Real-time Planning for Automated Multi-View Drone Cinematography

TOBIAS NÄGELI, AIT Lab, ETH Zurich
LUKAS MEIER, AIT Lab, ETH Zurich
ALEXANDER DOMAHIDI, Embotech GmbH
JAVIER ALONSO-MORA, Delft University of Technology
OTMAR HILLIGES, AIT Lab, ETH Zurich

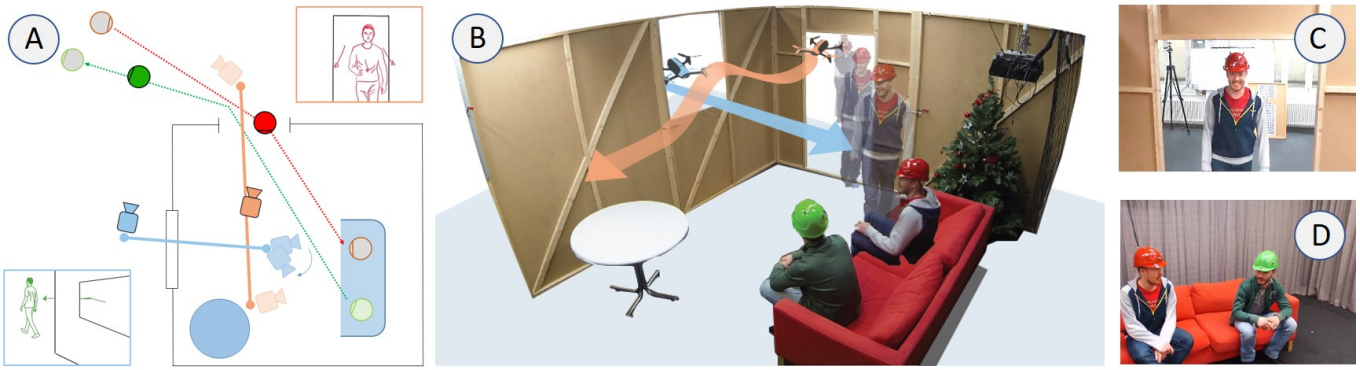

Fig. 1. We propose a method to jointly optimize 3D trajectories and control inputs for automated drone videography in dynamic scenes. Taking user specified high-level plans and screen-space framing objectives as input(a), our method generates trajectories that respect the physical limits of the quadrotor and constraints imposed by the environment while fulfilling the high-level plan and aesthetic objectives as well as possible (b). The method automates single-shot recording of complex multi-view scenes in cluttered and dynamic environments (d+c).

We propose a method for automated aerial videography in dynamic and cluttered environments. An online receding horizon optimization formulation facilitates the planning process for novices and experts alike. The algorithm takes high-level plans as input, which we dub virtual rails, alongside interactively defined aesthetic framing objectives and *jointly* solves for 3D quadcopter motion plans and associated velocities. The method generates control inputs subject to constraints of a non-linear quadrotor model and dynamic constraints imposed by actors moving in an a priori unknown way. The output plans are physically feasible, for the horizon length, and we apply the resulting control inputs directly at each time-step, without requiring a separate trajectory tracking algorithm. The online nature of the method enables incorporation of feedback into the planning and control loop, makes the algorithm robust to disturbances. Furthermore, we extend the method to include coordination between multiple drones to enable dynamic multi-view shots, typical for action sequences and live TV coverage. The algorithm runs in real-time on standard hardware and computes motion plans for several drones in the order of milliseconds. Finally, we evaluate the approach qualitatively with a number of challenging shots, involving multiple drones and actors and qualitatively characterize the computational performance experimentally.

This work is partially supported by a Microsoft Research grant.

DOI: http://dx.doi.org/10.1145/3072959.3073712

CCS Concepts: •**Computing methodologies** → **Computer graphics; Robotic planning; Motion path planning;**

Additional Key Words and Phrases: aerial videography, multi-drone, collision-avoidance

ACM Reference format:
Tobias Nägeli, Lukas Meier, Alexander Domahidi, Javier Alonso-Mora, and Otmar Hilliges. 2017. Real-time Planning for Automated Multi-View Drone Cinematography. *ACM Trans. Graph.* 36, 4, Article 132 (July 2017), 10 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073712

1 INTRODUCTION

Accessible quadrotor hardware now allows for end-user creation of aerial videography which previously resided firmly in the realm of high-end film studios. However, designing trajectories that fulfill aesthetic objectives *and* respect the physical limits of real robots remains a challenging task both for non-experts and professionals. Especially when filming in dynamic environments with moving subjects, the operator has to consider and trade off many degrees of freedom relating to subjects' motions, aesthetic considerations and the physical limits of the robot simultaneously, rendering manual approaches infeasible.

Existing methods for planning of quadcopter trajectories [Gebhardt et al. 2016; Joubert et al. 2015; Roberts and Hanrahan 2016] allow users to specify shots in 3D virtual environments and to generate flight plans automatically. Typically, this is formulated as an *offline* optimization problem which generates a timed reference

trajectory and control input parameters from user-specified 3D positions and camera look-at directions, subject to a model of the robot dynamics. The resulting plan is then tracked *online* using a feedback controller. Due to this feedforward, open-loop nature for trajectory planning and tracking, such algorithms are not well suited to handle drastic environmental disturbances [Chen et al. 1992], typical for cluttered environments with moving subjects. Therefore they are restricted to filming of mostly static scenes. In contrast, dynamic scenes require continuous re-planning in real-time to guarantee collision-free trajectories and record the intended footage, for example to keep a moving actor properly framed.

In this paper, we propose a general method for planning of aerial videography in cluttered and dynamic environments. The method jointly optimizes 3D motion paths, the associated velocities and control inputs for a flying camera in an *online* fashion. Our method takes user specified, high-level plans alongside image-based framing objectives as input (Fig. 1, a+b). The input paths do *not* need to be physically feasible in the sense of [Roberts and Hanrahan 2016], since our method only uses them for guidance. Furthermore, the inputs can be updated interactively at every time-step by the user. The algorithm adapts the high-level plans in real-time to produce dynamically feasible trajectories for the drones. It takes the motion of the filmed subjects into consideration and inherently accounts for the dynamic constraints due to the actuation limits of the drone, which is crucial to generate collision-free paths.

These multiple objectives and constraints are expressed mathematically in a non-linear model predictive contouring control (MPCC) formulation, solving for quadrotor states and control inputs online and simultaneously in a receding horizon fashion: The first control move of the plan is applied to the quadrotors, and the entire trajectory is re-computed at the next sampling instance. Solving non-linear MPCC problems numerically at the sampling rates required by fast mechanical systems, i.e. on the order of a few milliseconds, is a computationally demanding task and solving such problems in real-time has only recently become feasible thanks to specialized solvers [Domahidi and Jerez 2016].

Furthermore, the algorithm allows for planning of multi-angle shots and for the positioning of several quadrotors to film one or more dynamic subjects simultaneously. This is a common approach in films and TV broadcasts when depicting moving subjects, such as in action and sports sequences. In such settings, it is desirable to provide different views of the subjects, which have to be filmed in a single take, since humans struggle in precisely reproducing their motions from the recorded footage. To enable such shots, we extend our method to produce collision-free paths between multiple drones and subjects simultaneously. The formulation also minimize mutual visibility of multiple cameras so that the recorded shots are unobstructed and do not contain the other flying cameras.

We demonstrate our method via several challenging, and in some cases previously impossible shots, involving multiple, moving subjects and using several flying cameras simultaneously. Furthermore, we report initial feedback elicited from an expert camera operator. Finally, we characterize the computational cost of our method in controlled experiments and show that it is capable of generating feasible trajectories in the order of milliseconds, even for multiple subjects and multiple drones.

2 RELATED WORK

Aerial videography design tools: Various tools support the task of planning quadrotor-based video shots. Commercially available applications are often limited to placing waypoints on a 2D map [apm 2015; dji 2015; lit 2015] and some consumer-grade drones allow to interactively control the quadrotor's camera as it tracks a predetermined path (e.g., [3dr 2015]). These tools generally do not provide means to ensure feasibility of the resulting plans. In consequence, several algorithms for the planning of physically feasible quadcopter trajectories have been proposed. Such tools allow for planning of aerial shots in 3D virtual environments [Gebhardt et al. 2016; Joubert et al. 2015; Roberts and Hanrahan 2016] and employ offline optimization methods to ensure that both aesthetic objectives and robot modelling constraints are considered. Joubert et al.'s method [2015] computes control inputs along a pre-defined path and detects violations of the robot model constraints. However correcting these violations is offloaded to the user. Gebhardt et al. [2016] generates feasible trajectories subject to a linearized quadrotor model and hence requires conservative limits on the control inputs. The method proposed in [Roberts and Hanrahan 2016] takes physically infeasible trajectories and computes the closest possible feasible trajectory by re-timing the user-defined velocities subject to a non-linear quadrotor model.

While [Joubert et al. 2015] allows to adjust the velocity along the planned trajectory at execution time, all of the above methods are *offline* and convert the user's desired path into a time-dependent reference trajectory which is then tracked by a separate feedback controller at flight-time. Furthermore, generating control inputs over the length of the entire trajectory is computationally expensive and existing methods are not capable of re-planning a suitable trajectory online, for example to avoid collisions with dynamic obstacles or to film targets that move in unpredictable ways.

The dimensionality of the problem can be reduced by planning in the torus-subspace [Lino and Christie 2015] to attain real-time performance [Galvane et al. 2016; Joubert et al. 2016], albeit at the cost of loosing generality in the types of plans that can be generated. Very recently a model predictive control (MPC) formulation to optimize cinematographic constraints, such as visibility and position on the screen, subject to robot constraints in real-time has been proposed [Nägeli et al. 2017]. However, the method is limited to a single drone and, more importantly, only computes *local* trajectories and can not handle user-defined paths as input. In contrast, our method integrates high-level input paths for guidance of an online trajectory planner, applies to multiple drones and optimizes for inter-drone collision-freedom and suppresses mutual visibility.

Camera control in virtual environments: Our problem is similar to that of automatic camera placement in virtual environments (VE), which has been studied extensively in computer graphics. We refer to the comprehensive review in [Christie et al. 2008], with the majority of methods using discrete optimization formulations. Many of these methods define viewing constraints in screen-space for single subjects [Drucker and Zeltzer 1994; Gleicher and Witkin 1992; Lino et al. 2011] and two actors shot simultaneously [Lino and Christie 2015], citing better usability as main motivation. Our approach is related to these approaches in that it minimizes projection error

of subjects in screen-space to produce desired framing effects. Finally, we take inspiration in Christie et al. [2002] in that we take geometric primitives as input paths and 'warp' these in real-time to adhere to model, environment and aesthetic constraints. However, virtual environments are not limited by real-world physics and robot constraints and hence can produce arbitrary camera trajectories, velocities and viewpoints.

Trajectory optimization and tracking control: The problem of trajectory generation for dynamical systems is well studied in the computer graphics [Geijtenbeek and Pronost 2012] and robotics literature (cf., [Betts 2010]). Traditionally, sequences of input positions are converted to time-dependent reference trajectories using an appropriate trajectory optimization method and model of the system dynamics. Approaches encoding the system dynamics as a set of equality constraints are known as spacetime constraints in graphics [Rose et al. 1996; Witkin and Kass 1988] and direct collocation in robotics [Betts 2010] but this approach can lead to slow convergence when optimizing over the entire trajectory, in particular for systems with highly non-linear dynamics such as quadrotors.

Exploiting the differential flatness of quadrotors in the output space, several methods exist for the generation of trajectories for aggressive Micro Aerial Vehicle (MAV) flight [Bry et al. 2015; Mellinger and Kumar 2011], for generating collision-free trajectories via convex optimization [Alonso-Mora et al. 2015] or for state interception with a quadrotor [Mueller and D'Andrea 2013]. The previously discussed videography tools extend the method in [Mellinger and Kumar 2011].

An alternative to optimization-based methods are sampling-based approaches, which leverage rapidly expanding random trees (RRT) [Karaman and Frazzoli 2011] or exact algorithms such as the A* algorithm to find an optimal or near-optimal path through cluttered environments [MacAllister et al. 2013; Saunders et al. 2005]. Regardless of the trajectory planning algorithm it is necessary to use feedback controllers to track the reference trajectory accurately. However, tracking open loop reference trajectories inherently involves the risk of performance deterioration and constraint violation if disturbances or modeling errors arise [Chen et al. 1992].

Online path planning and contouring control: To avoid issues associated with tracking based methods and to reduce reliance on feasibility of the high-level path planner, unified approaches have been proposed that address path optimization and path following jointly. Such methods determine the evolution of the path and the actuator inputs simultaneously using available feedback. It has been shown that appropriate online path-following can alleviate performance limitations for both linear and non-linear systems [Aguiar et al. 2008]. In particular, Model Predictive Control (MPC) [Faulwasser et al. 2009] approaches have been used successfully for 2D industrial contouring [Lam et al. 2010] and 2D RC racing [Liniger et al. 2015] applications, in which time-optimal progress along the path is the main objective. Our work is inspired by this particular MPCC formulation but to the best of our knowledge we are the first to adapt and extend this framework to aerial videography (in 3D space). This is inherently a different problem: instead of solving for the sole objective of following a trajectory in a time optimal fashion [Lam et al. 2010; Liniger et al. 2015] or tracking a trajectory with pre-determined timings [Roberts and Hanrahan 2016], we determine how fast and locally where to the quadrotor should fly based on the dynamics of the filmed subjects and the user specified framing objectives. We then solve for the resulting state-space trajectories via online MPCC. The proposed method is particularly well-suited for dynamic shots and filming in densely cluttered environments because it can find, in the least-squares sense, an optimized trade-off between high-level user plans and a priori unknown actor and environmental motion.

3 METHOD

Our real-time method, summarized in Alg. 1, enables the automation of aerial shots in cluttered and dynamic environments with one or more subjects to be filmed. The method is general enough to account for both global guidance provided by the user (e.g., a cameraman) as well as to account for real-time constraints and aesthetic requirements imposed by the scene being recorded. In particular, we account for the following:

- Coarsely follow a 3D guidance path for the flying camera. We will refer to this path as "virtual rails" in analogy to physical camera cranes and dollies. This path may be adjusted and moved online (cf. Fig. 3).
- Satisfy cinematographic objectives, again specified interactively, such as the framing or size of the object on screen.
- Respect the dynamic model and environmental constraints to ensure feasibility of the resulting plan.

From these objectives, we formulate a receding horizon non-linear optimization problem under constraints that can be solved with state-of-the-art software. The proposed method computes and adapts in real-time a feasible and collision-free trajectory to record a dynamic scene as close as possible to the user-provided input specification.

Algorithm 1 Compute drone state

```
1:  x_0 ←initialize_horizon(nr. subjects)
2:  loop
3:      retrieve measurements and predict states:          ▷ Sec. 3.1
4:      x ←KalmanFilter(z_quad)
5:      [p_s, ṗ_s] ←KalmanFilter(z_sub)
6:      retrieve dynamic inputs from user:
7:      S_s ←framing setpoints from UI                     ▷ Sec. 3.2
8:      S_c ←input trajectories from UI                    ▷ Sec. 3.4
9:      solve for path and quadrotor configuration:
10:     s(θ) ←compute_virtual_rail(p_s, ṗ_s, S_s, S_c)     ▷ Sec. 3.4
11:     update cost & constraints, solve MPCC Eq. (8)      ▷ Sec. 3.5
12:     apply_inputs(u_0)
13: end loop
```

3.1 Dynamical models

We introduce the models used in our formulation. We do not include a full notation section and refer to Appendix A for completeness.

Subjects to be filmed: Let $\mathbf{p}_s \in \mathbb{R}^3$ denote the position of a subject to be filmed and $\dot{\mathbf{p}}_s \in \mathbb{R}^3$ its velocity. The full state of the subject is

then denoted by $\xi = (\mathbf{p}_s, \dot{\mathbf{p}}_s) \in \mathbb{R}^6$, with simple linear dynamics

$$\dot{\xi} = (\dot{\mathbf{p}}_s, 0),$$

A standard Kalman filter is used to estimate this state and to update it with measured position data. Further details can be found in [Nägeli et al. 2017].

Flying camera: Our method is agnostic to the particular quadrotor or drone hardware employed. It is based on a mathematical model in form of a differentiable function $f : \mathbb{R}^{n_x \times n_u} \rightarrow \mathbb{R}^{n_x}$, denoting a discrete-time state update equation of the flying camera,

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k),$$

where n_x are the dimensions of the states $\mathbf{x} \in \mathbb{R}^{n_x}$ and n_u is the dimensionality of inputs $\mathbf{u} \in \mathbb{R}^{n_u}$ and k denotes the discrete time instant. Typically, the state $\mathbf{x}$ of the flying camera includes at least the position of the camera $\mathbf{p}_c \in \mathbb{R}^3$, its velocity $\dot{\mathbf{p}}_c \in \mathbb{R}^2$ and its orientation, i.e. roll, pitch and yaw, as well as the gimbal pitch θ_g and yaw ψ_g angle. We use an unmodified Parrot Bebop2 and include dynamics of the (software) gimbal with resulting dimensionalities $n_x = 10$ and $n_u = 6$ (see Appendix B).

We denote by $\mathcal{X}$ and $\mathcal{U}$ the set of admissible states and inputs. These can be derived from physical limits of the environment and by the internal constraints of the flying camera hardware, e.g. bounds on vertical and horizontal velocities as well as on roll and pitch angles. We obtained the limits from the documentation of the Parrot SDK [Par 2015]. While each quadrotor model has different values of these bounds, in general such bounds exist and can be assumed to be known for a particular model. The trajectory generation method should ensure $\mathbf{x}_k \in \mathcal{X}$ and $\mathbf{u}_k \in \mathcal{U}$ for all k.

3.2 Actor-driven framing objectives

When planning aerial shots, one has to consider both the camera motion and how objects appear in the image. We control this framing directly in the 2D image space via several cost terms similar to [Nägeli et al. 2017] and inspired by Arijon's 'grammar' of film [1976]. Here, we only provide a brief intuition and give detailed derivations for completeness in Appendix C. We allow the user to interactively specify the desired 2D position of the actor in the screen and to specify the relative distance of the subject to the camera via the projected screen size. These two metrics already provide control over the most important framing objectives and can be adjusted in real-time via a GUI. Incorporating further framing objectives is straightforward [Nägeli et al. 2017].

Image space locations are controlled via a quadratic error measure $c_\mathrm{i} : \mathbb{R}^{n_x+6} \rightarrow \mathbb{R}_+$ on the residual ϵ^m between the actual and desired viewing directions of the camera:

$$c_\mathrm{i}(\mathbf{x}, \xi) = \|\epsilon^m\|_{\mathbf{Q}_m} \quad \text{with} \quad \epsilon^m = \frac{\mathbf{r}^c_{cs}}{\|\mathbf{r}^c_{cs}\|} - \frac{\mathbf{r}^c_d}{\|\mathbf{r}^c_d\|}, \tag{1}$$

where $\mathbf{r}^c_{cs}$ is the ray from the camera to the target and $\mathbf{r}^c_d = (\mathbf{m}, 1) \in \mathbb{R}^3$ is the vector through the desired screen position. The pixel coordinates $\mathbf{m}_d$ are given by the camera intrinsics (see App. C).

Similarly, the size of objects in the image is controlled via the quadratic error function $c_\mathrm{d} : \mathbb{R}^7 \rightarrow \mathbb{R}_+$ on the residual between the actual and desired Euclidean distance, σ and σ_d, between the position of the filmed subject $\mathbf{p}_s$ and that of the camera $\mathbf{p}_c$:

$$c_\mathrm{d}(\mathbf{p}_s, \mathbf{p}_c, \sigma_d) = \| \, \|\mathbf{p}_s - \mathbf{p}_c\|_2 - \sigma_d \, \|_{\mathbf{Q}_\sigma} \, . \tag{2}$$

Note that minimizing these costs will require actuation of the robot and hence the actor's natural motions cause the robot to move in order to minimize these costs.

3.3 Subject collision avoidance

To guarantee that the flying camera does not collide with moving subjects, we introduce a collision avoidance constraint. We model the to be avoided region using an ellipse $\mathcal{E}(\mathbf{p}_s)$ around each subject and ask the method to compute quadrotor positions $\mathbf{p}$ that lie *outside* or on its boundary, i.e. the non-convex set

$$\begin{aligned}\bar{\mathcal{E}}(\mathbf{p}_s, \zeta) &:= \mathbb{R}^3 \setminus \mathcal{E}(\mathbf{p}_s) \\ &:= \{\mathbf{p} \in \mathbb{R}^3 \mid (\mathbf{p} - \mathbf{p}_s)^T \mathbf{E} (\mathbf{p} - \mathbf{p}_s) \geq 1 - \zeta\},\end{aligned} \tag{3}$$

is the admissible region for camera positions $\mathbf{p}$ for some positive definite matrix $\mathbf{E}$. The scalar $\zeta \geq 0$ is a slack variable necessary in practice to ensure finding a solution (soft constraint). It can be shown that under sufficiently high penalization of a linear cost such slack variables, the solution of the hard constrained problem, i.e. $\zeta \equiv 0$, is recovered when it exists; otherwise, a plan with minimum deviation will be computed by the optimizer [Kerrigan and Maciejowski 2000]. Such use of slack variables to relax hard constraints is common practice in the MPC literature. We use a 3-4 orders of magnitude higher penalization than for remaining costs. Note that unusually high values of the slack variables, indicating infeasible solutions or exhaustion of the computational budget, can be detected and handled, for example by triggering an emergency landing.

3.4 3D virtual camera rails

The above objectives and constraints *locally* determine the position and orientation of the flying camera in relationship to the subjects in the scene. To control *global* motion, we use *virtual rails*, an analogy to physical camera rails and camera cranes, routinely used on film sets to produce smooth camera motion (see Chapter 22 in [Daniel Arijon 1976]). A virtual rail is a user-specified geometric path, or a set of positions in 3D space, which may be modified interactively. This process is schematically illustrated in Fig. 3. To incorporate virtual rails into our method, we first approximate them by smooth curves and use an MPCC path following approach to generate a feasible path close to the user-defined rails.

Smooth path approximation $\mathbf{s}(\theta)$ of virtual rails: At each time-step we compute a differentiable path $\mathbf{s}(\theta)$ given by a second order approximation of the input. We then seek to follow this smooth path as closely as possible. To do so we want to minimize the projection of the drone's position $\mathbf{p}_q$ onto the path:

$$\theta^* = \arg\min_\theta \|\mathbf{s}(\theta) - \mathbf{p}_q\|. \tag{4}$$

This projection yields the closest distance to the path which is commonly denoted as the contouring error ϵ^c, illustrated Fig. 2, A. However, this projection is not suited as error measure within an optimization formulation since it is an optimization problem itself (over the entire path) and cannot be solved analytically. In

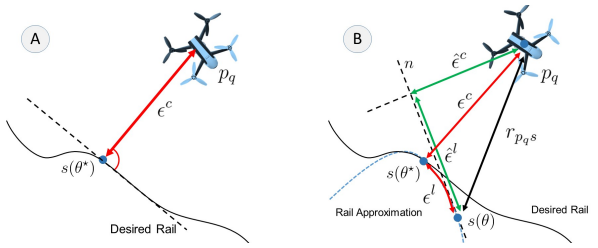

Fig. 2. Exact projection of the quadrotor position onto the virtual rail (A) and it's approximation in the local linearized frame (B).

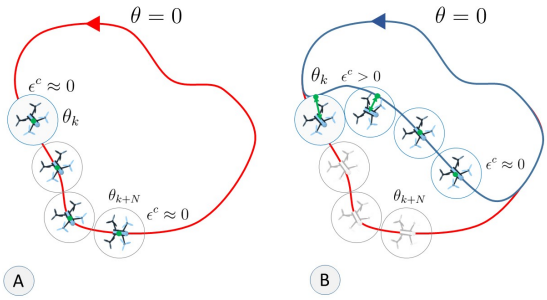

Fig. 3. Online warping of camera reference path. The original reference (red) is changed by the user (blue) at runtime, causing the contour error ϵ^c (in green) to increase for the initial stages of the planning horizon (B). ϵ^c quickly converges to the changed reference path (B).

2D MPCC it is common practice to approximate ϵ^c via separation of contouring and lag-error. The lag-error $\epsilon^l = \int_{\theta^*}^{\theta_k} s(x)dx$ is the integral over the path segment between the desired location on the path θ^* and the location θ found by solving the final MPCC problem (Eq. (8)).

Error measures: Previous work for planar (2D) motion [Lam et al. 2010; Liniger et al. 2015] uses expressions for the contouring and lag error that are not directly transferable to the 3D case. We propose formulations suitable for 3D which also separate lag from contour error. We approximate ϵ^l, ϵ^c by projecting the current MAV position $\mathbf{p}_q$ onto the tangent vector $\mathbf{n}$, with origin at the current path position $\mathbf{s}(\theta)$. The relative vector between $\mathbf{p}_q$ and the tangent point $\mathbf{s}$ can be written as $\mathbf{r}_{\mathbf{p}_q\mathbf{s}} := \mathbf{s}(\theta) - \mathbf{p}_q$. Further the derivative of the path $\mathbf{s}(\theta)$ with respect to the path parameter θ is defined as: $\mathbf{s}' := \frac{\partial \mathbf{s}(\theta)}{\partial \theta}$ which defines the normalized tangent vector $\mathbf{n} = \frac{\mathbf{s}'}{\|\mathbf{s}'\|}$. The approximation of the lag error is then given by:

$$\hat{\epsilon}^l(\mathbf{p}_q, \theta) = \|\mathbf{r}_{\mathbf{p}_q\mathbf{s}}^T \mathbf{n}\|, \tag{5a}$$

The approximations of the contour error is computed by:

$$\hat{\epsilon}^c(\mathbf{p}_q, \theta) = \|\mathbf{r}_{\mathbf{p}_q\mathbf{s}} - \left(\mathbf{r}_{\mathbf{p}_q\mathbf{s}}^T \mathbf{n}\right)\mathbf{n}\|, \tag{5b}$$

With these error measures in place, we define a cost function $c_\mathrm{p} : \mathbb{R}^4 \to \mathbb{R}_+$ which represents the trade-off between path following accuracy and progress $\dot{\theta}$ along the path:

$$c_\mathrm{p}(\mathbf{p}_q, \theta, \dot{\theta}) = \begin{bmatrix} \hat{\epsilon}^l(\mathbf{p}_q, \theta) \\ \hat{\epsilon}^c(\mathbf{p}_q, \theta) \end{bmatrix}^T \mathbf{Q} \begin{bmatrix} \hat{\epsilon}^l(\mathbf{p}_q, \theta) \\ \hat{\epsilon}^c(\mathbf{p}_q, \theta) \end{bmatrix} - \beta\dot{\theta}, \tag{6}$$

where $\mathbf{Q} \in \mathbb{S}_+^2$ is a positive definite weight matrix (typically diagonal) chosen by the user, and $\beta \geq 0$ is a scalar weight such that:

- If $\beta = 0$: the camera is forced to stay on the virtual rail, but its position along the path is free running. In this case the movement along the rail is entirely subject driven.
- If $\beta > 0$: the camera will automatically move along the rail. The movement velocity can be controlled by the user and will be traded off with framing objectives.

Fig. 2, B illustrates that as $\hat{\epsilon}^l(\mathbf{p}_q, \theta)$ becomes small the approximation quality of the contour error increases. In particular when $\hat{\epsilon}^l(\mathbf{p}_q, \theta) \to 0$ then $\hat{\epsilon}^c(\mathbf{p}_q, \theta) \approx \epsilon^c$. We therefore typically chose a high penalty on $\hat{\epsilon}^l(\mathbf{p}_q, \theta)$. For the contouring error we allow some flexibility in order to account for the subject-driven framing objectives and constraints since it might be desirable to deviate locally from the virtual rail in favor of these other objectives (cf. Fig. 3, B). The relative weight of the objectives can be set by the user via an appropriate tuning of the cost function Eq. (6).

3.5 MPCC Formulation

We take a linear combination of the error measures for image location, Eq. (1), and size, Eq. (2), and the path following cost, Eq. (6), to define a stage cost that serves as a performance index for the path generation, following and videography goals of the method:

$$J_k = a_i c_\mathrm{i}(\mathbf{x}_k, \xi_k) + a_d c_\mathrm{d}(\mathbf{p}_{sk}, \mathbf{p}_{ck}, \sigma_d) + a_p c_\mathrm{p}(\mathbf{p}_{qk}, \theta_k, \dot{\theta}_k), \tag{7}$$

where the scalar weight parameters $a_i, a_d, a_p > 0$ can be set interactively to control the (relative) importance of the different terms. The trajectory and control inputs of the drone at each time-step are computed via the solution of the following N-step finite horizon constrained non-linear optimization problem at time instant t:

$$\begin{aligned}
\underset{\mathbf{u},\mathbf{x},\theta,\dot{\theta},\zeta}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \left(J_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k\right) + a_N J_N \quad + \lambda\|\zeta\|_\infty && (8)\\
\text{subject to} \quad & \mathbf{x}_0 = \hat{\mathbf{x}}(t) && \text{(Initial state)}\\
& \theta_0 = \hat{\theta}(t) && \text{(Initial path parameter)}\\
& \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) && \text{(Dynamics)}\\
& \theta_{k+1} = \theta_k + \dot{\theta}_k T_s && \text{(Progress virtual rail)}\\
& \mathbf{p}_{qk} \notin \mathcal{E}(\mathbf{p}_{sk}, \zeta_k)\ \forall s && \text{(Collision avoidance)}\\
& 0 \leq \theta_k \leq L && \text{(Path length)}\\
& \mathbf{x}_k \in \mathcal{X}, && \text{(State constraints)}\\
& \mathbf{u}_k \in \mathcal{U}, && \text{(Input constraints)}\\
& \zeta_k \geq 0, && \text{(Slack positivity)}
\end{aligned}$$

where $\mathbf{R} \in \mathbb{S}_+^{n_u}$ is a positive definite penalty matrix avoiding excessive use of the control inputs. The vector $\hat{\mathbf{x}}(t)$ and the scalar $\hat{\theta}(t)$ denote the (estimated or measured) values of the current states $\mathbf{x}$ and θ, respectively. The scalar T_s is the sampling time. The scalar $a_N > 0$ is a weight parameter used to weight a so-called terminal cost J_N on the final stage. This is common in finite-horizon schemes such as MPCC to mimic long horizons, approximating the infinite

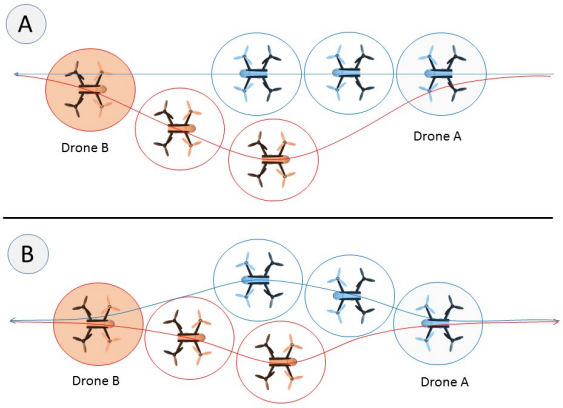

Fig. 4. Schematic of collision between two quadrotors. (A) shows classical priority based collision avoidance where the blue quadrotor has higher priority than the red. (B) shows our solution with iterative sharing of planned trajectories. This results in cooperative collision avoidance.

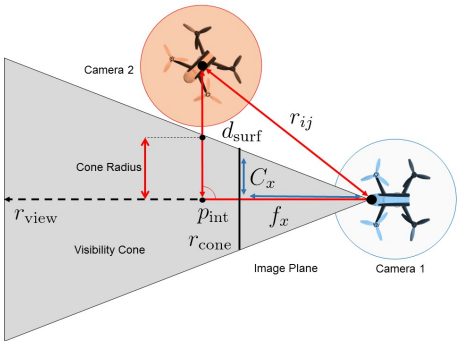

Fig. 5. Schematic of the mutual visibility cost Eq. (10).

horizon solution. Finally, the scalar λ is a penalty parameter for the slack variables associated with the softened obstacle avoidance constraints. The drone is actuated using the optimal inputs from the first step $\mathbf{u}_0$. Importantly, a new trajectory is recomputed at each time-step, taking updated sensor data, rail configurations and user-specified viewpoints into consideration.

4 MULTI-DRONE FLIGHT

In the previous section we discussed the proposed online trajectory planning method for the case of a single quadrotor. In this section we describe additional constraints and costs that allow for the filming of multi-view shots in a single take. For instance, when filming highly dynamic scenes, such as live sports, it is often desirable to provide different views of a subject as it moves through the environment, requiring usage of several cameras to orchestrate views to allow for spontaneous, non-scripted motion.

Collision avoidance with coordination: Further to avoiding collisions with subjects, see Sec. 3.3, each drone now needs to avoid collisions with the other drones that are videographing the scene. This becomes especially critical when filming in cluttered spaces with little room to navigate and when the camera trajectories may overlap and intersect.

In traditional priority planning each robot would avoid only the robots of higher priority, leading to highly suboptimal trajectories (Fig. 4, A), which can conflict with fulfilling the cinematographic objectives. In typical videography scenarios the multiple drones will be centrally controlled or at least have a communication channel. Leveraging this communication channel, our method is sequential consensus, not priority based.

To this end we extend our algorithm to consider Eq. (3) for each drone's future states. In this scheme each drone receives the current plans from all other drones and plans a collision-free trajectory with respect to the complete set of plans. Further, to guarantee safety, we assume that planning is performed sequentially and plans are communicated to all other drones after each planning iteration. While in the first iteration this is equivalent to priority planning, in the subsequent iterations it is not and leads to more cooperative trajectories, illustrated in Fig. 4, B.

Formally, the non-convex set $\bar{\mathcal{E}}(\mathbf{p}_k^j, \zeta^j)$ is the admissible region for drone position $\mathbf{p}_q$ for all time-steps $k \in \{0, N\}$ and for all drones $j \in \{1, M\}$ other than i:

$$\bar{\mathcal{E}}(\mathbf{p}_k^j, \zeta^j) := \mathbb{R}^3 \setminus \mathcal{E}(\mathbf{p}_k^j) \tag{9}$$

where scalar $\zeta^j \geq 0$ again are slack variables ensuring that a solution is found in practice.

Mutual visibility: When filming a multi-view scene it is desirable to reduce visibility of other cameras. Our method can take this into account by extending the stage-cost of Eq. (7) with an additional term c_{v} for each pair of drones, penalizing mutual visibility. The computation of this cost is schematically illustrated in Fig. 5 and details of the derivation can be found in Appendix D. Here we provide only an intuition for the procedure.

We approximate the camera's view frustum by a bounding cone and test for all other drones if their bounding sphere intersects the bounding cone C. For this, we project the relative vector between the drones i and j onto the view direction of drone i to attain the distance to drone j along the viewing direction. We then compute the signed distance d_{surf} from the position of drone j to the cone surface at the intersection point $\mathbf{p}_{\text{int}}$ and normal to the viewing direction. If this distance is positive, then drone j is outside of the viewing cone of drone i. The stage cost is then

$$c_{\text{v}}(\mathbf{x}^i, \mathbf{x}^j) = \begin{cases} Q_v d_{\text{surf}}^2 & \text{if } d_{\text{surf}} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{10}$$

where $\mathbf{x}^i$ is the state of drone i. The cost c_{v} is added to Eq. (7) with a tunable weight Q_v. The resulting behavior is shown in Fig. 6.

MPCC Formulation: Algorithm 2 summarizes the procedure for multiple quadrotors. At each time-step a new trajectory is computed for each drone independently and sequentially, by solving the MPCC problem of Eq. (8). For each neighboring drone, we add: a) the collision constraints of Eq. (9), and b) the mutual visibility cost of Eq. (10). Note that *communicating* motion plans, rather than relying on *estimates*, can enable highly dynamic maneuvers.

Algorithm 2 Multi-drone algorithm

Input: Trajectories $\mathcal{T}_i$ for all drones $i \in \{1, M\}$ at time $t-1$.
Input: Inputs $[S_s, S_c, \mathbf{p}_s, \dot{\mathbf{p}}_s]$ from Algorithm 1.
for $i \in \{1, M\}$ **do**
  for $j \in \{1, M\}, j \neq i$ **do**
    compute collision avoidance constraints w.r.t $\mathcal{T}_j$.
    compute mutual visibility cost w.r.t j.
  end for
  $\mathcal{T}_i \leftarrow$ solve Eq. (8) with new constraints and costs.
end for

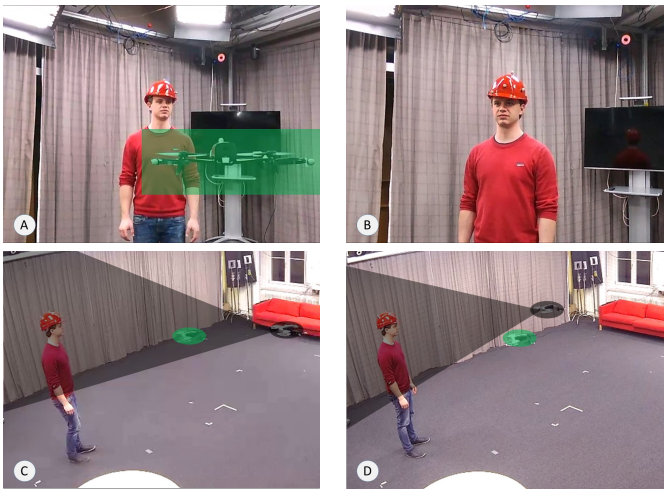

Fig. 6. Influence of penalizing mutual visibility. A and C: single subject is filmed by two drones simultaneously, the first drone (green) is in the field of view of the second drone. B and D: Enabling the mutual visibility cost triggers re-planning of trajectory resulting in unobstructed view with correctly framed subject.

5 EVALUATION AND DISCUSSION

Here we discuss quantitative and qualitative results and experiments conducted to evaluate our method and its components.

5.1 Implementation Details

Our experiments are conducted on a standard desktop PC (Quadcore Intel i7 CPU@3.5 GHz). The subjects and drones are tracked via a Vicon motion capture system for indoors experiments. We solve the MPCC problem via the FORCES Pro [Domahidi and Jerez 2016; Domahidi et al. 2012] software which generates fast solver code, exploiting the special structure in the non-linear program (NLP).

Quadrotor hardware: We use unmodified Parrot Bebop2 quadrotors in all our experiments with an integrated electronic gimbal. All communication between the drones and the host PC is handled via ROS [Quigley et al. 2009] and we directly send the control inputs from the first time-step $\mathbf{u}_0$ computed in Eq. (8), *without* an additional feedback controller for trajectory tracking on the drone.

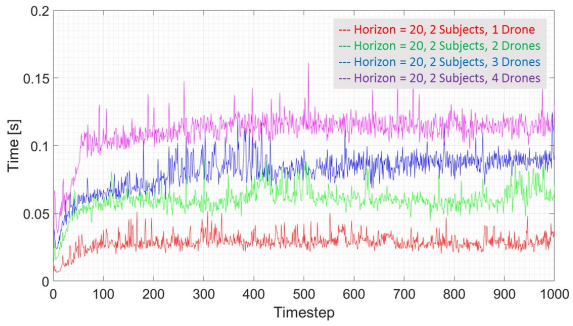

Fig. 7. Solve times for horizon length $N = 20$ and 2 subjects with number of drones varying from 1 - 4.

Initialization: The problem of Eq. (8) is non-convex and therefore initialization is a concern. We initialize the solver with the solution vector computed at the previous time-step, perturbed by random noise. Generally speaking, the method is robust to initialization and we did not observe drastic changes in solve time even if the initialization is drastically perturbed.

5.2 Quantitative and qualitative experiments

Computational performance: To assess the computational performance of the method we record overall solve time of the method (Algorithm 1 and 2). We use fixed horizon length $N = 20$ and a fixed number of two subjects and vary the number of drones from 1 - 4. During the experiment we use framing and mutual visibility cost terms and enable collision avoidance constraints. Importantly the rails used in this last experiment are designed to force collision avoidance maneuvers between the drones. Fig. 7 shows that the computational cost grows linearly with the number of drones. This is expected due to our sequential planning approach. The collision avoidance and visibility optimization does not yield significant overheads when adding further drones.

Multi-view cinematography: Fig. 6 shows the impact of penalizing mutual visibility in multi-view scenarios. The three frames are taken without and with the cost active, resulting in an avoidance maneuver of the drone. This setup forces the drone to 'warp' the input trajectory to fulfil all cinematographic constraints.

5.3 Preliminary expert feedback

Finally we invited a trained camera operator from the local university's film program to assess the overall utility of our approach. The expert designed a number of multi-view shots with our system. While our work is mostly algorithmic and does not have a sophisticated user interface at this point, the expert was still able to plan and execute a number of challenging shots with receiving only very little instructions and no more than 10 minutes training. The resulting shots have been included in the accompanying video figure as so-called real-time cuts (i.e., a single video sequence containing views from multiple cameras off the same duration as the individual clips). Note that these shots include aspects that would

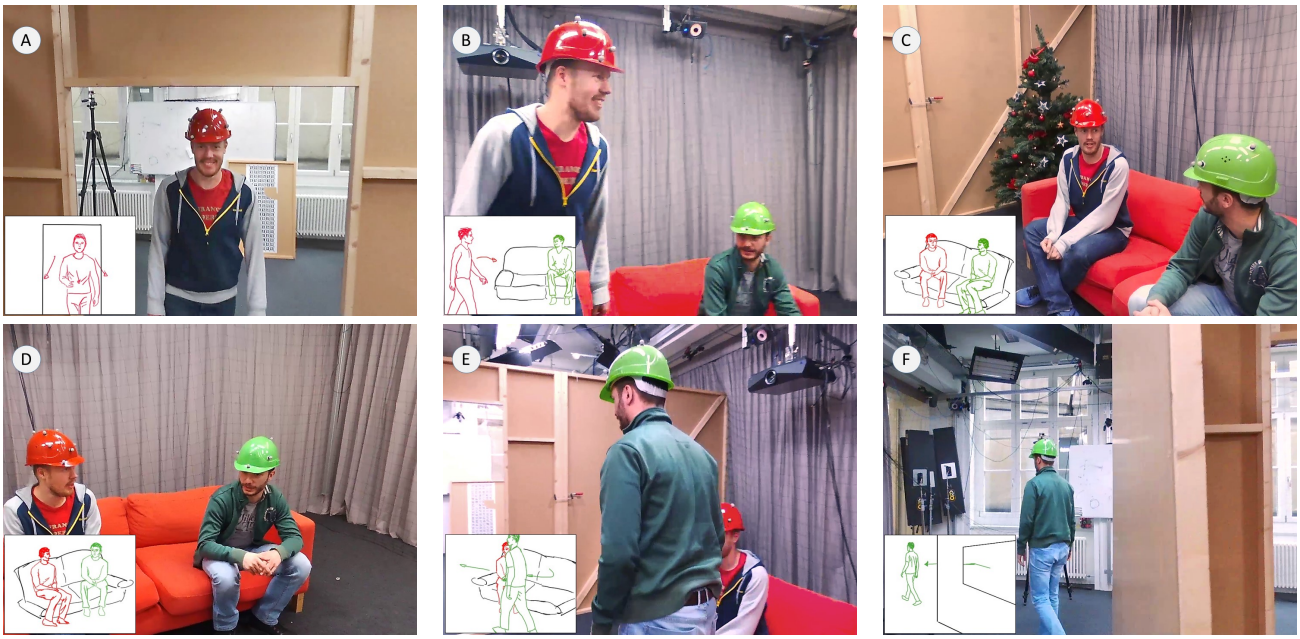

Fig. 8. Figures A - F: Multi-view, multi-person shot, transcribed form story board (insets). Two drones enter and leave the buildin to frame the subjects. The user interactively refocuses the subjects during filming. Full sequence as real-time cut in the accompanying video.

be difficult or impossible to achieve with traditional camera cranes or dollies, for example entering and leaving buildings through doors and windows and positioning of multiple cameras in a tight space with several moving subjects.

Multi-view real-time shot: Fig. 8 illustrates a shot entailing two flying cameras and two actors that move into and out of a building on a simulated film set. The shot was loosely defined using a storyboard (shown in the insets), which is then transcribed into virtual rails, constraining the camera motion. In this case the drone velocities are entirely driven by subject motion. Note that the subject focus and framing is changed interactively by the user at runtime.

Discussion: The expert user was able to design several shots of which we included some. Overall the expert felt that the approach fits well into the practice of filming and that it drastically reduces the complexity of a number of shots in dynamic environments, an area in which aerial videography was previously not applicable.

Our expert also provided a number of interesting ideas for future work including requirements for the user interface and the control algorithm itself. Foremost, the user would have liked to be able to have additional control over the exact framing or in other words would have liked to manually refine the yaw and pitch of the camera on top of computed control inputs. This idea is compatible with the proposed method but is left for future work. Other interesting ideas include being able to control the actual camera parameters such as depth of field and focus points interactively and to incorporate stabilization and smoothing of the optical flow.

While not explicitly mentioned in our evaluation it became clear that there is also an interesting opportunity to optimize camera placement and virtual rails more globally, subject to a high-level script or storyboard. In other words to provide a domain-specific language to make the method more usable by non-technical users.

6 CONCLUSIONS

In this paper we proposed a method for the real-time generation of multi-drone aerial cinematography motion plans. Our proposed method formulates the motion plan generation and tracking problems as a *joint* real-time receding horizon optimization problem. The algorithm respects high-level user goals as well as possible *and* ensures physical feasibility of the resulting plans at every time-step. Importantly, the real-time nature of the method allows for incorporation of feedback and dynamic constraints, enabling the planning of collision-free paths in cluttered environments with moving actors and multiple drones. We have evaluated our method in a number of quantitative and qualitative experiments including single and multi-view shots in dynamic environments.

7 ACKNOWLEDGEMENTS

We are grateful to Christoph Gebhardt for composing the video and Cécile Edwards-Rietmann for narrating it. Velko Vechev was of great help with the figures. We thank Christina Welter and Stefan Stevsic for invaluable feedback and participation in our experiments. A special thanks goes to Edouard Leurent, Jean-Baptiste Dubois, Bertrand Djavan and Mathieu Babel from Parrot and to Andreas Hempel from embotech for their technical support.

REFERENCES

2015. 3DR Solo. (2015). http://3drobotics.com/solo.
2015. APM Autopilot Suite. (2015). http://ardupilot.com.
2015. DJI Ground Station. (2015). http://www.dji.com/product/pc-ground-station.
2015. Parrot SDK. (2015). http://developer.parrot.com/.

2015. VC Technology Litchi Tool. (2015). https://flylitchi.com/.
A Pedro Aguiar, João P Hespanha, and Petar V Kokotović. 2008. Performance limitations in reference tracking and path following for nonlinear systems. *Automatica* 44, 3 (2008), 598–610.
Javier Alonso-Mora, Tobias Naegeli, Roland Siegwart, and Paul Beardsley. 2015. Collision Avoidance for Aerial Vehicles in Multi-agent Scenarios. *Auton. Robot.* (Jan. 2015).
John T Betts. 2010. *Practical methods for optimal control and estimation using nonlinear programming*. Vol. 19. Siam.
Adam Bry, Charles Richter, Abraham Bachrach, and Nicholas Roy. 2015. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal of Robotics Research* 34, 7 (2015), 969–1002.
Yaobin Chen, Stanley Y-P Chien, and ALAN A DESROCHERS. 1992. General structure of time-optimal control of robotic manipulators moving along prescribed paths. *Internat. J. Control* 56, 4 (1992), 767–782.
Marc Christie, Éric Languénou, and Laurent Granvilliers. 2002. *Modeling Camera Control with Constrained Hypertubes*. Springer Berlin Heidelberg, Berlin, Heidelberg, 618–632. https://doi.org/10.1007/3-540-46135-3_41
Marc Christie, Patrick Olivier, and Jean-Marie Normand. 2008. Camera Control in Computer Graphics. *Computer Graphics Forum* 27, 8 (Dec. 2008), 2197–2218.
Daniel Arijon. 1976. *Grammar of the film language*. https://scholar.google.ch/citations?view
Alexander Domahidi and Juan Jerez. 2016. FORCES Pro: code generation for embedded optimization. (September 2016). https://www.embotech.com/FORCES-Pro.
Alexander Domahidi, Aldo U Zgraggen, Melanie N Zeilinger, Manfred Morari, and Colin N Jones. 2012. Efficient interior point methods for multistage problems arising in receding horizon control. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 668–674.
Steven M. Drucker and David Zeltzer. 1994. Intelligent Camera Control in a Virtual Environment. In *In Proceedings of Graphics Interface '94*. 190–199.
Timm Faulwasser, Benjamin Kern, and Rolf Findeisen. 2009. Model predictive path-following for constrained nonlinear systems. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. IEEE, 8642–8647.
Quentin Galvane, Julien Fleureau, Francois-Louis Tariolle, and Philippe Guillotel. 2016. Automated Cinematography with Unmanned Aerial Vehicles. In *Eurographics Workshop on Intelligent Cinematography and Editing*. The Eurographics Association.
Christoph Gebhardt, Benjamin Hepp, Tobias Naegeli, Stefan Stevsic, and Otmar Hilliges. 2016. Airways: Optimization-based Planning of Quadrotor Trajectories according to High-Level User Goals. In *SIGCHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA.
T. Geijtenbeek and N. Pronost. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Comput. Graph. Forum* 31, 8 (Dec. 2012), 2492–2515. https://doi.org/10.1111/j.1467-8659.2012.03189.x
Michael Gleicher and Andrew Witkin. 1992. Through-the-lens camera control. In *ACM SIGGRAPH Computer Graphics*, Vol. 26. ACM, 331–340.
Niels Joubert, L. E. Jane, Dan B. Goldman, Floraine Berthouzoz, Mike Roberts, James A. Landay, and Pat Hanrahan. 2016. Towards a Drone Cinematographer: Guiding Quadrotor Cameras using Visual Composition Principles. *CoRR* abs/1610.01691 (2016). http://arxiv.org/abs/1610.01691
Niels Joubert, Mike Roberts, Anh Truong, Floraine Berthouzoz, and Pat Hanrahan. 2015. An Interactive Tool for Designing Quadrotor Camera Shots. *ACM Transactions on Graphics (SIGGRAPH Asia 2015)* (2015).
Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30, 7 (June 2011), 846–894.
Eric C Kerrigan and Jan M Maciejowski. 2000. Soft constraints and exact penalty functions in model predictive control. In *Control 2000 Conference, Cambridge*.
Denise Lam, Chris Manzie, and Malcolm Good. 2010. Model predictive contouring control. In *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 6137–6142.
Alexander Liniger, Alexander Domahidi, and Manfred Morari. 2015. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods* 36, 5 (2015), 628–647. https://doi.org/10.1002/oca.2123
C Lino and M Christie. 2015. Intuitive and efficient camera control with the toric space. *ACM Transactions on Graphics (TOG)* (2015).
Christophe Lino, Marc Christie, Roberto Ranon, and William Bares. 2011. The Director's Lens: An Intelligent Assistant for Virtual Cinematography. In *Proceedings of the 19th ACM International Conference on Multimedia (MM '11)*. ACM, New York, NY, USA, 323–332. https://doi.org/10.1145/2072298.2072341
Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev. 2013. Path planning for non-circular micro aerial vehicles in constrained environments. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 3933–3940.
Daniel Mellinger and Vijay Kumar. 2011. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2520–2525.
Mark W Mueller and Raffaello D'Andrea. 2013. A model predictive controller for quadrocopter state interception. In *Control Conference (ECC), 2013 European*. IEEE, 1383–1389.
Tobias Nägeli, Javier Alonso-Mora, Alexander Domahidi, Daniela Rus, and Otmar Hilliges. 2017. Real-time Motion Planning for Aerial Videography with Dynamic Obstacle Avoidance and Viewpoint Optimization. *IEEE Robotics and Automation Letters* (2017).
Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. 2009. ROS: an open-source Robot Operating System. In *IEEE ICRA Workshop on Open Source Software*.
Mike Roberts and Pat Hanrahan. 2016. Generating Dynamically Feasible Trajectories for Quadrotor Cameras. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)* 35, 4 (2016).
Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F Cohen. 1996. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 147–154.
Jeffery B Saunders, Brandon Call, Andrew Curtis, Randal W Beard, and Timothy W McLain. 2005. Static and dynamic obstacle avoidance in miniature air vehicles. *AIAA Infotech@ Aerospace* 96 (2005).
Andrew Witkin and Michael Kass. 1988. Spacetime constraints. *ACM Siggraph Computer Graphics* 22, 4 (1988), 159–168.

A NOTATION

Due to the many degrees-of-freedom of the problem discussed in this paper, the resulting notation is rather verbose. For completeness and reproducibility of our method we provide a summary of the notation used in the paper in Table 1.

Symbol	Description
$\mathbf{p}_q$, $\dot{\mathbf{p}}_q$	Quad position and velocity
v_z	Quad velocity in z direction
ω_{q_z}	Quad angular velocity around body-z
$\mathbf{p}_{q_i}$	Position of quad i
Φ_q,Θ_q,ψ_q	Quad roll, pitch, yaw
ϕ_q,θ_q	Quad desired roll, pitch
$\mathbf{R}_{\psi_q}(\psi_q)$	Quad yaw matrix
$\mathbf{p}_c$, $\dot{\mathbf{p}}_c$	Camera position and velocity
$\mathbf{p}_{c\,i}$	Position of camera i
θ_g, ψ_g	Gimbal pitch, yaw
$\mathbf{R}_{\theta_g}(\theta_g)$, $\mathbf{R}_{\psi_g}(\psi_g)$	Gimbal pitch, yaw matrix
ω_{θ_g}	Gimbal pitch rate
$\mathbf{x}$,$\mathbf{u}$	Quad states & inputs
$\mathcal{T}$	Planned trajectory
ϵ^{i}	Projection error in image space
$\mathbf{r}_k^{ij}$	Relative vector between quad i and j
$\mathbf{r}_{ij}$	Relative vector between camera i and j
$\mathbf{p}_s$, $\dot{\mathbf{p}}_s$	Position and velocity subject
ξ	State of subject =($\mathbf{p}_s$,$\dot{\mathbf{p}}_s$)
$\mathcal{E}$	Ellipsoid capturing subjects for avoidance
$\mathbf{E}$	Matrix defining collision ellipsoid
ζ	Slacks used for collision ellipsoid
C	Cone of the actual field of view
N	Prediction horizon in MPCC problem
T_s	Sampling time

Table 1. Summary of notation used in the body of the paper

B QUADROTOR DYNAMICS MODEL

The state of the quadrotor is given by its position $\mathbf{p}_q \in \mathbb{R}^3$, its velocity $\dot{\mathbf{p}}_q \in \mathbb{R}^2$ and its orientation, i.e. roll Φ_q, pitch Θ_q and yaw ψ_q. The camera is attached to the robot via a pan-tilt gimbal (in case of the Bebop this is a software gimbal). The state of the camera is given by its position $\mathbf{p}_c$ (rigid body transformation from $\mathbf{p}_q$), the x and y velocity $\dot{\mathbf{p}}_q$ and a separate pitch and yaw angle Φ_q, Θ_q as well as the gimbal states θ_g, ψ_g. We denote the state of the system, consisting of quadrotor and gimbal, by

$$\mathbf{x} = [\mathbf{p}_q, \dot{\mathbf{p}}_q, \Phi_q, \Theta_q, \psi_q, \theta_g, \psi_g] \in \mathbb{R}^{10}. \tag{11}$$

Following the Parrot Bebop 2 SDK, the control inputs to the system are given by the vector

$$\mathbf{u} = [v_z, \phi_q, \theta_q, \omega_{\psi_q}, \omega_{\theta_g}, \omega_{\psi_g}] \in \mathbb{R}^6, \tag{12}$$

where v_z is the velocity of the quadrotor in the body-z axis, ϕ_q and θ_q are the desired roll and pitch angles of the quadrotor, respectively, ω_{ψ_q} is the angular speed around the body-z axis and $\omega_{\theta_g}, \omega_{\psi_g}$ are the pitch and yaw rates of the camera gimbal. The horizontal velocities are not directly controlled.

We employ a first order low-pass Euler approximation of the quadrotor dynamics, as follows. The translational dynamics are then given by $\dot{\mathbf{p}}_q = [\dot{\mathbf{p}}_{qx,y}, v_z]$ and $\ddot{\mathbf{p}}_q = [\ddot{\mathbf{p}}_{qx,y}, 0]$, with

$$\ddot{\mathbf{p}}_{qx,y} = \mathbf{R}_{\psi_q}(\psi_q) \begin{bmatrix} -tan(\Phi_q) \\ tan(\Theta_q) \end{bmatrix} g - C\dot{\mathbf{p}}_{qx,y}, \tag{13}$$

where $g = 9.81 \frac{m}{s^2}$ is the earth's gravity, $\mathbf{R}_{\psi_q}(\psi_q) \in SO(2)$ is the rotation matrix only containing the yaw rotation of the quadrotor and C is the drag coefficient at low speeds. The rotational dynamics of the quadrotor are

$$\dot{\Phi}_q = \tau_a(\phi_q - \Phi_q), \;\; \dot{\Theta}_q = \tau_a(\theta_q - \Theta_q) \;\text{ and }\; \dot{\psi}_q = \omega_{\psi_q}, \tag{14}$$

and the gimbal pitch rate is given by $\dot{\theta}_g = \omega_{\theta_g}$.

C FRAMING OBJECTIVES

We adapt rules from the 'grammar' of film [Daniel Arijon 1976]. In particular, we are interested in framing objectives – that is formal rules that specify how objects should appear on the screen. Here we summarize the derivation of the framing objective for the position of the subject on the camera screen, which is used in our optimization framework. For further details and additional cost terms for viewing angle and projection size, we refer to [Nägeli et al. 2017]. Each of cost terms is computed for each state $k \leq N$ of the planning horizon and for each subject $i \leq K$. For simplicity of exposition, the derivation of the costs is described for a single subject and time-step. We define the vector $\mathbf{r}_{cs}$ which is the relative vector between the subject and the camera as well as the rotation into the camera frame of $\mathbf{r}_{cs}$ which is denoted as $\mathbf{r}^c_{cs}$:

$$\mathbf{r}_{cs} = \mathbf{p}_s - \mathbf{p}_c \quad \text{and} \quad \mathbf{r}^c_{cs} = \mathbf{R}_c(\mathbf{x})\mathbf{r}_{cs},$$

where $\mathbf{R}_c(\mathbf{x})$ is the rotation matrix induced by the orientation of the camera. Given the desired position $[\mu_{x\,d}, \mu_{y\,d}]$ of each target's projection on the image plane, the desired pixel coordinates $\mathbf{m}_d$ are computed via the camera intrinsics with focal point $[C_x, C_y]$ and focal length $[f_x, f_y]$:

$$\mathbf{m}_d = \begin{bmatrix} (\mu_{x\,d} - C_x)\, f_x \\ (\mu_{x\,d} - C_y)\, f_y \end{bmatrix} \quad \text{with } [\mu_{x\,d}, \mu_{y\,d}] \in [0, 2\max(C_x, C_y)]^2.$$

Consider the vector $\mathbf{r}^c_d = [\mathbf{m}_d, 1]^T \in \mathbb{R}^3$ pointing from the camera center through the desired pixel coordinates $\mathbf{m}_d$. We compute the quadratic image space location cost $c_{\text{i}}(\mathbf{x}, \xi)$ using the residual given by the difference between the ray $\mathbf{r}^c_{cs}$ from the camera to the target and the desired direction $\mathbf{r}^c{}_d$,

$$c_{\text{i}}(\mathbf{x}, \xi) = \|\epsilon^m\|_{Q_m} \quad \text{with} \quad \epsilon^m = \frac{\mathbf{r}^c_{cs}}{\|\mathbf{r}^c_{cs}\|} - \frac{\mathbf{r}^c_d}{\|\mathbf{r}^c_d\|}$$

D MUTUAL VISIBILITY

To ensure that no other camera is in the field of view, for camera i we approximate its view frustum by a bounding cone C, defined by the image plane P, the focal length $[f_x, f_y]$, the focal point $[C_x, C_y]$, the center of the camera $\mathbf{p}_c$ and its orientation $\mathbf{R}_c(\mathbf{x})$, see Fig. 5. For each other drone j, we test if it is inside the cone C. We first compute the view direction $\mathbf{r}_{\text{view}} = \mathbf{R}_c(\mathbf{x})[0, 0, 1]^T$ of the camera and the intersection point $\mathbf{p}_{\text{int}} = c_{\text{int}}\mathbf{r}_{\text{view}}$, where $c_{\text{int}} = \mathbf{r}^T_{ij}\mathbf{r}_{\text{view}}$ and $\mathbf{r}_{ij} = \mathbf{p}_q{}^j - \mathbf{p}_c$ is the relative vector from the camera to drone j. To determine if drone j is outside of the view cone C, we compute the distance d_{surf} from the drone to the cone surface at the intersection point $\mathbf{p}_{\text{int}}$,

$$d_{\text{surf}} = \|\mathbf{r}_{ji} - \mathbf{p}^T_{\text{int}}\mathbf{r}_{\text{view}}\| - r_{\text{cone}} \;\text{ with }\; r_{\text{cone}} = \frac{\max(C_x, C_y)}{\max(f_x, f_y)} c_{\text{int}}$$

If $d_{\text{surf}} > 0$ then drone j is outside of the viewing cone and if $d_{\text{surf}} \leq 0$, then drone j is visible in the camera image of drone i.

To minimize mutual visibility, we define, for each pair of drones at states $\mathbf{x}$ and $\mathbf{x}^j$, the cost term

$$c_{\text{v}}(\mathbf{x}, \mathbf{x}^j) = \begin{cases} Q_v d^2_{\text{surf}} & \text{if } d_{\text{surf}} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{15}$$

where $\mathbf{Q}_v$ is a tunable weight.