doi.org/10.1145/2766919

22. Joseph V Mascelli. 1998. *The five C's of cinematography: motion picture filming techniques*. Silman-James Press.

23. Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. 2012. PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots* 33, 1-2 (Feb. 2012), 21–39. http://link.springer.com/10.1007/s10514-012-9281-4

24. Daniel Mellinger and Vijay Kumar. 2011. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2520–2525.

25. Florian 'Floyd' Mueller and Matthew Muirhead. 2015. Jogging with a Quadcopter. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 2023–2032. DOI: http://dx.doi.org/10.1145/2702123.2702472

26. Tobias Nägeli, Christian Conte, Alexander Domahidi, Manfred Morari, and Otmar Hilliges. 2014. Environment-independent formation flight for micro aerial vehicles. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. 1141–1146. DOI: http://dx.doi.org/10.1109/IROS.2014.6942701

27. Tayyab Naseer, J Sturm, and D Cremers. 2013. Followme: Person following and gesture recognition with a quadrocopter. *Proc. IROS* (2013). https://vision.in.tum.de/_media/spezial/bib/naseer2013iros.pdf

28. Kei Nitta, Keita Higuchi, and Jun Rekimoto. 2014. HoverBall: Augmented Sports with a Flying Ball. In *Augmented Human International Conference (AH '14) (AH '14)*. ACM, New York, NY, USA, 13:1—-13:4. DOI: http://dx.doi.org/10.1145/2582051.2582064

29. Daisuke Sakamoto, Koichiro Honda, Masahiko Inami, and Takeo Igarashi. 2009. Sketch and run. In *ACM SIGCHI*. ACM Press, New York, New York, USA, 197. http://dl.acm.org/citation.cfm?id=1518701.1518733

30. Jürgen Scheible, Achim Hoth, Julian Saal, and Haifeng Su. 2013. Displaydrone: a flying robot based interactive display. In *ACM International Symposium on Pervasive Displays (PerDis '13)*. ACM Press, New York, New York, USA, 49. http://dl.acm.org/citation.cfm?id=2491568.2491580

31. Yuta Sugiura, Diasuke Sakamoto, Anusha Withana, Masahiko Inami, and Takeo Igarashi. 2010. Cooking with robots. In *ACM SIGCHI*. ACM Press, New York, New York, USA, 2427. http://dl.acm.org/citation.cfm?id=1753326.1753693

32. Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. 2014. Pteromys: Interactive Design and Optimization of Free-formed Free-flight Model Airplanes. *ACM Trans. Graph.* 33, 4, Article 65 (July 2014), 10 pages. DOI: http://dx.doi.org/10.1145/2601097.2601129

33. I-Cheng Yeh, Chao-Hung Lin, Hung-Jen Chien, and Tong-Yee Lee. 2011. Efficient camera path planning algorithm for human motion overview. *Computer Animation and Virtual Worlds* 22, 2-3 (2011), 239–250. DOI: http://dx.doi.org/10.1002/cav.398

34. Shigeo Yoshida, Takumi Shirokura, Yuta Sugiura, Daisuke Sakamoto, Tetsuo Ono, Masahiko Inami, and Takeo Igarashi. 2015. RoboJockey: Designing an Entertainment Experience with Robots. *IEEE computer graphics and applications* 1 (Jan. 2015), 1. http://www.computer.org/csdl/mags/cg/preprint/07005367-abs.html

35. Shengdong Zhao, Koichi Nakamura, Kentaro Ishii, and Takeo Igarashi. 2009. Magic Cards: A Paper Tag Interface for Implicit Robot Control. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 173–182. DOI: http://dx.doi.org/10.1145/1518701.1518730

APPENDIX

In the work proposed here we use an approximation of a full, non-linear quadcopter model for the optimization-based generation of trajectories. However, the resulting trajectories need to be flown by a real quadcopter and hence one must relate the approximate model to the full model of the quadcopter. Here we briefly summarize the modeling and control aspects necessary for replication of our method. A full introduction to this topic is beyond the scope of this work and we refer the interested reader to [20].

Quadrotor Model

A quadrotor is a robot with four identical rotors which generate a thrust and a moment orthogonal to the square they span. Our quadrotor model closely follows [24]. To describe the configuration of a quadrotor we define its position as the location of the center of mass in an inertial world coordinate frame $(\mathbf{x}_W, \mathbf{y}_W, \mathbf{z}_W)$, and its attitude as the rotation of the body-fixed frame $(\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B)$ with respect to the world frame (see Fig. 12). The rotation matrix from body to world frame is then given by $R_{BW} = [\mathbf{x}_B\ \mathbf{y}_B\ \mathbf{z}_B] \in \mathbb{SO}(3)$. Each rotor of the drone has an angular speed ω_i and produces a force F_i and moment M_i, according to

$$F_i = k_F \omega_i^2, \quad M_i = k_M \omega_i^2,$$

where k_F and k_M are constants specific to the rotors. Therefore, the control input to the quadrotor can be written as $\mathbf{u}$ where u_1 is the net force in $\mathbf{z}_B$ direction and u_2, u_3, u_4 are the moments in $\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B$ direction acting on the quadrotor. The input can be expressed in terms of the rotor speeds ω_1, ω_2, ω_3, ω_4:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}, \qquad (11)$$

where L is the distance from the axis of rotation of the rotors to the center of mass of the quadrotor.

The position of the quadrotor in the world frame can be specified according to Newton's equation of motion governing the acceleration of a mass point:

$$m\ddot{\mathbf{r}} = u_1 \mathbf{z}_B + m\mathbf{g} \in \mathbb{R}^3, \qquad (12)$$

where $\mathbf{r}$ is the position vector, $\mathbf{g} = [0, 0, -g]^T$ is the gravity vector pointing along the $-z$ axis of the world frame, g is the gravitational constant and m is the mass of the quadrotor.

The Euler rotation equations are

$$\mathbf{M}_B = I\dot{\omega}_{BW} + \omega_{BW} \times I\omega_{BW} \in \mathbb{R}^3, \qquad (13)$$

where $\mathbf{M}_B = [u_2, u_3, u_4]^T = [\omega_x, \omega_y, \omega_z]^T$ is the moment vector acting on the quadrotor in the body frame, ω_{BW} is the angular velocity of the body frame in the world frame and I is the moment of inertia of the quadrotor in the body frame.

Quadrotor Control

As can be seen from the equations Eq. (11), Eq. (12) and Eq. (13), the quadrotor configuration has 6 degrees of freedom but only 4 actuators. Therefore it is an underactuated system and cannot follow arbitrary trajectories in the configuration space. However, Mellinger et al. show that the system is *flat* [6] with respect to the 4 *flat outputs* $[\mathbf{r}, \psi]^T$ and thus a quadrotor can follow trajectories in this space, given that the corresponding inputs are bounded to values that the quadrotor can achieve [24]. This flat output space is the configuration of our approximate quadrotor model.

We use the linear controller from [1] to generate the corresponding inputs for the quadrotor. The desired thrust along the z-axis of the body frame is computed as

$$\mathbf{F}_d = -K(\mathbf{x} - \mathbf{x}_d) + m(g\mathbf{z}_w + \ddot{\mathbf{r}}_d), \tag{14}$$

where $x = [\mathbf{r}, \dot{\mathbf{r}}]^T$ is the actual and $\mathbf{x}_d$ the desired position and velocity of the quadrotor and $m(g\mathbf{z}_w + \ddot{\mathbf{r}}_d)$ the feedforward term which compensates for gravity and known accelerations. The state feedback matrix K is computed using a linear quadratic control strategy with integral action.

The desired force F_d already defines two degrees of freedom of the quadrotor attitude. Using the nonlinear control strategy on $SO(3)$ described in [15] we employ the desired yaw angle ψ_d to compute the desired attitude R_{BWd} of the quadrotor:

$$\begin{aligned}
\mathbf{z}_{Bd} &= \frac{\mathbf{F}_d}{||\mathbf{F}_d||} \\
\mathbf{y}_{Bd} &= \frac{\mathbf{z}_{Bd} \times [cos(\psi_d), sin(\psi_d), 0]^T}{||\mathbf{z}_{Bd} \times [cos(\psi_d), sin(\psi_d), 0]^T||} \\
\mathbf{x}_{Bd} &= \mathbf{y}_{Bd} \times \mathbf{z}_{Bd} \\
R_{BWd} &= [\mathbf{x}_{Bd}, \mathbf{y}_{Bd}, \mathbf{z}_{Bd}],
\end{aligned}$$

where $\mathbf{y}_{Bd}$ are the desired x- and y-axis of the body frame. To control the attitude we can now calculate the desired moment vector $\mathbf{M}_{Bd}$ in $\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B$ direction,

$$\begin{aligned}
\mathbf{e}_R &= \frac{1}{2} vee\left(R_d^T R_{BW} - R_{BW}^T R_d\right) \\
\mathbf{e}_\omega &= R_{BW}^{-1}\left(\omega_{BW} - \omega_{BWd}\right) \\
\mathbf{M}_{Bd} &= -K_R \mathbf{e}_R - K_\omega \mathbf{e}_\omega,
\end{aligned}$$

where R_{BW} is the actual rotation of the quadrotor, $\mathbf{e}_R$ is the rotation error, ω_{BW}, ω_{BWd} are the angular and desired angular velocity, $\mathbf{e}_\omega$ is the angular velocity error and vee is the *vee map* from $so(3) \rightarrow \mathbb{R}^3$. From $\mathbf{F}_d$ and $\mathbf{M}_{Bd}$ we can calculate the input u and thereby the velocities of the rotors needed to reach the desired position and yaw angle:

$$u_1 = \mathbf{F}_d \cdot \mathbf{z}_B \tag{15}$$

$$[u_2, u_3, u_4]^T = \mathbf{M}_{Bd},$$

where Eq. (15) is the projection of the desired thrust $\mathbf{F}_d$ on the actual z-vector of the body frame $\mathbf{z}_B$. Finally, using Eq. (11) we can compute the angular velocities ω_i corresponding to the input u.

Validity of Approximate Quadrotor Model

Following the approach in [26], we assume that the rotational dynamics of a quadrotor are fast compared to its translational dynamics thus we can describe the behavior of the quadrotor by the thrust vector $\mathbf{u}_r$ and the moment u_ψ along the body-frame z-axis. In the following we will only refer to the norm u_r of the force vector $\mathbf{u}_r$

Let $F_{\max}$ be the maximum force and $M_{\max}$ be the maximum moment each motor can produce. Then the bound on the maximum possible thrust that the quadrotor can achieve (i.e. all motors full on) is

$$u_r \leq u_{r,\max} = 4F_{\max}$$

and the bound on the maximum possible moment (i.e. two motors rotating in same direction full on and the other two off) is

$$u_\psi \leq u_{\psi,\max} = 2M_{\max}.$$

Because the force and moment are coupled it is not possible to achieve full thrust $u_{r,\max}$ and full moment $u_{\psi,\max}$ at the same time.

Let us now assume a stricter bound on the maximum moment of the quadrotor:

$$u_\psi \leq u_{\psi,lim} = \beta u_{\psi,\max}$$

where $\beta \in [0, 1]$. If we want to be able to achieve a moment of $u_{\psi,lim}$ at all times we have to take into account that in the extreme case two motors will be limited to a force of $F_{lim} = (1-\beta)F_{\max}$ and thus the bound on the thrust of the quadrotor is

$$u_r \leq u_{r,lim} = (2 + 2(1-\beta))F_{\max} = \left(1 - \frac{\beta}{2}\right) u_{r,\max}.$$

For example, if $\beta = 0.2$, i.e. bounding the moment to 20% of the quadrotors maximum moment the quadrotor can still achieve 90% of its maximum thrust at all times. These limits still allow the agility of a quadrotor to be sufficient for many use cases.

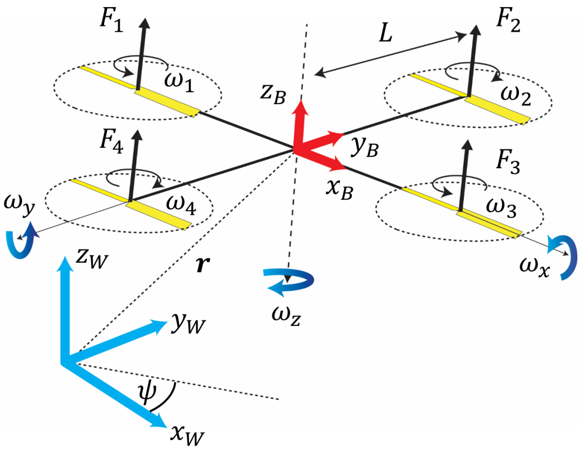

Figure 12: A quadrotor in 3D with its flat outputs (position $\mathbf{r}$, yaw angle ψ), world ($\mathbf{x}_W$, $\mathbf{y}_W$, $\mathbf{z}_W$) and body frame ($\mathbf{x}_B$, $\mathbf{y}_B$, $\mathbf{z}_B$), the rotational velocities of the quadrotor in each dimension (ω_x, ω_y, ω_z), the distance L from the axis of rotation of a rotor to the center of mass of the quadrotor, as well as the thrust forces F_i and angular velocities ω_i of each rotor.