DISS. ETH NO. 25441

Intelligent Drone Cinematography

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH Zürich
(Dr. sc. ETH Zürich)

presented by

Tobias Nägeli

MSc ETH in Electrical Engineering
ETH Zürich

born 01.09.1986
citizen of Bülach ZH,
Switzerland

accepted on the recommendation of

Prof. Otmar Hilliges, Supervisor
Prof. Daniela Rus, Co-supervisor
Prof. Javier Alonso-Mora, Co-supervisor
Prof. Markus Gross, Co-examiner

2018

Abstract

Smooth shots, spectacular images and photo-realistic animation scenes are a crucial demand in the media and entertainment industry. Creating this content for sporting events or action movies, however, still requires heavy equipment such as dollies, cranes or huge camera setups. Whereas for example in lighting there have been huge achievements towards small and light-weight devices, for stabilization systems, the industry still relies on the same heavy tools as decades ago. Drones can be equipped with high-end cameras and serve as flying camera platforms. They can provide smooth camera shots, fly lower to the ground, provide stunning aerial filming content and even serve as flying sensor platforms to provide precise motion estimates of the actors. However, using drones for filming, has two major issues: i) Filming often requires more than one take of the same shot. Therefore, drones need to be able to fly the exact same path several times. This is hardly possible with a manually controlled drone, and if so, it requires hours and hours of training. ii) Computer-controlled drones may solve this problem. However, their current positioning systems are not reliable enough yet, due to the lack of precision technology. In this thesis, we present new methods to solve these two issues which allows to use drones as a compelling replacement of classical filming equipment.

We explored new methods and algorithms for robust, fast and accurate real-time intelligent cinematography. This includes keyframe based filming, collision avoidance, virtual dolly movements and human

pose estimation. To reach this we use advanced and very fast nonlinear recursive estimation techniques in combination with real-time nonlinear programming for model predictive control.

In the first part of this thesis, we present methods to provide accurate vision and inertial sensor based position estimates for flying drones. The algorithm is based on a very small baseline stereo camera and a separation between feature tracking and position estimation. In addition, we present an extension to the proposed algorithm to reach the highest level of coupling by directly estimating the position using pixel intensity measurements.

In the second part of the thesis we introduce the concept of intelligent real-time multi-drone Cinematography. We present a method for viewpoint optimization of a single drone and multiple actors. The user can define how the resulting video should look like while the algorithm controls the drone in real-time to provide the desired image content. In addition, we present a method for subject collision avoidance in order to fly safe around the actors. We also present a technology to emulate camera cranes and dollies using drones. We present how multiple drones can be used to capture a scene in a collaborative way. We also present the algorithmic foundations to minimize mutual visibility, which means the single cameras do not film each other.

In the last part of the thesis, we combine the methods presented above and develop a method for real-time drone-based human motion estimation. We show the working system in a number of compelling experiments independent of the environment.

Zusammenfassung

Ästhetisch ansprechende und spektakuläre Kamerabilder sowie fotorealistische Animationsszenen sind ein zentrales Element in der Medien- und Entertainmentbranche. Die Erstellung dieses Inhalts für Sportveranstaltungen oder Action-Filme erfordert jedoch immer noch schwere Ausrüstung wie Dollies, Kräne oder riesige Kameraaufbauten. Während beispielsweise in der Beleuchtung grosse Fortschritte gemacht wurden in Richtung kleinerem und leichterem Equipment, setzt die Kamerabranche immer noch auf die gleichen schweren Werkzeuge wie vor Jahrzehnten. Drohnen können mit High-End-Kameras ausgestattet werden und dienen als fliegende Kameraplattformen. Sie können sanfte Kamerafahrten bieten, nahe dem Boden fliegen, atemberaubende Luftaufnahmen liefern und sogar als fliegende Sensor-Plattform dienen, um eine genaue Bewegungsanalyse der Schauspieler zu liefern. Um Drohnen zum Filmen zu verwenden gibt es jedoch zwei Hauptprobleme: i) In einem Film werden oft mehrere Aufnahmen der gleichen Szene benötigt. Daher müssen Drohnen den exakt gleichen Pfad mehrmals fliegen können. Dies ist mit einer manuell gesteuerten Drohne kaum möglich oder es erfordert stundenlanges Training. ii) Computergesteuerte Drohnen können dieses Problem lösen. Jedoch sind die derzeitigen Positionierungssysteme aufgrund fehlender Präzisionstechnologie nicht zuverlässig genug.

In dieser Arbeit stellen wir neue Methoden zur Lösung dieser beiden Probleme vor. Diese ermöglichen es, Drohnen als überzeugenden Ersatz für klassische Filmausrüstung zu verwenden.

Wir erforschten neue Methoden und Algorithmen für robuste, schnelle und genaue intelligente Kameraführung in Echtzeit. Dazu gehören Keyframe-basierte Filmaufnahmen, Kollisionsvermeidung, virtuelle Dolly-Bewegungen und Echtzeit-Posen-Schätzung von Schauspielern. Um dies zu erreichen, verwenden wir fortgeschrittene und sehr schnelle nichtlineare rekursive Schätztechniken in Kombination mit nichtlinearer Echtzeitprogrammierung zur modellprädiktiven Steuerung.

Im ersten Teil dieser Arbeit stellen wir Methoden vor, um genaue Positions- und Inertiasensor-basierte Positionsschätzungen für fliegende Drohnen zu erzeugen. Der Algorithmus basiert auf einer sehr kleinen Stereokamera. Zusätzlich stellen wir eine Erweiterung des vorgestellten Algorithmus vor, bei dem wir eine maximale Integration erreichen, indem wir die Position anhand von Pixelintensitätsmessungen direkt schätzen.

Im zweiten Teil der Arbeit stellen wir das Konzept der intelligenten Echtzeit-Multidrohnenfilmproduktion vor. Wir präsentieren eine Methode zur Optimierung des Blickwinkels einer einzelnen Drohne und mehrerer Schauspieler. Der Benutzer kann festlegen, wie das resultierende Video aussehen soll. Der Algorithmus steuert die Drohne in Echtzeit, um den gewünschten Bildausschnitt zu erzeugen. Darüber hinaus stellen wir eine Methode zur Kollisionsvermeidung vor, welche es erlaubt sicher um die Schauspieler zu fliegen. Weiter präsentieren wir Methoden, um Kamerakran und Dollies mit Drohnen zu emulieren. Wir zeigen, wie mehrere Drohnen verwendet werden können, um eine Szene mit mehreren Kameras zu filmen. Wir stellen auch die algorithmischen Grundlagen vor, die es braucht, so dass die einzelnen Kameras sich nicht gegenseitig filmen.

Im letzten Teil der Arbeit kombinieren wir die oben aufgeführten Algorithmen und entwickelten eine Methode für Echtzeit-Drohnenbasierte Bewegungsschätzung für Menschen. Wir zeigen das funktio-

nierende System in einer Reihe von Experimenten, unabhängig von der Umgebung.

Acknowledgement

First of all, I am grateful to my advisor, Prof. Otmar Hilliges, for his impeccable guidance, for convincing me to start in his group - as first PhD student - and for giving me opportunities to do my research with the highest possible degree of freedom you can think. These projects gave me the chance to work with many researchers at ETH but also in collaboration with other groups. In addition, a special thank goes to Prof. Daniela Rus for having the incredible possibility to do a research internship at the CSAIL croup at MIT and to Markus Gross for providing me feedback on my thesis. I also want to thank especially to my co-advisor and friend Prof. Javier Alonso-Mora. Without his brilliant ideas, many publications in this thesis would not have been possible. We discussed in endless calls new ideas and concepts. I wish also to tank our great AIT group. It was great to see the group growing as one of the first members. A very special thank goes to Benni for the disputes about manifolds, Fabrizio for teaching us about the best Pizza in town, Jie as our great guide in China and Stephan to be a fantastic office partner. A special thank also goes to Christioph for helping me all the time in the last minute to do magic and cut an incredible submission video and Wookie for providing the voice-overs. In addition, a special thank goes to Manuel, Emre, Adrian and Velko. A special thanks goes to Edouard Leurent, Jean-Baptiste Dubois, Bertrand Djavan and Mathieu Babel from Parrot for their great help and

support. I am also very thankful to Alexander Domahidi and Andreas Hempel from embotech for their incredible technical support.

I also want to thank the great individuals from our research lab and the former PIXHAWK team - consisting of students and PhD students - for all the fruitful discussions. Especially to Lorenz Meier for his support, Petri Tanskanen for the great discussions on all visual and camera problems you can think off, Dominik Honegger for being a great shared-flat partner and being a great opponent for discussions over lunch. Further to all my former students, especially Lukas Meier, Nicolas de Palézieux, Silvan Pluess and Samuel Oberholzer. Without their help, many of the contributions of this theses where not have been possible. Also a special thank goes to Lisa Sturzenegger for her very valuable linguistic advice. I consider myself very lucky to have worked with so many people together. I would like to thank all of them and hope I do not miss anyone: Wilko Schwarting, Alex Wallar, Andy Spielberg, Brandon Araki, Cenk Baykal, Cristian Vasile, Jeffery I. S. Lipton, Liam Paull, Lucas Liebwein, Robert Katzschmann, Shugang Li, Changil Kim, and Martin Rutschman.

Last but not least, I'm very thankful to my family for their unconditional support. Especially in memory of my father: It was a crazy attempt to explain the difference between current and voltage to a five-year-old Tobi during a hike. Although he thought I got it, he guided me on the right way by suggesting studying at ETH electrical engineering. Now, 31 years later, I think I finally got the difference now.

Contents

List of Figures

List of Algorithms

Chapter 1.

Introduction

1.1. Motivation

Figure 1.1.: *James bond scene from the movie Skyfall (left). A behind the scene picture from the same scene (right).*

Nothing beats the thrill of watching a good movie. Films are able to tell fascinating stories with the help of stunning camera footage, transporting us to places we have never been before and allowing us to explore the perspectives of people quite different from ourselves.

To create these unique impressions, incredible viewpoints and smooth camera shots are required. Indeed, these elements have been in high demand since the very first days of cinema.

Filming a smooth camera shot still requires heavy camera equipment such as dollies or cranes, however handling such heavy equipment is very time-consuming and cost-intensive. For example, in the movie *Skyfall*, a camera crane and whole film crew were put on a moving train, as can be seen in Fig. 1.3 (left). Working with such bulky and inflexible equipment is not only expensive, but also raises safety issues. Therefore, producers began to actively search for other solutions, as the owner and founder of Singorell Productions explained:

> "Setting up and handling a dolly is very time-consuming and cost-intensive. I rarely use them and prefer to look for another solution."
> **Riccardo Signorell Owner, Producer Signorell Productions**[1]

To overcome the problem of such high-cost setups – as well as to generate completely new visual content – the film industry started to animate such scenes. Due to the technical progress of computer graphics in recent years, these movies have started to become more and more photorealistic. Fig. 1.3 (right).

Computer-animated content provides us with many more possibilities and far greater flexibility. It is not only possible to render, for example, a special effect in a classical feature film, but also to completely animate a whole movie. However, to produce such scenes, an incredible amount of knowledge and time is needed[2].

To cut down on costs, studios are using so-called motion-capturing technology, which draws on people's natural movements. This involves placing several infrared cameras around a film set (as shown in Fig. 7.9), which detect reflective markers that are placed on actors. This allows the cameras to capture their motions in real time. This not

[1] http://signorell.com/

[2] http://www.ign.com/articles/2014/07/11/a-brief-history-of-motion-capture-in-the-movies

only saves time and money, but also results in animated characters with natural, human-like mobility, as was the case in the movie *Dawn of the Planet of the Apes*, shown in Fig. 1.3.

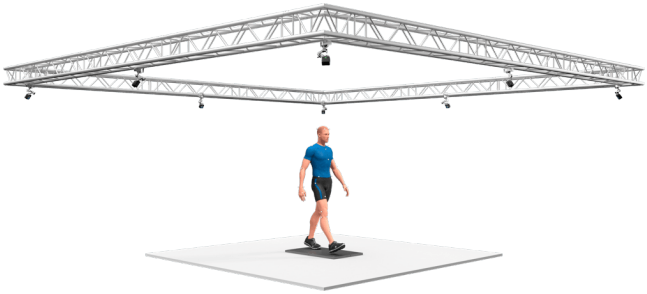


Figure 1.2.: *A person walking in a motion capturing system. Several cameras at the ceiling are filming the person's motion from different perspectives. Using all image information allows to reconstruct the human motion in real-time and to animate a character.*

Although motion capturing solves many problems, it also has several disadvantages. For example, specialized hardware is needed and the costs for even small productions are immense, meaning it is not practical to use such systems on a large scale.

Quick setup changes or motion capturing of long-distance and dynamic sequences are thus extremely difficult to achieve, and are in any case only possible with huge time and cost overheads.

The use of bulky, immobile and heavy camera equipment is therefore a general problem in both classical filming as well in movies where motion-capturing technology is used. To overcome this problem, in recent years drones have become increasingly popular. Drones can be equipped with high-end movie or even infrared cameras, and they can fly low against the ground and provide smooth camera shots. Not only could they potentially replace heavy equipment such as dollies or cranes, they could also serve as flying motion-capturing platforms

to reduce camera overheads and allow motion capturing independent of a fixed infrastructure. However, film sets are highly dynamic spaces involving many people, objects and environmental challenges. This means using drones for this purpose requires very precise flights, particularly where dynamic and complicated processes are concerned.

Figure 1.3.: *Behind the scene of the James Bond film Skyfall (left) and Dawn of the Planet of the Apes (right). Motion capturing was used to capture the human motion in order to animate the virtual apes.*

1.2. Research Opportunities

Our vision is to transfer the boundless possibilities of a flying camera carried by a drone to creative people on film sets. However, there are two very big challenges towards this ambitious goal.

1. The cognitive load to simultaneously control a drone and produce artistic shot compositions is incredibly high.
2. A film set has an unpredictable and unstructured nature where actors are walking around and scenes may change very quickly.

The automated trajectory generation methods presented by Roberts and Hanrahan [2016] as well as by Gebhardt et al. [2016, 2018] address the first point by providing a complete automated flight tool to automatically film static scenes by defining key frames.

However, a cinematographer wants to keep control over his work by having the ability to react on unpredicted movements and frame actors in real-time. Therefore, we believe an end-to end motion planning from global user inputs to low-level drone control is not the right way to address both aforementioned problems. We believe a separation into a local feasible trajectory controller and a global planner will be the right way to address the challenging problem and bring technology one step closer towards our vision.

In this thesis, we provide a possible solution of a modular low-level motion planning strategy. In order to prove our proposed method, we show the modularity in a wide range of challenging and compelling scenarios where we go far beyond the state of the art of real-time drone cinematography and call this concept *intelligent drone cinematography*.

1.3. Intelligent Cinematography in the Literature

Intelligent drone cinematography can be understood as a special case of a sensor placement problem, where the *sensors* are the cameras and the objective are the filming requirements of the user. In the following, we present a historical background of the building blocks needed for intelligent drone cinematography. First, we discuss active robot vision approaches. They can be interpreted as the first camera control problem. This is followed by presenting the literature of virtual camera placement in virtual environments such as in computer games. These two paragraphs serve as background for the drone cinematography as well as drone based human motion capturing. A detailed related work of the individual and detailed topics, such as GPS less navigation, will follow in the individual subsequent chapters.

Active robot vision: Sensor placement has been an active area of research in recent years in a number of areas as for example in ro-

botics and computer vision. For instance, the general problem of task planning in robotics and its component areas of motion-, grasp-, and assembly-planning can be seen as different facets of a sensor placement problem. In active sensing, sensor parameters are controlled in response to the requirements of the task as shown in Bajcsy [1988] and Swain and Stricker [1993]. It has been shown in Aloimonos et al. [1988a] that active sensing can take ill-posed vision problems and render them well posed through constraints introduced by the dynamic nature of the sensor.

With the development of more powerful computers, people started to explore the possibility of using camera-controlled robots for automatic product assemble. In Sakane et al. [1987], a system is presented where the goal was to actively minimize occlusions for e.g. automated manipulation task. Therefore, the authors present a system for hand-eye occlusion minimization system. Tarabanis et al. [1991] present a method to determine viewpoints for a robotic vision system for which object features of interest will simultaneously be visible, inside the field-of-view, in-focus and magnified as required.

Virtual camera placement: Automatic camera control was studied extensively in the context of virtual environments in computer graphics. From an application point of view, approaches to control a camera can be distinguished based on whether the user has some degree of interactive control, or the application assumes full control of the camera itself. According to Christie et al. [2008a], there are a number of fundamental issues in interactive camera control. Firstly, direct interactive control of a virtual camera requires an expert user. This because there is a huge cognitive load to deal simultaneously with all six degrees of freedom of a camera. Cameras in computer graphics are modeled using three degrees of freedom for Cartesian coordinates and three rotational degrees.

This problem was tried to solve by not directly control the cameras degrees of freedom but to describe the resulting video, as for example shown in [Gleicher and Witkin, 1992]. They proposed the through-

the-lens control concept of a camera. There, the user can define *where* and *how* the actors should appear on the screen instead of directly manipulating the camera's degrees of freedom.

A second issue is the partially automated camera planning. Motion control of a virtual camera can be considered as a special case of path planning and thus is a hard problem with a complexity that is exponential in the number of degrees of freedom, as shown by Christie et al. [2008b]. Furthermore, the mathematical relation between an object in a 3D scene and its projection on the 2D screen is non-linear.

For camera control techniques, interactive computer games serve as a benchmark. A classical camera control problem involves following one or more characters whilst simultaneously avoiding occlusions in a cluttered environment.

To control a camera automatically, several problems have to be solved: Visibility of the characters, collision avoidance with the camera as well as the environment, smooth motion, keeping characters in the view and aesthetic pleasant filming. Insights of how cameras are placed in film-sets are found in the cinematographic literature such as [Arijon, 1976] and [Katz, 1991] where for example the triangle principle or shot compositions are introduced.

Arijon's triangle principle invokes the notation of a line of action, which for single actors is determined by the direction of the actor's view, and for two actors the line between their heads. This ensures, especially in narrative scenes or sequences, that by ensuring the triangle principle, it can be assured that viewers would not be confused by changes in the relative positions, or by camera cuts from one shot to another. In order to produce aesthetically pleasant camera combinations, especially at a practical level, Mascelli [1965] observes in a number of compositional heuristics as for example, that *neither strong vertical nor horizontal line should be centered.* This is known in photography and cinematography as the *rule of thirds.* The guideline proposes that an image should be imagined as divided into nine equal parts by two equally spaced horizontal lines and two equally spaced vertical lines, and that important compositional elements should be placed along these lines or their intersections

Recent work in gaming was presented by Galvane et al. [2013] where they present in an automated computation of appropriate viewpoints in complex 3D scenes. In [Galvane et al., 2015] the same authors present a method for generating virtual camera rails and computing smooth camera motions on these rails.

Galvane et al. [2017] present a similar system presenting an active tool for shot compositions in dynamic environment. The only work using multiple drones for subject filming is presented by Poiesi and Cavallaro [2015]. There the authors try to keep the subject in the field of view. They show the approach working in simulation.

However, virtual environments are not limited by real-world physics and robot constraints and hence can produce arbitrary camera trajectories, velocities and viewpoints. Also all positions of all participants, cameras and objects are known a-priori, which makes it easier for planning

Drone cinematography: There is currently very little literature on autonomous drones applied to cinematography [Mademlis et al., 2018]. For example in [Srikanth et al., 2014] an interesting approach is presented to control quadrotors for lighting purposes. The authors present a solution, which can to achieve automatically a specific lighting effect on dynamic subjects using a quadrotor equipped with a fixed portable light source. Their solution processes the images from a static camera to compute the 3D motion commands for the quadrotor.

Closer to our work, Joubert et al. [2015] address the challenge of autonomously performing camera shots with quadrotors. They present an interactive tool that allows users to design physically plausible trajectories by visually specifying shots. They use a virtual environment to compose and preview the shots. Their tool however remains limited to outdoor environment. It also requires to manually describe the path beforehand and does not allow to track targets in real-time. Roberts and Hanrahan [2016] propose a method that re-times a user-defined physically infeasible Drone Trajectory. With infeasible, the authors define that the trajectory cannot respect the dynamics and

physical limits of the quadrotor. A similar method was presented by Gebhardt et al. [2016, 2018] where feasible trajectories for drones were computed according to user preferences in static scenes. Although this methods are interesting, they only deal with aesthetic camera paths and cannot react in real-time on moving objects with a unknown motion pattern. This would require recomputing the whole path, which is computationally not trackable. All this approaches worked with a global optimization strategy which only worked in interactive rates (e.g. around 1-2 seconds on normal desktop computers).

A step forward to real-time drone cinematography is presented in [Joubert et al., 2016]. The author presents a system working with moving subjects. It is able to capture drone video footage of human subjects performing limited movements. The authors track the subjects using an RTK GPS and IMU sensors. Joubert et al. [2016] uses a simplification of the sampling based search space, called torus space. This was introduced in by Lino and Christie [2015]. However, although the approach worked in real-time, they could only frame two subjects on the screen. It is also not possible due easily integrate a collision avoidance. Galvane et al. [2018] combine a torus-space based approach with a high-level planner. The system takes handcrafted trajectories as inputs and computes a C4 continuous spline. The system is still only able to frame two person at the time. Ceiling and floor constrains are taken into account. However, local collision avoidance is not included in the system

Drone based motion capture: Using drones as a mobile camera platform for real-time motion capturing is a new research topic and therefore not much work is published. In [Zhou et al., 2018], the authors used a single quadrotor equipped with a RGB camera following a human, similar to a system proposed by [Lim and Sinha, 2015]. The pose of the human is computed in an offline process using a multi-frame skeleton detection. A general problem with such approaches is first the problem of occlusion and ambiguities using a monocular camera, and second the system can nor run online. The system also only can

track humans. A more general approach was proposed by [Xu et al., 2017]. The authors used multiple quadrotors following a human using visual detection. Further, the drones are equipped with a depth camera to capture and record the motion sequences. The presented pipeline computed a deformable mesh in an offline process. Although this system could minimize the occlusion and ambiguity, it is still not able to process data in real-time and react on the human.

1.4. Contribution Statement

In this thesis, we made contributions in the field of GPS less navigation, drone cinematography and real-time human motion capturing.

Our first core contribution is the development of conceptual building blocks for modular low-level motion planning which we state to be the basis for *intelligent drone cinematography*.

- We provide a very flexible model predictive control formulation that allows to abstract a drone as a flying camera. The method allows an intuitive and easy way to include cinematographic concepts by defining key frames. The developed control strategy steers the drone in a way that the desired key frame – the frame the user wants to see the resulting camera footage – is fulfilled. Cinematographic concepts such as shot size, subject framing and relative view angles can easily be integrated. In addition, we also show how to integrate subject visibility and subject collision avoidance. The work is presented in chapter 5.

- We developed a real-time method for drones that allows to track arbitrary high-level trajectories with guaranteed constraints. The input trajectories do *not* need to be physically feasible in the sense Roberts and Hanrahan [2016] defined. Further, we show extensions of the proposed algorithm by adding the possibility to do multi-camera productions using drones. The algorithms have a real-time inter drone collision avoidance, and can film

with a reciprocal visibility minimization. We present the work chapter 6.

We used the above presented building blocks as a basis for our second core contribution:

- We developed the first real-time human motion capturing system using drones. We provided a completely self-contained method that allows human skeleton tracking in real-time and to control of the states of multiple drones in real-time. We present the work in chapter 7.

In order to develop the above presented core building blocks for intelligent drone cinematography, we did a third core contribution in the field of environment independent position estimation:

- We provided a very lightweight and fast Visual Inertial Odometry (VIO) algorithm to accurately estimate the position of a drone, presented in chapter 3. In addition, we present an extension to the presented VIO algorithm that directly includes pixel intensity for position estimation, presented in chapter 4.

1.5. Publications

In the context of this thesis, the technical contributions have lead to top-tier peer reviewed conference publications:

- **Nägeli, T.**, Oberholzer, S., Plüss S., A., Alonso-Mora, J., Hilliges, O. (2017). Flycon: Real-time Environment-independent Multi-view Human Pose Estimation with Aerial Vehicles (TOG), 36(4), 132.

- **Nägeli, T.**, Meier, L., Domahidi, A., Alonso-Mora, J., Hilliges, O. (2017). Real-time planning for automated multi-view drone cinematography. ACM Transactions on Graphics (TOG), 36(4), 132.

- **Nägeli, T.**, Alonso-Mora, J., Domahidi, A., Rus, D., Hilliges, O. (2017). Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization. *IEEE Robotics and Automation Letters, 2(3), 1696-1703.*

- de Palézieux, Nicolas, **Nägeli, T.**, and Hilliges, O. "Duo-vio: Fast, light-weight, stereo inertial odometry." *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on. IEEE, 2016.*

- Tanskanen, P., **Nägeli, T.**, Pollefeys, M., Hilliges, O. (2015, September). Semi-direct EKF-based monocular visual-inertial odometry. *In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on (pp. 6073-6078). IEEE.*

Althought not directly related to the work presented in this thesis, the following peer-reviewed publications were published during my PhD studies:

- Stevsic, S., **Nägeli, T.**, Alonso-Mora, J., Hilliges, O. (2018). Sample Efficient Learning of Path Following and Obstacle Avoidance Behavior for Quadrotors. *IEEE Robotics and Automation Letters.*

- Alonso-Mora, J., Montijano, E., **Nägeli, T.**, Hilliges, O., Schwager, M., Rus, D. (2018). Distributed multi-robot formation control in dynamic environments. *Autonomous Robots, 1-22.*

- Araki, B., Strang, J., Pohorecky, S., Qiu, C., **Nägeli, T.**, Rus, D. (2017, May). Multi-robot path planning for a swarm of robots that can both fly and drive. *In Robotics and Automation (ICRA), 2017 IEEE International Conference on (pp. 5575-5582). IEEE.*

- Gebhardt, C., Hepp, B., **Nägeli, T.**, Stevšić, S., Hilliges, O. (2016, May). Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. *In Proceedings*

of the 2016 CHI Conference on Human Factors in Computing Systems (pp. 2508-2519). ACM.

- Hepp, B., **Nägeli, T.**, Hilliges, O. (2016, October). Omni-directional person tracking on a flying robot using occlusion-robust ultra-wideband signals. *In Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on (pp. 189-194). IEEE.*

- Alonso-Mora, J., **Nägeli, T.**, Siegwart, R., Beardsley, P. (2015). Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots, 39(1), 101-121.*

1.6. Patents

In the context of this thesis, the technical contributions have lead two patent applications:

- **Nägeli, T.** et al. Flycon: "Method and computer program for detecting the form of a deformable object", 2018, EU Patent (pending).

- **Nägeli, T.** et al. VirtualRails: "A Drone and Method of Controlling Flight of a Drone", 2017, EU Patent (pending).

Chapter 2.

Preliminaries and Mathematical Foundation

In the following, we provide an overview of the used mathematical notation as well as the basic Lie algebra math, quadrotor modeling, state estimation concepts. We also give an introduction into model predictive control.

2.1. Notation

We provide a consistent and easy to follow notation which means we avoid unnecessary sub - or superscripts. Whenever possible, we follow the standard notation proposed in literature.

2.1.1. Points and Vectors

In the following, we denote points in 3D as $\mathbf{p}_{(\cdot)}$ with a name as subscript, e.g. $\mathbf{p}_{\mathrm{q}}$ for $\mathbf{p}_{\mathrm{quadrotor}}$. If the context is clear and additional subscripts make the variable not readable anymore, we shorten the subscript to e.g. $\mathbf{p}_q$ for $\mathbf{p}_{\mathrm{quadrotor}}$. A relative vector between two

points $\mathbf{p}_a$ and $\mathbf{p}_b$ is denoted as $\mathbf{r}_{ab}$. A superscript $\mathbf{r}_{ab}^W$ indicates the vector $\mathbf{r}_{ab}$ is expressed in frame W. Without subscript vectors are expressed in the standard Earth North Up (ENU) inertial global world coordinate system frame W e.g. $\mathbf{r}_{ab}^W := \mathbf{r}_{ab}$

The following coordinate frames are used throughout this thesis and are illustrated in Fig. 3.2: W – the inertial world frame; O – the origin frame; A_j – anchor frames; C – the camera frame; I – the IMU or Quadrotor body frame.

2.1.2. Rotation Matrices and Quaternions

Rotation matrices performing rotations from frame A to frame B are denoted by $\mathbf{R}_{AB} = \mathbf{R}(\bar{\mathbf{q}}_{AB}) \in SO(3)$, where $\bar{\mathbf{q}}_{AB}$ is the corresponding quaternion. Similar to position vectors, we shorten $\mathbf{R}_{AB}$ to $\mathbf{R}_B$ if A is the inertial global world coordinate system frame W. We adhere to the JPL quaternion definition [Breckenridge, 1979] and denote a quaternion by $\bar{\mathbf{q}} = [q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k} + q_w] = [\mathbf{q}, q_w]^T$. Quaternion multiplication is denoted by $\otimes$ and is defined as:

$$\bar{\mathbf{q}}_{q\otimes}\bar{\mathbf{q}}_{p=}\begin{bmatrix} p_w & -p_z & p_y & p_x \\ p_z & p_w & -p_x & p_y \\ -p_y & p_x & p_w & p_z \\ -p_x & -p_y & -p_z & p_w \end{bmatrix}\begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix}.$$

A rotation around a unit length axis $\omega = [\omega_x,\ \omega_y,\ \omega_z]$ by an angle α is expressed as a quaternion as follows:

$$\bar{\mathbf{q}}_{q(}\omega, \alpha) = \begin{bmatrix} \sin(\alpha/2)\omega_x \\ \sin(\alpha/2)\omega_y \\ \sin(\alpha/2)\omega_z \\ \cos(\alpha/2) \end{bmatrix} \tag{2.1}$$

The rotation defined by a general 3 dimensional vector $\mathbf{v}$, where $\|\mathbf{v}\| \neq \mathbf{0}$, is written as a quaternion by decomposing $\mathbf{v}$ into $\alpha = \|\mathbf{v}\|$ and $\omega = \frac{\mathbf{v}}{\alpha}$ and applying (2.1). This operation is denoted by:

$$\bar{\mathbf{q}}(\mathbf{v}) = \bar{\mathbf{q}}\left(\frac{\mathbf{v}}{\|\mathbf{v}\|}, \|\mathbf{v}\|\right) \tag{2.2}$$

To compute a rotation matrix from a quaternion, we use the the function $\mathbf{R}(\bar{\mathbf{q}}_{BA})$ defined as:

$$\mathbf{R}(\bar{\mathbf{q}}_{BA}) = \begin{bmatrix} q_x^2 - q_y^2 - q_z^2 + q_w^2 & 2(q_x q_y + q_z q_w) & 2(q_x q_z - q_y q_w) \\ 2(q_x q_y - q_z q_w) & -q_x^2 + q_y^2 - q_z^2 + q_w^2 & 2(q_y q_z + q_x q_w) \\ 2(q_x q_z + q_y q_w) & 2(q_y q_z - q_x q_w) & -q_x^2 - q_y^2 + q_z^2 + q_w^2 \end{bmatrix}.$$

The concatenation of rotations are composed by multiplication on the left. Thus, the rotation matrix that rotates vectors from a frame A to a frame C can be composed by:

$$\mathbf{R}_{CA} = \mathbf{R}_{CB}\mathbf{R}_{BA}$$

Analogously, the quaternion $\bar{\mathbf{q}}_{CA}$ corresponding to $\mathbf{R}_{CA}$ can be composed by:

$$\bar{\mathbf{q}}_{CA} = \bar{\mathbf{q}}_{CB} \otimes \bar{\mathbf{q}}_{BA}$$

Note that, with the JPL quaternion convention, the order of multiplication is the same for rotation matrices as for quaternions. Orientation errors are described in $so(3)$, the tangent space of $SO(3)$, and are written as δ_θ.

2.1.3. Lie Derivatives

In this thesis, we use lie derivatives of a rotation in $SO(3)$ with respect to its tangent space $so(3)$. We are using the derivation provided in

[Mourikis et al., 2009] and [Mourikis and Roumeliotis, 2007]. A detailed derivation can be found in [de Palézieux et al., 2016]. Consider the rotation matrix $\mathbf{R} \in SO(3)$ and the vector $\mathbf{x} \in \mathbb{R}^3$. Let

$$\mathbf{y} = \mathbf{R}\mathbf{x} \qquad \mathbf{y}' = \mathbf{R}^T\mathbf{x}$$

The differentiation by $\mathbf{x}$ is straightforward:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{R} \qquad \frac{\partial \mathbf{y}'}{\partial \mathbf{x}} = \mathbf{R}^T$$

The differentiation with respect to the rotation parameters that define $\mathbf{R}$ (according to Eq. (2.2)), which we simply denote by $\frac{\partial}{\partial \mathbf{R}}$, is:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{R}} = \lfloor \mathbf{y} \rfloor_\times \qquad \frac{\partial \mathbf{y}'}{\partial \mathbf{R}} = -\mathbf{R}^T \lfloor \mathbf{x} \rfloor_\times,$$

where $\lfloor \mathbf{x} \rfloor_\times$ is the skew symmetric matrix of a three dimensional vector and defined as

$$\lfloor \mathbf{w} \rfloor_\times = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}.$$

2.1.4. States, Estimates and Set-points

In this thesis, states are denoted as $\mathbf{x}_{(\cdot)}$. We define them as variables which evolve over time according to a mathematical model and also might be influenced by other variables or inputs. Expected or estimated values of a variable $\mathbf{x}_{(\cdot)}$ are denoted by $\mathbb{E}\left[\mathbf{x}_{(\cdot)}\right] = \hat{\mathbf{x}}_{(\cdot)}$. Desired setpoints are indicated with a subscript $(\cdot)_d$.

We distinguish between an a-priori estimate and an a-posteriori estimate. The prior state is the *a priori* estimate of a state $\mathbf{x}$ and denoted as $\hat{\mathbf{x}}_k$. If the context is clear, we also write $\mathbf{x}_{\text{prior}}$ for the a-priori state. The a-posteri estimate of a state is denoted as $\hat{\mathbf{x}}_k^+$.

We denote the set of positive definite matrices of size n by $\mathbb{S}^n_{++}$ and the set of positive semi-definite matrices of size n by $\mathbb{S}^n_{+}$. Given a vector $\mathbf{x}_{(\cdot)} \in \mathbb{R}^n$, we define the square of the norm as $\|\mathbf{x}_{(\cdot)}\|_P \triangleq \mathbf{x}^T_{(\cdot)} P \mathbf{x}_{(\cdot)}$ for $P \in \mathbb{S}^n_{++}$.

2.1.5. Measurements and Residuals

A general measurement is denoted by $\mathbf{z}_{(\cdot)}$ and defined with the measurement-function $\mathbf{z}_{(\cdot)} = \mathbf{h}_{\mathbf{x}_{(\cdot)}}$. For example if only the length of a vector $\mathbf{r}_{ab}$ can be observed the measurement function is given as $\mathbf{z}_l = \mathbf{h}_{\mathbf{x}_l} = \|\mathbf{r}_{ab}\|$. An estimated measurement is defined as $\hat{\mathbf{z}}_{(\cdot)} = \mathbf{h}_{\hat{\mathbf{x}}_{(\cdot)}}$. A residual is written as $\rho_{\cdot}$ and defined as the difference between a measurement and an estimated measurement $\rho_{\cdot} := \mathbf{z}_{(\cdot)} - \mathbf{h}_{\hat{\mathbf{x}}_{(\cdot)}}$.

2.2. Quadrotor Model

In the following we derive and introduce the mathematical model of a general quadrotor, presented by Alonso-Mora et al. [2015], followed by a specification needed for a Parrot Bebop 2, which is used in this thesis. A quadrotor is a multi-copter with four rotor blades where the propellers are mounted in one plane. Although latest research showed that rotor-crafts with less than four propellers are still controllable [Zhang et al., 2016] the quadrotor is the most used multi-rotor in research and therefore also our choice. A general quadrotor has four rotors which are mounted in fixed positions with respect to the body frame. Their motor speeds $\omega_{\mathbf{m}}$ can be controlled individually as indicated in Fig. 2.1. Each motor of the quadrotor produces a force $\mathbf{F}_{\mathbf{q}_i} = a_t \omega^2_{m_i} \in \mathbb{R}^1$ and a moment $M_{q_i} = a_\omega \omega^2_i \in \mathbb{R}^1$ where a_t and a_ω are model specific constants. The roll Φ_q, pitch Θ_q and yaw Ψ_q angle as well as the total thrust $T \in \mathbb{R}^1$ are controlled by differential control of the individual rotor speeds ω_{m_i}. Due to the physical configuration of the actuators, a quadrotor is an unteractuated system. In order to control the translational dynamics, we have to indirectly control them by first rotating around the roll Φ_q and pitch Θ_q angle in order

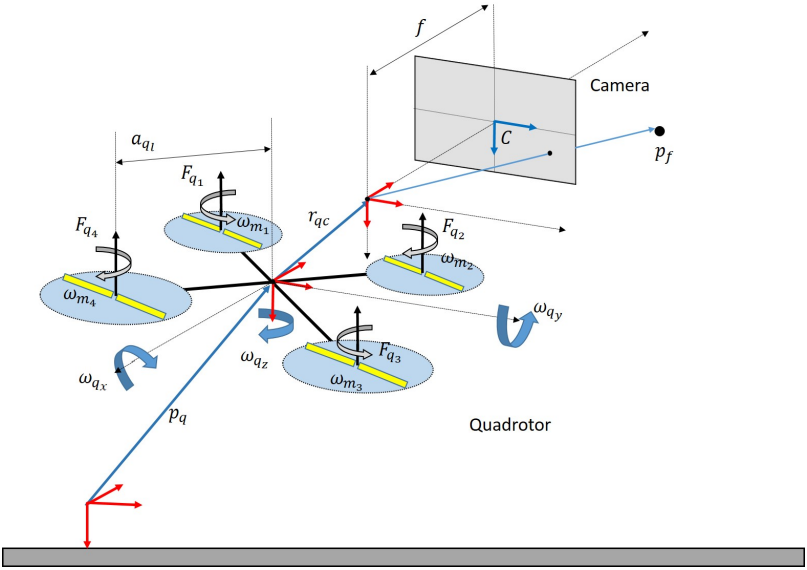


Figure 2.1.: *Illustration of a quadrotor model.*

to perform a translation. Following Mellinger and Kumar [2011], the total force along the body z axis T and the angular moments around the body axes $\mathbf{M_q} = [M_{q_x}, M_{q_y}, M_{q_z}]^T \in \mathbb{R}^3$ can be written in matrix form and are given by:

$$\begin{bmatrix} T \\ \mathbf{M_q} \end{bmatrix} = \begin{bmatrix} a_t & a_t & a_t & a_t \\ 0 & a_t a_l & 0 & -a_t a_l \\ -a_t a_l & 0 & a_t a_l & 0 \\ a_\omega & -a_\omega & a_\omega & -a_\omega \end{bmatrix} \begin{bmatrix} \omega_{m_1}^2 \\ \omega_{m_2}^2 \\ \omega_{m_3}^2 \\ \omega_{m_4}^2 \end{bmatrix},$$

where a_l is the length from the center of gravity of the quadrotor to the rotor axis of the motor. Following [Mellinger and Kumar, 2011],

the angular acceleration around the center of gravity is given by

$$\dot{\omega_q} = {a_q}_J^{-1} \left[-\omega_{\mathbf{q}} \times {a_q}_J \omega_{\mathbf{q}} + \mathbf{M_q} \right]$$

with ${a_q}_J$ the moment of inertia matrix, which is assumed to be diagonal, of the center of mass along the body axes of the quadrotor and $\omega_{\mathbf{q}} = [\omega_{q_x}, \omega_{q_y}, \omega_{q_z}]$ the angular speeds around the body axes of the quadrotor. If $\bar{\mathbf{q}}_q$ resp. $\mathbf{R}_q(\bar{\mathbf{q}}_q)$ denotes the rotation from the body B to the inertial world frame W, the rotational velocity dynamic of the quadrotor's body frame with respect to the inertial frame is given by

$$\dot{\bar{\mathbf{q}}}_q = \frac{1}{2} \Omega(\omega_{\mathbf{q}}) \bar{\mathbf{q}}_q. \quad \text{or} \quad \dot{\mathbf{R}}_q = \lfloor \omega_{\mathbf{q}} \rfloor_\times \mathbf{R}_q.$$

For the position dynamic, we use a simplified point-mass model. This can be done due to the symmetric distribution and mass concentration around the center of the quadrotor. Therefore, the accelerations of the quadrotor's mass-point in the inertial frame $\ddot{\mathbf{p}}_q$ can be written as

$$\ddot{\mathbf{p}}_q = \mathbf{R}_q(\bar{\mathbf{q}}_q) \begin{bmatrix} 0 \\ 0 \\ \frac{T}{{a_q}_m} \end{bmatrix} - {a_q}_D \begin{bmatrix} \dot{\mathbf{p}}_{q_x} \\ \dot{\mathbf{p}}_{q_y} \\ \dot{\mathbf{p}}_{q_z} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{2.3}$$

where two forces are applied:

1. **Gravity:** $F_G = m{a_q}_m g$ in the z direction e_{Wz} of the inertial frame.
2. **Thrust:** T in the negative z direction e_{Bz} of the quadrotor body-frame.

2.2.1. Parrot Bebop 2 Model

For our experiments we used a Parrot Bebop 2 drone. Therefore, we slightly have to modify the general quadrotor dynamics in order to

Figure 2.2.: *The Parrot Bebop 2 Drone type which is used in this thesis.*

develop a controller which is able to use the interfaces provided by parrot's SDK[1]. The physical inputs which are provided by the SDK are four inputs to control the drone and the attached camera gimbal:

- **Physical drone inputs:** ϕ_q the desired roll angle, θ_q the desired pitch angle, ω_{q_ψ} the desired yaw speed and $\dot{\mathbf{p}}_{q_z}$ the desired z velocity.
- **Physical gimbal inputs:** θ_g the desired absolute camera pitch angle and ψ_g the desired relative camera yaw angle with respect to the quadrotor yaw angle.

the input vector is therefore defined as

$$\mathbf{u}_{\text{drone}} = [\mathbf{u}_{\text{q}}, \mathbf{u}_{\text{gimb}}] = [\dot{\mathbf{p}}_{q_z}, \omega_{q_\phi}\, \omega_{q_\theta}, \omega_{q_\psi} | \omega_{g_\theta}, \omega_{g_\psi}] \in \mathbb{R}^6.$$

Drone model: We are following the modeling presented in [Bristeau et al., 2011]. Because the z dynamic of the Bebop 2 is controlled

[1] http://developer.parrot.com/

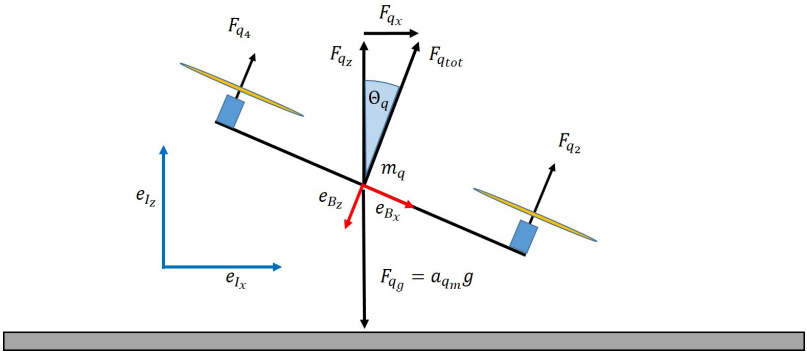


Figure 2.3.: *A schematic view to illustrate the Parrot Bebop 2 model dynamics in the x axis. The y axis can be derived similarly.*

separately from the translational x and y dynamics of the drone we can separate these. We can assume the Bebop controller keeps the actual height constant during a translation of the drone. The thrust T which is needed for a drone to hover, e.g. $\phi_q \approx 0$ and $\theta_q \approx 0$, is given as $T = a_{q_m} g$ with a_{q_m} the mass of the quadrotor and g the earth's gravity. If the quadrotor performs a translation, the norm of the thrust vector, which is always pointing along the body z e_{B_z} axis, has to be increased in order to keep the height. This fact is illustrated in Fig. 2.3 and we can compute the force acting in the earth's x direction as:

$$F_{q_x} = \tan(\Theta_q) F_{q_g}$$

with F_{q_x} the force in the x axis of the intertial frame and $F_{q_g} = a_{q_m} g$, acting in the negative z direction of the inertial frame (note the definition of the angle cancels out the minus sign of F_{q_g}).

According to Newton's law, we can simply divide both sides by the quadrotor's mass and get the position dynamics. The horizontal

position dynamics of the drone are then given as

$$\ddot{\mathbf{p}}_{q_{x,y}} = \mathbf{R}_{\Psi_q(\Psi_q)} \begin{bmatrix} \tan(\Theta_q) \\ -\tan(\Phi_q) \end{bmatrix} g - \underbrace{\begin{bmatrix} \dot{\mathbf{p}}_{q_x} a_{q\,Dx} \\ \dot{\mathbf{p}}_{q_y} a_{q\,Dy} \end{bmatrix}}_{\text{Drag}} \in \mathbb{R}^2$$

where the matrix $\mathbf{R}_{\Psi_q(\Psi_q)} \in \mathbb{SO}(2)$ rotates the acceleration into the inertial frame. The air drag of a quadrotor can be approximated as a linear and quadratic term, dependent on the quadrotor's velocity, and acts against the moving direction of the quadrotor. According to Bristeau et al. [2009, 2011] the linear drag component is dominant at lower speeds and therefore we only add the linear approximation of the drag force. The z position dynamics is simply given as

$$\ddot{\mathbf{p}}_{q_z} = a_{q\,Dz} \dot{\mathbf{p}}_{q_z} \in \mathbb{R}^1.$$

As described in the general quadrotor model Sec.2.2, the quadrotor has to adjust the rotor speeds in order to track a reference roll or pitch angle. To approximate the internal controller dynamics, we add an artificial state, representing the rotation speed around the body x and y axis. The dynamics of the Euler angles ϕ_q, θ_q and Ψ_q are then given as:

$$\begin{bmatrix} \dot{\Phi}_q \\ \dot{\Theta}_q \\ \dot{\Psi}_q \end{bmatrix} = \begin{bmatrix} \omega_{q_\phi} \\ \omega_{q_\theta} \\ \omega_{q_\psi} \end{bmatrix}$$

and the additional rotational speed dynamics for the roll and pitch axis can be written as:

$$\begin{bmatrix} \dot{\omega}_{q_\phi} \\ \dot{\omega}_{q_\theta} \end{bmatrix} = \tau_{\omega_0}^2 \begin{bmatrix} \phi_{q_{\text{ref}}} - \Phi_q \\ \theta_{q_{\text{ref}}} - \Theta_q \end{bmatrix} - \underbrace{\tau_\eta \tau_{\omega_0} \begin{bmatrix} \omega_{q_\phi} \\ \omega_{q_\theta} \end{bmatrix}}_{\text{Drag}}$$

with model specific constants τ_{ω_0} and τ_η.

Camera gimbal: The camera of the bebop is attached to the robot via a software pan-tilt gimbal. The roll and pitch axes of the gimbal are stabilized, which mean the roll and pitch angles of the quadrotor during flying are compensated in order to have a stable and still image. The input which the SDK provides are the absolute pitch and Yaw angle of the gimbal, denoted as θ_g and ψ_g.

To get a smooth camera response, the underlying camera stabilization algorithm inside the parrot Bebop 2 drone is a 3rd. order model which we approximate. It allows us also easily to control the gimbal rotation speed.

$$\dot{\theta}_g = \omega_{g_\theta} \quad \text{and} \quad \dot{\psi}_g = \omega_{g_\psi}$$

While the camera is not directly attached to the center of the quadrotor, experimentally we have found it to be sufficient assuming the camera position and the quadrotor's position $\mathbf{p}_q$ to be identical.

The Full camera orientation matrix is given as the multiplication of the three rotations:

$$\mathbf{R}_{c(\bar{\mathbf{q}}_c)} = \mathbf{R}_{\theta_g(\bar{\mathbf{q}}_{\theta_g})}\mathbf{R}_{\psi_g(\bar{\mathbf{q}}_{\psi_g})}\mathbf{R}_{\Psi_q(\bar{\mathbf{q}}_{\Psi_q})}, \tag{2.4}$$

and in quaternion notation:

$$\bar{\mathbf{q}}_c = \bar{\mathbf{q}}_g(\theta_g) \otimes \bar{\mathbf{q}}_g(\psi_g) \otimes \bar{\mathbf{q}}_q(\Psi_q), \tag{2.5}$$

where $\bar{\mathbf{q}}_g(\theta_g)$ is the quaternion defined by the gimbal pitch angle ψ_g, $\bar{\mathbf{q}}_g(\psi_g)$ is the quaternion defined by the gimbal pitch angle ψ_g and $\bar{\mathbf{q}}_g(\theta_g)$ is the quaternion defined by the quadrotor's yaw angle Ψ_q The full state the quadrotor is given by its position $\mathbf{p}_q \in \mathbb{R}^3$ its velocity $\dot{\mathbf{p}}_q \in \mathbb{R}^2$ its orientation , i.e. roll Φ_q, pitch Θ_q and yaw Ψ_q. as well as the body angular states ω_{q_ϕ} and ω_{q_θ}

$$\mathbf{x}_\mathrm{q} = [\mathbf{x}_\mathrm{q}, \mathbf{x}_\mathrm{gimb}] = [\mathbf{p}_q, \dot{\mathbf{p}}_q, \Phi_q, \Theta_q, \Psi_q | \theta_g, \psi_g] \in \mathbb{R}^{10}.$$

The sets $\mathcal{X}$ and $\mathcal{U}$ are given by the physical limits of the environment (e.g. $p_z > 0$) and by the internal constraints of the Parrot Bebop 2 (e.g. maximal vertical and horizontal velocities, maximal roll pitch angle). The limits are described in the documentation of the Parrot SDK [2].

2.2.2. Set of States

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 $\mathbf{x}_{\mathrm{q}_k} \in \mathcal{X}$ and $\mathbf{u}_{\mathrm{q}_k} \in \mathcal{U}$ for all k.

2.3. Camera Projection Model

In order to use a camera we need a mathematical model which describes how 3D information is mapped on a 2D image and vice versa. The simplest projection model is the pinhole camera model that describes the camera with a simple projection. But in practice, this model is normally not enough, since the actual camera lenses used in real cameras create a distortion of the simple projection.

The pinhole camera model projects a 3D point $\mathbf{p}_f$ in the camera frame onto the virtual image plane according to:

$$\begin{bmatrix} u^u \\ v^u \end{bmatrix} = \begin{bmatrix} f_y \frac{\mathbf{p}_f}{\mathbf{p}_f} + C_x \\ f_x \frac{\mathbf{p}_f}{\mathbf{p}_f} + C_y \end{bmatrix}, \tag{2.6}$$

[2] http://developer.parrot.com/

where f_y and f_x are the focal lengths in the horizontal and vertical image directions in pixel units, respectively. C_x and Cy are the camera's center of projection in units of pixels.

Due to lens distortion, the feature $\mathbf{p}_f$ does not actually appear at pixel coordinates (u^u, v^u), but at distorted coordinates (u^d, v^d). This distortion is modeled with the plumb bob model [Brown, 1971], where the tangential distortion is assumed to be negligible:

$$\begin{bmatrix} u^d \\ v^d \end{bmatrix} = \begin{bmatrix} f_y & 0 \\ 0 & f_x \end{bmatrix} \left(1 + \kappa_1 r_u^2 + \kappa_2 r_u^4 + \kappa_3 r_u^6\right) \begin{bmatrix} u_n^u \\ v_n^u \end{bmatrix} + \begin{bmatrix} C_u \\ C_v \end{bmatrix}, \quad (2.7)$$

where κ_1, κ_2, and κ_3 are distortion parameters that depend on the camera lens, and $r_u = \sqrt{(u_n^u)^2 + (v_n^u)^2}$. (u_n^u, v_n^u) are the normalized undistorted pixel coordinates

$$\begin{bmatrix} u_n^u \\ v_n^u \end{bmatrix} = \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix}^{-1} \left(\begin{bmatrix} u^u \\ v^u \end{bmatrix} - \begin{bmatrix} C_x \\ C_y \end{bmatrix} \right)$$

If we need to recover the undistorted pixel coordinates (u^u, v^u) from distorted measurements (u^d, v^d), we need to invert equation (2.7) . This requires the computation of r_u. While r_u is straightforwardly computed from (u^u, v^u), it cannot be analytically computed from (u^d, v^d). This can iteratively be done using the Newton-Raphson method [Abramowitz et al., 1965]:

$$r_d = r_u \left(1 + \kappa_1 r_u^2 + \kappa_2 r_u^4 + \kappa_3 r_u^6\right).$$

2.4. Inertial Measurement Unit (IMU)

For quadrotor controlling as well as for navigation in GPS denied areas an Inertial Measurement Unit (IMU) is needed. An IMU is a device that normally consists of 3 sensors to estimate the orientation of an object with respect to the inertial world frame.

1. **Gyroscope:** Measures the rotational speed $\omega_{\mathbf{q}}$ around the body angular axis of the object to which the sensor is attached.

2. **Accelerometer:** Measures the superposition of the earth's gravity field and the acceleration of the object, to which the sensor is attached.

3. **Magnetometer:** Measures the superposition of the earth's magnetic field and the magnetic fields caused by electrical installations. Especially iron disturbs the magnetic field.

By integrating the gyroscope values, it is possible to capture very fast attitude changes. However, real-world sensors are affected by unknown and drifting sensor offsets. By integrating the pure measurements, the estimated value will drift quickly which results in a very bad attitude estimation. This drift can be compensated using the measurement of the static gravity field and the earth magnetic field. Using an accelerometer can fix two degrees of freedom of the attitude estimation. However, especially on a quadrotor it is not trivial to measure the gravity vector due to the high sensor noise level and the acceleration of the quadrotor itself. In order to correct the third component of the gyro integrated attitude, a magnetometer can be used which measures the earth magnetic field. However, especially indoors this estimate can be highly affected by additional magnetic fields caused by electrical installations. To accurately estimate the orientation, there is a bunch of literature in how these three types of sensors can be fused in an effective way [Lefferts et al., 1982a] or [Madgwick, 2010].

2.5. State Estimation

In general, state estimation describes the method to gain an *optimal* estimate $\hat{\mathbf{x}}_{(\cdot)}$ of a set of *true* system states $\mathbf{x}_{(\cdot)}$. Often, the true state is hidden and only part of the state-space can be measured. This can be illustrated by a moving object, such as a car: If the position can be

measured, only this state is directly observable, while the velocity of the car is a so-called hidden state and has to be estimated through the system dynamics. Estimating states is done using two fundamental resources of information: The system model describing how states evolve over time $f(\mathbf{x}_{(\cdot)}, \mathbf{u}_{(\cdot)})$ and measurement model $h(\mathbf{x}_{(\cdot)})$:

- **A system model:** The system model $f(\mathbf{x}_{(\cdot)}, \mathbf{u}_{(\cdot)})$ describes how the system states are evolving over time.

- **Observations:** The measurement model $h(\mathbf{x}_{(\cdot)})$ describes how measurements are related to observations.

In general we can distinguish between three different concepts for state estimation depending on the time where the information is processed or generated as illustrated in Fig. 2.4: Smoothing in the past, Filtering at the current time-step and predicting in the future. These tree concepts are illustrated in Fig. 2.4.

Smoothing: Smoothing is the method to correct states in the past using new, incoming information.

- Fixed point: The smoothed state is only computed at a fixed point in time, or possible, several fixed points. This technology is used to improve the anchor estimates of our visual odometry presented in Sec.3.

- Fixed lag: The smoothed state is computed at a fixed lag in time from the current measurement. This method is used to be able to merge sensor data arriving with a time delay to the estimator. They can correct the current state estimate by updating the fixed lag state.

- Fixed interval: Smoothed state estimates are computed at every measurement time from the start to the end of a fixed interval.

Filtering: In state filtering, the goal is to continuously provide the best estimate of the system state $\mathbf{x}_{(\cdot)}$. When a new measurement becomes available, the filter processes the measurement and provides an improved estimate of the state at the new measurement time.

Prediction: If states in the future are of interest, the process model can be used to predict the future states. The prediction is based on the state $\mathbf{x}_0$ at time-step t_0 and known inputs $\mathbf{u}_0$. This concept is used in the Model Predictive Controlling to predict the future to optimally plan the control actions.

2.5.1. Kalman Filtering

The Kalman Filter [Gibbs, 2011] is an algorithm that produces an optimal estimate in real-time, assuming the measurements are affected by Gaussian noise and the system is time linear and invariant. In the case of a non-linear system, there is a bunch of suboptimal approaches based on the so-called *Extended Kalman Filter* (EKF). We quickly introduce the EKF and two extensions, the *iterated extended Kalman filter* (IEKF) for highly non linear measurement functions and the Error State Extended Kalman Filter (ISEKF) for Kalman filtering on manifolds. Both of these extensions are designed to improve the state estimate produced by an EKF.

Extended Kalman filtering

The extended Kalman filter (EKF) is the nonlinear version of the Kalman filter witch is linearized around an estimate of the current mean and covariance. Although there are no general guarantees about optimality of the EKF, it is the standard in the theory of real-time nonlinear estimation. The equations are given in algorithm 1.

Algorithm 1 Extended Kalman Filtering

Require: Previous a posteriori state estimate: $\hat{\mathbf{x}}_{k-1}^{+}, \mathbf{P}_{k-1}^{+}$

Prediction

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

$\mathbf{P}_k = \mathbf{F}_k \mathbf{P}_{k-1}^{+} \mathbf{F}_k^T + \mathbf{Q}$

Update

$\rho_k = \mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_k)$

$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_k \mathbf{H}_k^T + \mathbf{R}$

$\mathbf{K}_k = \mathbf{P}_k \mathbf{H}_k^T \mathbf{S}_k^{-1}$

$\hat{\mathbf{x}}_k^{+} = \hat{\mathbf{x}}_k + \mathbf{K}_k \rho_k$

$\mathbf{P}_k^{+} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\, \mathbf{P}_k$

with $\mathbf{F}_k = \frac{\partial f(\hat{\mathbf{x}}_{k-1}^{+}, \mathbf{u}_k)}{\partial \mathbf{x}_k}$ and $\mathbf{H}_k = \frac{\partial \mathbf{h}(\hat{\mathbf{x}}_k)}{\partial \mathbf{x}_k}$

Iterated Kalman filtering

In case a highly non-linear measurement function $h(\mathbf{x}_{(\cdot)})$ it is possible to improve the estimate using a modification of the standard EKF, kalled *Iterated Extended Kalman Filter* (IEKF) [Denham and Pines, 1966]. The IEKF can remove the effects of measurement model non-linearities. After processing the measurement in the update step, is used in the IEKF to recompute the measurement Jacobian $\mathbf{H}^j$, where the superscript j means the j'th iteration. The IEKF can be interpreted as a reweighed gauss newton method [Bell and Cathey, 1993]. The process is repeated until the change in the state estimate is small. It is important to notice that the measurement z_k is only used once to update the state at the end of all iterations. We provide in algorithm 4 the basic equations of the IEKF. Details can be found in [Gibbs, 2011].

Algorithm 2 Iterated Kalman Filtering

$\hat{\mathbf{x}}_k = f(\hat{\mathbf{x}}^+_{k-1}, \mathbf{u}_k)$
$\mathbf{P}_k = \mathbf{F}_k \mathbf{P}^+_{k-1} \mathbf{F}^T_k + \mathbf{Q}$

Update
$\eta^0 = \hat{\mathbf{x}}_k$
for $j = 0$ to `max_it` **do**
  $\rho^j_k = \mathbf{z}_k - \mathbf{h}(\eta^j)$
  $\mathbf{H}^j_k = \frac{\partial \mathbf{h}(\eta^j)}{\partial \mathbf{x}_k}$
  $\mathbf{S}^j_k = \mathbf{H}^j_k \mathbf{P}_k (\mathbf{H}^j_k)^T + \mathbf{R}$
  $\mathbf{K}^j_k = \mathbf{P}_k (\mathbf{H}^j_k)^T \mathbf{S}^{-1}_k$
  $\delta^{j+1}_\eta = \mathbf{K}_k \left(\rho^j_k - \mathbf{H}^j_k \left(\hat{\mathbf{x}}_k - \eta^j \right) \right)$
  $\eta^{j+1} = \hat{\mathbf{x}}_k + \delta^{j+1}_\eta$
  if $\|\delta^{j+1}_\eta\|$ small **then**
    Stop iteration
  end if
end for
$\hat{\mathbf{x}}^+_k = \eta^{j+1}$
$\mathbf{P}^+_k = \left(\mathbf{I} - \mathbf{K}^j_k \mathbf{H}^j_k \right) \mathbf{P}_k$

with $\mathbf{F}_k = \frac{\partial f(\hat{\mathbf{x}}^+_{k-1}, \mathbf{u}_k)}{\partial \mathbf{x}_k}$ and $\mathbf{H}_k = \frac{\partial \mathbf{h}(\hat{\mathbf{x}}_k)}{\partial \mathbf{x}_k}\Big|_{\hat{\mathbf{x}}_= \eta^j}$

Error state Kalman filtering

The classical EKF computes a state update according to the general linear additive update equation $\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k + \mathbf{K}_k \rho_k$. This update step works for euclidean vector spaces e.g. $\mathbf{x} \in \mathbb{R}^n$.

But as soon as the underlying system uses states which are not in an euclidean space any more, which is for example the case using rotations $\mathbf{R}_{ab}, \bar{\mathbf{q}}_{ab} \in \mathbb{SO}3$, it is not possible to use an additive update step anymore. An error State kalman filter (ESKF) [Roumeliotis et al., 1999] can help to circumvent this problem, by transferring the filtering problem from the manifold Ω into a linearized, euclidean tangent space τ for the update step, as illustrated in Fig. 2.5.

Roumeliotis et al. [1999] presented an intuitive example about quaternion estimation is described which was proposed and derived in [Lefferts et al., 1982a]. Instead of estimating the total state, as is the case for the direct Kalman filter, an error state is estimated. Therefore, it can be differentiated between the *total state* $\hat{\mathbf{x}}$, and the *error state* δ. Usually, the error between two quantities is defined as the arithmetic difference between the two, which is how we define the error in estimated values which are linear, such as positions $\delta := \mathbf{x} - \hat{\mathbf{x}}$. For orientation errors however, the arithmetic difference is not suitable. We define the error of an orientation using an error quaternion $\delta\bar{\mathbf{q}}_q$, a small rotation between the estimated and true orientation. This error is multiplicative and can be defined as $\delta\bar{\mathbf{q}}_A = \bar{\mathbf{q}}_A^{-1} \otimes \hat{\bar{\mathbf{q}}}_A$. Using $\delta\theta$ to represent orientations in the Kalman filter reduces their dimensionality to 3. This is both computationally advantageous and circumvents the issues with a 4×4 orientation covariance matrix. In the error state EKF, the error state δ is the quantity being estimated and the covariance matrix $\mathbf{P}$ describes the uncertainty of δ. The total state $\hat{\mathbf{x}}$ is always updated such that the expected value of the error state $\mathbb{E}[\delta] = \mathbf{0}$. In other words, the total state $\hat{\mathbf{x}}$ always represents the best estimate of $\hat{\mathbf{x}}$. The ESKF equations are given in algorithm 3.

Algorithm 3 Eror State Kalman Filtering

Require: Previous state estimate: $\hat{\mathbf{x}}_{k-1}^{+}, \mathbf{P}_{k-1}^{+}$

Prediction

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

$\mathbf{P}_k = \mathbf{F}_k \mathbf{P}_{k-1}^{+} \mathbf{F}_k^T + \mathbf{Q}$

Update

$\delta_\eta^0 = \mathbf{0}$

$\rho_k = \mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_k)$

$\mathbf{H}^j = \frac{\partial \mathbf{r}^j}{\partial \delta_x}\Big|_{\hat{\mathbf{x}}=\eta^j}$

$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_k \mathbf{H}_k^T + \mathbf{R}$

$\mathbf{K}_k = \mathbf{P}_k \mathbf{H}_k^T \mathbf{S}_k^{-1}$

$\mathbf{S}^j = \mathbf{H}^j \mathbf{P}_{k|k-1} (\mathbf{H}^j)^T + \mathbf{R}$

$\mathbf{K}^j = \mathbf{P}_{k|k-1} (\mathbf{H}^j)^T \left(\mathbf{S}^j\right)^{-1}$

$\delta_\eta^{j+1} = \mathbf{K}^j \rho_k$

$\hat{\mathbf{x}}_k^{+} = \hat{\mathbf{x}}_k \boxplus \delta_\eta^{j+1}$

$\mathbf{P}_k^{+} = \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k\right) \mathbf{P}_k$

with $\mathbf{F}_k = \frac{\partial f(\hat{\mathbf{x}}_{k-1}^{+}, \mathbf{u}_k)}{\partial \mathbf{x}_k}$ and $\mathbf{H}_k = \frac{\partial \mathbf{h}(\hat{\mathbf{x}}_k)}{\partial \delta_x}$

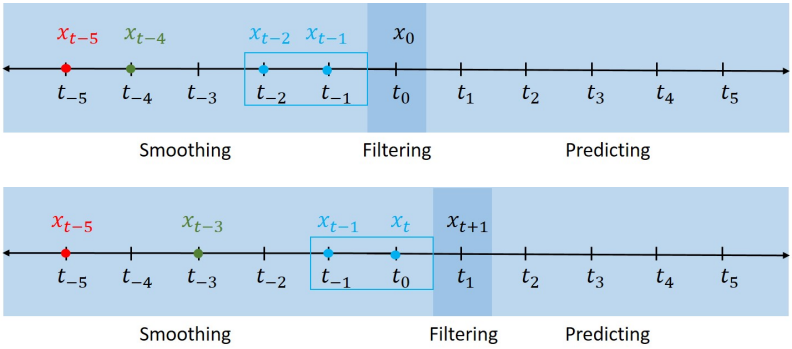


Figure 2.4.: *Top: Smoothing computes estimates in the past, Filtering computes an optimal estimate at the current time step k and Predicting uses the system model and inputs to predict future estimates. Smoothing can be distinguished into fixed point smoothing(red) where the estimate at a fixed time-step is computed using all information available, fixed lag (greed) where the estimate is produced at a fixed time lag and fixed interval (blue) which is a sliding window smoothing approach. Down: All three concepts at the next time step $k+1$. Note that the fixed point smoothing point (red) stays at the same position.*

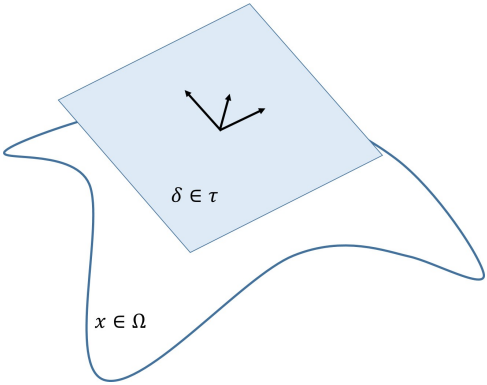


Figure 2.5.: *The manifold Ω with the embedded state x. The Tangent space τ is linearized around a specific state on the manifold. The error state $\delta$is defined in the tangent space.*

2.6. Model Predictive Control

Figure 2.6.: *While car driving, the driver has a intuition where the car is in one or two seconds (left). If something happens, the driver immediately recomputes the originally planned actions and computes a new plan (middle, right).*

Model Predictive Control (MPC) is a way to compute a control law by predicting the future using the full systen dynamics as well as state and input constraints. It was developed in the late 70's and first used in Chemical engineering processes [Qin and Badgwell, 2003]. With availability of fast computers and microprocessors, MPC is increasingly finding application in many other engineering domains such as automobiles, aerospace industries and robotics. The major strengths of MPC are abilities to handle multi-variable interactions and operating constraints as well as non linear systems. MPC is formulated as a constrained optimization problem, which is solved on-line repeatedly by carrying out model based forecasting over a moving window of time as illustrated in Fig. 2.7. In Fig. 2.6 an intuitive explanation of MPC is given. While driving a car, the driver has an intuition of where the car is in one or two seconds (left). If something happens, the driver immediately recomputes the originally planned actions and computes a new plan (middle, right). In algorithm 2.8 we present the most basic MPC formulation which we will adopt later in the following Chapters to the specific problems. The basic MPC formulation is given as the

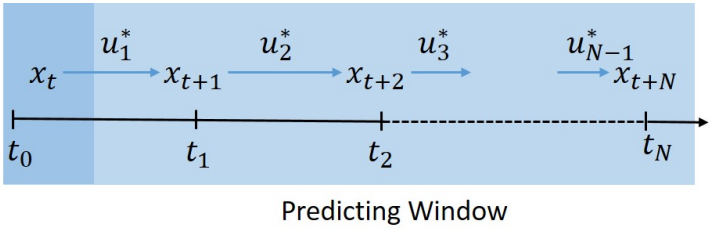


Figure 2.7.: *MPC uses the system dynamics to compute the optimal states* $\mathbf{x}$ *and inputs* $\mathbf{u}$ *according to their physical constraints in a prediction window.*

following optimization problem:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} \quad & \Sigma_{k=0}^{N-1} \mathbf{c}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{c}(\mathbf{x}_N, \mathbf{u}_N) \\
& \mathbf{x}_0 = \hat{\mathbf{x}}_0 && \text{(Initial State)} \\
\text{s.t.} \quad & \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k), && \text{(Dynamics)} \\
& \mathbf{x}_k \in \mathcal{X}, && \text{(State Constraints)} \\
& \mathbf{u}_k \in \mathcal{U}, && \text{(Input Constraints)}
\end{aligned}
\tag{2.8}
$$

where $\mathbf{c}(\mathbf{x}_k, \mathbf{u}_k)$ is the objective at a given timestep k, $\mathbf{c}(\mathbf{x}_N, \mathbf{u}_N)$ is the terminal set cost, $f(\mathbf{x}_k, \mathbf{u}_k)$ is the system model, $\mathcal{X}$ is the set of feasible States (e.g. the values of states which can be fulfilled by the system) and $\mathcal{U}$ are the feasible inputs of the system. A good introduction to MPC can be found in [Borrelli et al., 2017].

Part I.

Vision Based Position Estimation

Chapter 3.

Fast, Light-weight, Stereo Inertial Odometry

In order to fly accurately with a drone also in GPS denied areas, a robust and reliable position estimate is needed. Therefore, we developed a fast and accurate metric visual inertial odometry which relies on commercially available and affordable hardware both for sensing and computation. We use a low baseline stereo camera with an integrated Inertial Measurement Unit. Stereo measurements are used to initialize the depth of a tracked feature, while features are tracked over time using only a single camera. An indirect Extended Kalman Filter fuses IMU measurements and feature measurements extracted from the camera images to estimate the position, orientation, and velocity of the camera, in addition to IMU biases and a map of feature points. The algorithm runs in real time on an ARM based embedded micro-computer on-board a drone. In experiments, we demonstrate the performance of the system both indoors and outdoors, in hand held an in-flight scenarios. The achieved accuracy of the experiments is competitive with other research which uses custom designed hardware and desktop-grade processors.

3.1. Introduction

In this chapter we present a Visual-Inertial Odometry (VIO) algorithm that has been purposefully designed for the usage on small drones. A major draw of the presented system is that it is designed to run on affordable and off the self hardware. The algorithm runs at 100Hz on a low-power ARM CPU and works with forward-facing cameras, allowing for fast flight and removing the need for a second camera for collision avoidance. Furthermore, the algorithm provides accurate metric scale estimates without requiring specific initialization. We detail the algorithm here and release the code as open-source software.

3.2. Related Work

There is a vast body of literature on camera pose estimation, we concentrate our discussion on approaches of particular interest in the context of small, agile robots. One of the first flying robots leveraging vision was shown in [Altuğ et al., 2003, 2005], using two cameras to estimate the 6 degrees of freedom necessary for flight stabilization. Frauendorfer et al. [Fraundorfer et al., 2012] used a downward looking camera for optical flow based flight stabilization and an additional stereo camera pair for collision avoidance, mapping and short horizon path planning. Others have used the PTAM algorithm [Klein and Murray, 2007] directly to fly with a downward-looking monocular camera [Blösch et al., 2010] or leveraged modified versions [Engel et al., 2014, 2012] in conjunction with off-board processing.

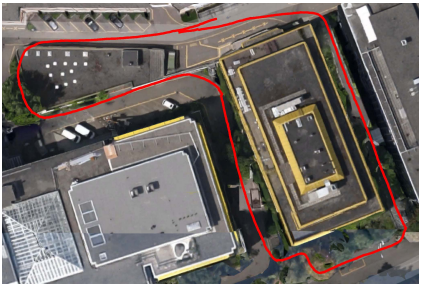
Key-frame based stereo approaches together with a loosely coupled IMU integration have been proposed to overcome the scale drift problem [Leutenegger et al., 2013]. Recently a number of approaches have been proposed that directly use dense surface measurements instead of extracted visual features, e.g. [Kerl et al., 2013]. Similarly, Forster et al. propose a sparse direct methods approach [Forster et al., 2014]. All of these methods use key-frames and hence require an explicit initialization phase to estimate scale from IMU data.

This initialization requirement can be circumvented using probabilistic approaches such as the Extended Kalman Filter (EKF), first applied to camera pose estimation in [Davison, 2003; Davison et al., 2007]. In aerospace engineering, the indirect or error-state Kalman Filter has been introduced by Lefferts [Lefferts et al., 1982b] and reintroduced by Roumeliotis et al. [Roumeliotis et al., 1999]. Similar approaches to visual odometry exist such as the Multi State Constrained Kalman Filter (MSCKF) [Li and Mourikis, 2013], or hybrid versions [Tsotsos et al., 2014; Jones and Soatto, 2011; Li and Mourikis, 2013]. EKF-based approaches have also been combined with direct photometric methods [Tanskanen et al., 2015; Bloesch et al., 2015].

The EKF framework has been used for vision only camera tracking and structure from motion [Davison et al., 2007]. It allows for straight forward sensor fusion and hence it is very popular for algorithms designed with mobile platforms in mind, which predominantly are shipped with cameras and IMUs [Hesch et al., 2013; Li and Mourikis, 2013]. However, to make the problem computationally tractable typically EKF approaches operate on sets of image-space features. As outlined above this comes with certain issues. In the filtering context a further issue is that they are uncoupled from the estimated system. It is only possible to use predicted locations to support the feature correlation or matching but the correlation itself is completely unconstrained by the overall system state. This requires costly outlier rejection (e.g., RANSAC) to detect features that where not matched or tracked correctly.

3.2.1. System Overview

The main goal of this work is the robust, fast and accurate estimation of the pose of a quadrotor without external sensing infrastructure, such as GPS or markers on the ground. Therefore, an important design goal was to perform all sensing and computing on-board, leveraging readily procurable off-the-shelf hardware only. Furthermore, due to the power constraints on small robots the algorithm needs to



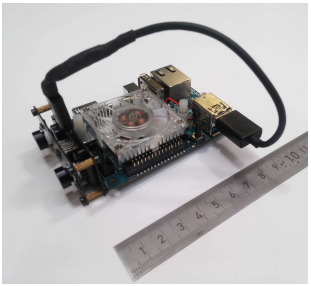


Figure 3.1.: *Left: Outdoor trajectory as estimated with our method (length: 230m, 2.9% drift). Right: Our hardware consisting of a low-cost stereo sensor and a single board ARM PC, weighing less than 100g total.*

be computationally efficient. To fulfill these constraints we contribute three main aspects:

Stereo initialization, monocular tracking: We use a small baseline stereo camera with an integrated Inertial Measurement Unit (IMU). Stereo measurements are only used to initialize the depth of features, while tracking over time is performed monocularly. This combines the efficiency of monocular approaches with reliable scale estimation via stereo.

Iterated error state Kalman Filter: An ieskf fuses IMU measurements and feature observations extracted from the camera images to estimate the position, orientation, and velocity of the camera. Furthermore we estimate additive IMU biases and a compact 3D map of feature point locations.

Anchor-centric parameterization: Feature points are parameterized by their inverse depth on an iteratively updated small set of past camera poses. This set of pose estimates are expressed in a reference frame

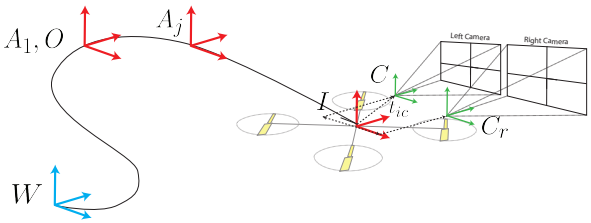


Figure 3.2.: *The coordinate frames used in the Kalman Filter framework. The camera and anchor poses are expressed in the origin frame O, which coincides with the pose of an anchor, here A_1.*

that is moved along with the current frame, keeping the camera pose and map uncertainty bounded and hence reducing drift over time.

3.2.2. Modeling

The state space of the Ieskfconsist of two parts, the Camera state and the Map state.

Camera model and state: The camera state describes the camera's estimated pose (position and orientation) and velocity, as well as the estimated accelerometer and gyroscope biases, denoted by $\mathbf{b}_a \in \mathbb{R}^3$ and $\mathbf{b}_\omega \in \mathbb{R}^3$, respectively. Further, the orientation of the origin frame in the inertial world frame, $\bar{\mathbf{q}}_{OW} \in \mathbb{R}^4$, is included. The purpose of this additional orientation is explained in Sec 3.3.2. The nonlinear process model follows the standard formulation of [Mourikis et al.,

2009].

$$\underbrace{\begin{bmatrix} \dot{\mathbf{p}}_c \\ \dot{\bar{\mathbf{q}}}_c \\ \ddot{\mathbf{p}}_c \\ \dot{\mathbf{b}}_a \\ \dot{\mathbf{b}}_\omega \\ \dot{\bar{\mathbf{q}}}_{OW} \end{bmatrix}}_{\dot{\mathbf{x}}_{\text{cam}}} = \underbrace{\begin{bmatrix} \dot{\mathbf{p}}_c \\ \bar{\mathbf{q}}_c \times \bar{\mathbf{q}}(z_\omega - \mathbf{b}_\omega) \\ \mathbf{R}_c(\bar{\mathbf{q}}_c)(\mathbf{z}_a - \mathbf{b}_a) \\ \mathbf{n}_{\mathbf{b}_a} \\ \mathbf{n}_{\mathbf{b}_\omega} \\ \mathbf{n}_o \end{bmatrix}}_{f(\mathbf{x})} \tag{3.1}$$

.

$$\mathbf{x}_{\text{cam}} = \left[\mathbf{p}_c,\ \bar{\mathbf{q}}_c,\ \dot{\mathbf{p}}_c,\ \mathbf{b}_a,\ \mathbf{b}_\omega,\ \bar{\mathbf{q}}_{OW}\right]^T \in \mathbb{R}^{20}$$

3.2.3. Point Parametrization

We parametrize the map of feature points by their inverse depths from the camera pose at which they are first seen [Pietzsch, 2008]. These past poses are termed anchor poses and denoted by $\mathbf{r}_{OA_j} \in \mathbb{R}^3$ and $\bar{\mathbf{q}}_{A_jO} \in \mathbb{R}^4$. The unit norm vector $\mathbf{r}_\mu$ encoding the ray in the anchor frame on which a feature i lies is stored statically for each map feature. For every time step where new points are initialized, the point state vector $\mathbf{x}_{\text{map}}$ is augmented with $x_{new} = [\mathbf{r}_{OA}, \bar{\mathbf{q}}_{AO}, \rho_0, ..., \rho_0]^T$ where $\mathbf{r}_{OA} = \mathbf{p}_c$ and $\mathbf{r}_{OA} = \bar{\mathbf{q}}_c$ are the current camera pose and ρ_0 the inverse depths which are set to an arbitrary value. In addition to the point state vector the location of each point in normalized image coordinates in the anchor frame is stored statically in a vector $\mathbf{r}_\mu$. The 3D position of a point can be computed as follows:

$$p_i = \mathbf{r}_{OA} + \frac{\mathbf{r}_\mu}{\rho_i} R(\bar{\mathbf{q}}_{AO}) \in \mathbb{R}^3 \tag{3.2}$$

where $\mathbf{r}_{OA}$ is the position and $R(\bar{\mathbf{q}}_{AO})$ the orientation of the according anchor frame and ρ_i the inverse depth of the point.

The 3D points are modeled as static scene points assuming that they do not move in the 3D space. Therefore, the feature space dynamics are given as $\dot{p_{f_i}} = 0$, $\dot{q_{f_i}} = 0$ and $[\dot{\rho}_1 \dots \dot{\rho}_N] = 0$.

By bundling several map features to the same anchor pose, a very efficient map state is achieved [Pietzsch, 2008; Tanskanen et al., 2015]. The total map state $\mathbf{x}_{\text{map}}$ is composed of l anchor states:

$$\mathbf{x}_{\text{map}} = \begin{bmatrix} \mathbf{x}_{\text{anch}_1}, & \dots, & \mathbf{x}_{\text{anch}_l} \end{bmatrix}^T \in \mathbb{R}^{(7+n)l}$$

$$\mathbf{x}_{\text{anch}_j} = \begin{bmatrix} \mathbf{r}_{OA_j}, & \bar{\mathbf{q}}_{A_jO}, & \rho_1, & \dots, & \rho_n \end{bmatrix}^T, \in \mathbb{R}^{(7+n)}$$

The total state of the ieskf has thus the following form:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_{\text{cam}}, & \mathbf{x}_{\text{map}} \end{bmatrix}^T \in \mathbb{R}^{20+(7+n)l \times 20+(7+n)l}$$

Due to the error state formulation, the covariance matrix needed to estimate the camera pose and l anchors, each with n features is $\mathbf{P} \in \mathbb{R}^{18+(6+n)l \times 18+(6+n)l}$.

3.3. Algorithm

We now discuss the most important aspects of the proposed algorithm.

3.3.1. Feature Initialization

Upon filter initialization or once features can no longer be tracked, new features need to be inserted into the state space. For this, salient features are extracted from both the left and right camera image and their inverse depth is initialized by triangulation in a least squares fashion. Together with a new anchor pose $\left(\mathbf{r}_{OA_j}\ \bar{\mathbf{q}}_{A_jO}\right)$, corresponding to the current camera pose estimate $\mathbf{r}_{OA_j} = \mathbf{p}_c$ and $\bar{\mathbf{q}}_{A_jO} = \bar{\mathbf{q}}_c$, these point estimates are added to the state space.

To capture that the new anchor pose estimate is identical to the current camera pose, the covariance matrix is updated with the Jacobian

of the insertion function.

$$\mathbf{P}^{+} = \mathbf{J}\mathbf{P}\mathbf{J}^{T} \qquad\qquad \mathbf{J} = \frac{\partial \delta_{+}}{\partial \delta}\Big|_{\delta=\mathbf{0}},$$

where $(\cdot)$ and $(\cdot)^{+}$ denote the time instances before and after the insertion. The parameter σ_{ρ_0} can be computed, given known camera intrinsics and extrinsics. Due to the inverse depth parameterization, σ_{ρ_0} does not depend on the feature's depth. For a monocular camera, an arbitrary, sufficiently large value must be chosen for σ_{ρ_0}.

3.3.2. Anchor-centric Estimation

In traditional approaches camera and feature locations are estimated relative to a global world reference frame and hence the uncertainty of the (unobservable) absolute camera position grows unbounded. This is detrimental to the filter performance, as large uncertainties result in large linearization errors in both the Kalman Filter propagation and update step [Martinez-Cantin and Castellanos, 2006]. Our approach circumvents this issue by marginalizing the unobservable component of the global position uncertainty out of the state space – by moving a relative reference frame with the current camera pose estimate. This improves the filter performance as less linearization error is incurred due to bounded uncertainty.

This bears similarity to so-called robo-centric EKF formulations [Castellanos et al., 2004; Civera et al., 2009; Martinez-Cantin and Castellanos, 2006; Bloesch et al., 2015]. An important difference is that in our anchor-centric approach, the reference frame O, which is chosen to coincide with one of the anchor poses, is only updated when the corresponding anchor pose is removed from the state space (see Fig. 3.2), rather than updating it on every iteration as is the case in robo-centric approaches. This is both computationally more efficient and reduces drift, as shown in experiments (see Sec 7.8).

O is moved to coincide with the anchor frame with the lowest uncertainty, denoted by A_O.

$$A_O \in A_1, ..., A_l \;\text{ s.t. }\; \|\mathbf{P}(A_O)\| \leq \|\mathbf{P}(A_j)\|, \quad j = 1, ..., l.$$

where $\|\mathbf{P}(A_j)\|$ denotes the matrix 2-norm of the entries of the covariance matrix $\mathbf{P}$ pertaining to anchor A_j.

The relative translation and rotation between the old origin frame and the new one is given by $\mathbf{r}_{O_\kappa O_{\kappa+1}} = \hat{\mathbf{r}}_{OA_O}$ and $\bar{\mathbf{q}}_{O_{\kappa+1}O_\kappa} = \hat{\bar{\mathbf{q}}}_{A_O O}$, respectively, where κ and $\kappa+1$ denote the time instances before and after the move, respectively. They are used to transform the camera pose and velocity as well as the anchor poses into the new origin frame. The bias states, $\mathbf{b}_a$, $\mathbf{b}_\omega$, and inverse depths ρ_i do not depend on the origin frame and therefore do not need to be transformed.

As the absolute position of the origin frame O in the world frame W is not observable, estimating this translation does not improve the performance of the filter. Therefore it is not part of the state space, but is stored statically and updated only when the origin frame is moved. The orientation of the origin frame in the world frame, $\bar{\mathbf{q}}_{OW}$, however, is included in the state space, as its roll and the pitch axes are observable through the gravity measurement from the IMU. Estimating these two components of the origin orientation allows for the pose estimate and the map to become aligned with gravity.

3.3.3. Prediction

Following the continuous-discrete hybrid approach suggested in [Roumeliotis et al., 1999] we perform a 4^{th} order Runga-Kutta integration of the continuous motion equations given in 3.2.2. The error covariance $\mathbf{P} = \left[\begin{smallmatrix} \mathbf{P}_{CC} & \mathbf{P}_{CM} \\ \mathbf{P}_{MC} & \mathbf{P}_{MM} \end{smallmatrix}\right]$ is propagated by:

$$\mathbf{P}^+ = \begin{bmatrix} \mathbf{P}^+_{CC_{k-1}} & \mathbf{\Phi}(t_{k+1}, t_k)\mathbf{P}^+_{CM_{k-1}} \\ \mathbf{P}^+_{MC_{k-1}}\mathbf{\Phi}(t_{k+1}, t_k)^\top & \mathbf{P}^+_{MM_{k-1}} \end{bmatrix} \tag{3.3}$$

The camera error covariance is numerically integrated by

$$\dot{\mathbf{P}}_{CC} = \mathbf{F}\mathbf{P}_{CC} + \mathbf{P}_{CC}\mathbf{F}^\top + \mathbf{G}\mathbf{Q}\mathbf{G}^\top \tag{3.4}$$

where $\mathbf{Q}$ represents the process noise and $\mathbf{\Phi}(t_{k+1}, t_k)$ is integrated by

$$\dot{\mathbf{\Phi}}(t_k + \tau, t_k) = \mathbf{F}\mathbf{\Phi}(t_k + \tau, t_k), \tau \in [0, T]. \tag{3.5}$$

The estimated state is propagated whenever measurements from the IMU become available. These measurements are affected by process noise and bias. Following [Mourikis et al., 2009; Tanskanen et al., 2015], the gyroscope and accelerometer process noise, denoted by $\mathbf{n}_g$ and $\mathbf{n}_a$, respectively, are modeled as white Gaussian noise processes with respective variances $\boldsymbol{\sigma}_a$ and $\boldsymbol{\sigma}_g$. The biases are modeled as random walks $\dot{\mathbf{b}}_\omega = \mathbf{n}_{\mathbf{b}_\omega}$, $\dot{\mathbf{b}}_a = \mathbf{n}_{\mathbf{b}_a}$, where $\mathbf{n}_{\mathbf{b}_\omega}$ and $\mathbf{n}_{\mathbf{b}_a}$ are zero mean white Gaussian noise processes. The IMU measurements and camera dynamics are modeled as in [Mourikis et al., 2009]. The dynamics of the camera state are discretized with a zero order hold strategy and are propagated using the expected values of the linear acceleration and rotational velocity. The map is assumed to be static. Thus, it remains unchanged in the propagation. Note that, after propagating the total state with the IMU measurements, the expected error state is still zero.The covariance matrix is propagated using the Jacobians of the error state dynamics [Trawny and Roumeliotis, 2005], taken with respect to the error state $\delta_x = \mathbf{0}$ and the process noise $\mathbf{n} = \mathbf{0}$:

$$\mathbf{F} = \frac{\partial \dot{\delta}_{\text{cam}}}{\partial \delta_{\text{cam}}}\Big|_{\substack{\delta_x=\mathbf{0} \\ \mathbf{n}=\mathbf{0}}} \qquad \mathbf{G} = \frac{\partial \dot{\delta}_{\text{cam}}}{\partial \mathbf{n}}\Big|_{\substack{\delta_x=\mathbf{0} \\ \mathbf{n}=\mathbf{0}}} \tag{3.6}$$

A state update is performed whenever image data becomes available. First, all currently estimated features are tracked from the previous to the current image of the left camera using the KLT tracker implemented in OpenCV[1].

Outlier rejection

Features may be badly tracked due to e.g. specular reflections or moving objects, necessitating the detection and rejection of these outliers. We apply two methods of outlier rejection consecutively.

1-Point RANSAC: This consensus based method builds on the standard RANSAC algorithm by taking into account prior information

[1] `www.opencv.org/`

about the model, dramatically reducing the computational complexity of the algorithm [Civera et al., 2009].

The residual of a randomly selected feature is computed by predicting the measurement according to the *a priori* state estimate:

$$\rho_{\mathbf{i}} = \mathbf{z}_i - \mathbf{h}(\hat{\mathbf{x}}, i), \tag{3.7}$$

where $\mathbf{h}(\mathbf{x}, i)$ is the map from the total state to image coordinates [Montiel et al., 2006]. An intermediate total state is then computed:

$$\mathbf{K}_{hyp} = \mathbf{P}\mathbf{H}_i^T \mathbf{S}_i^{-1} \tag{3.8a}$$

$$\delta_{apo}^{hyp} = \mathbf{K}_{hyp}\mathbf{r}_i \tag{3.8b}$$

$$\hat{\mathbf{x}}_{hyp} = \hat{\mathbf{x}} \boxplus \delta_{apo}, \tag{3.8c}$$

where $\mathbf{H}_i$ is the Jacobian of (7.5) taken with respect to δ_x, linearized around its expected value, $\mathbb{E}[\delta_x] = \mathbf{0}$, and $\mathbf{S}_i = \mathbf{H}_i\mathbf{P}\mathbf{H}_i^T + \mathbf{R}_i$ the measurement innovation. $\boxplus$ denotes the fusion of the *a priori* total state and the *a posteriori* error state. Linear quantities are updated additively, while rotational entries are updated multiplicatively.

Features which now have a small residual are considered inliers of this hypothesis. Equations (7.5) and (3.8) are applied repeatedly with different measurements. The iteration is stopped according to standard RANSAC criteria about the expected inlier ratio [Fischler and Bolles, 1981]. Once the algorithm has terminated, we have a set of *low innovation inliers*. The complementary set is termed the set of *high innovation candidates*, which will be further processed as described in the following. The covariance matrix is updated with the standard Kalman Filter equation:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}\mathbf{H})\,\mathbf{P}_{k|k-1} \tag{3.9}$$

Following the state update with the low innovation inliers, the high innovation candidates are further separated into *high innovation inliers* and outliers by testing their measurement likelihood:

$$\chi_i^2 = \mathbf{r}_i^T \mathbf{S}_i^{-1} \mathbf{r}_i \leq \chi_{thresh}^2. \tag{3.10}$$

The high innovation inliers are fused into the state estimate as described in the following.

Iterated state update

The Kalman gain $\mathbf{K}$ is computed using the linearization $\mathbf{H}$ of the measurement model $\mathbf{h}(\cdot)$, evaluated at the current state estimate. The computed *a posteriori* error state δ_{apo} is thus only a first order approximation of the true error state. The accuracy of the state estimate can be improved by repeatedly performing an update with a set of measurements. This is particularly the case for features with a high innovation. Therefore we apply an iterated state update according to Algorithm 4 [Gibbs, 2011; Denham and Pines, 1966] with the high innovation inliers. The iteration is stopped if a maximum number of

Algorithm 4 ieskf State Update

Require: Previous state estimate: $\hat{\mathbf{x}}$,$\mathbf{P}$

$\eta^0 = \hat{\mathbf{x}}$
$\delta_\eta^0 = \mathbf{0}$
for $j = 0$ to `max_it` **do**
  $\rho^j = \mathbf{z} - \mathbf{h}(\eta^j)$
  $\mathbf{H}^j = \frac{\partial \mathbf{r}^j}{\partial \delta_x}\Big|_{\hat{\mathbf{x}}=\eta^j}$
  $\mathbf{S}^j = \mathbf{H}^j \mathbf{P} (\mathbf{H}^j)^T + \mathbf{R}$
  $\mathbf{K}^j = \mathbf{P} (\mathbf{H}^j)^T \left(\mathbf{S}^j\right)^{-1}$
  $\delta_\eta^{j+1} = \mathbf{K}^j \left(\mathbf{r}^j + \mathbf{H}^j \delta_\eta^j\right)$
  $\eta^{j+1} = \delta \boxplus \delta_\eta^{j+1}$
  if $\|\delta_\eta^{j+1}\|$ small **then**
    Stop iteration
  end if
end for
$\hat{\mathbf{x}}^+ = \eta^{j+1}$
$\mathbf{P}^+ = \left(\mathbf{I} - \mathbf{K}^j \mathbf{H}^j\right) \mathbf{P}$

iterations has been reached or when δ_{apo} is very small. Note that, irrespective of the number of iterations performed, the covariance matrix $\mathbf{P}$ is updated only once.

3.4. Experimental Results

3.4.1. Hardware Setup

Our localization system consists of a small baseline stereo camera connected to a single-board ARM computer on which the presented algorithm runs. The system is depicted in Fig. 3.1. Both devices are commercially available and affordable and make for a very small and light-weight system, weighing less than $100g$. Such a small and portable form-factor makes the localization system suitable for a large variety of applications, particularly the use on board drones designed to fly in the close vicinity of people.

We use a DUO MLX camera by Duo3d[2], featuring two monochrome global shutter cameras with a $30mm$ baseline and a 6 degree of freedom IMU. Inertial measurements are provided at 100 Hz, while the image frame rate is configured at 50 Hz with a resolution of 320x240 pixels. The VIO algorithm runs on a Hardkernel Odroid XU4[3], equipped with a Samsung Exynos5422 ARM processor. This ARM based micro computer is equipped with a Samsung Exynos5422 processor which features four Cortex-A15 and four Cortex-A7 CPU cores. The presented VIO algorithm is implemented as a single core process, leaving the other processors available for other computations, such as position control or person tracking.

For flight experiments we mount the VIO system on a Parrot Bebop[4] equipped with a PixFalcon PX4 Autopilot. running a customized version of the Px4[5] Autopilot software.

[2] www.duo3d.com/product/duo-minilx-lv1
[3] www.hardkernel.com/main/products/prdt_info.php
[4] http://www.parrot.com/products/bebop-drone/
[5] www.px4.io

3.4.2. Experiments

We demonstrate the performance of the presented VIO system with several experiments. In particular, we show its utility for the usage on small, agile robots such as drones.

Hand-held accuracy

Table 3.1.: Hand-held Trajectory of Length 230 m

Experiment	1	2	3	4	5	Mean
Rel. drift [%]	2.88	3.57	2.96	3.23	3.43	3.21

As baseline and for comparison with the current state-of-the-art, we evaluate the system's accuracy in a hand-held scenario, where we walk around several buildings, a $230m$ long trajectory, and compute the relative position drift of the trajectory. One such trajectory is shown in Fig. 3.1. The same experiment is performed several times to assess repeatability. The relative drift of each repetition of the experiment is shown in Table 3.1. We observe that the system reproducibly estimates its trajectory accurately. Note that all computations are performed on-board, whereas the literature often reports results from off-board computations.

Fast motions

To evaluate the system's robustness and ability to track fast motions, we place the sensor in a foam cube and throw it we throw the system back and forth over a distance of $4m$. The estimate is compared to ground truth form a motion capture system in Fig. 3.3. Fig. 3.3 shows the speed reached by the device. Fig. 3.4 shows three frames of the experiment with the corresponding time instances labeled in Fig. 3.3. The system successfully tracks ego motion even at very high speeds of close to 6 m/s. Despite the high accelerations and motion

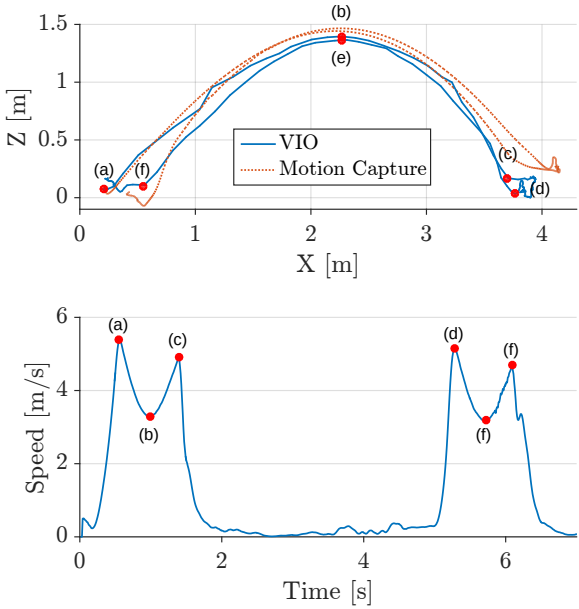


Figure 3.3.: *Fast motion experiment. The VIO system is thrown over a distance of 4m. The estimated trajectory is compared to ground truth data from a motion capture system.*

blur present in this scenario, the estimated trajectory does not deviate significantly from the ground truth. The ability to track fast motions The ability to track fast motions is interesting particularly for drone applications, where the presented system is expected to enable faster flight compared to an optical flow sensor, which is limited to tracking speeds of about $2m/s$ [Honegger et al., 2012].

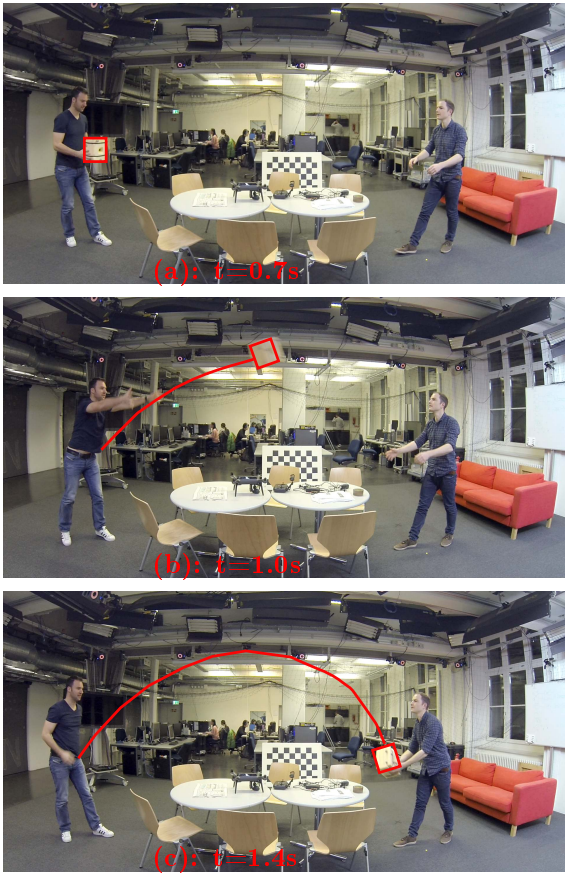


Figure 3.4.: *Still images of the throwing experiment. The labels denote the same time instances as in Fig. 3.3*

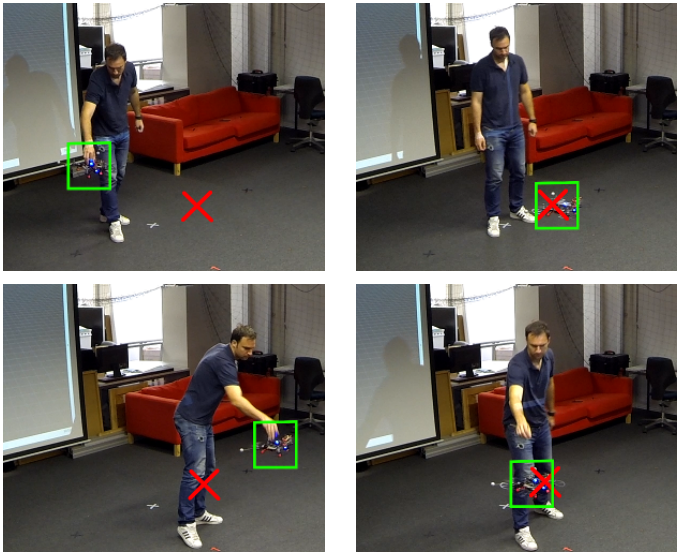


Figure 3.5.: *Still images of the disturbance rejection experiment. The quadrotor (green) is disturbed from its setpoint (red) and returns to it.*

Fast flight

Fast outdoor flight, faster than possible with downward optical flow. We already saw in the throwing experiment that speeds faster than optical flow are possible.

In-flight ground truth comparison

Furthermore, we We demonstrate the performance of the system during flight where a quadrotor is controlled to repeatedly fly a predefined figure-8 trajectory, using positional information from a motion capture system for reference. The trajectory has a length of 123 m and a

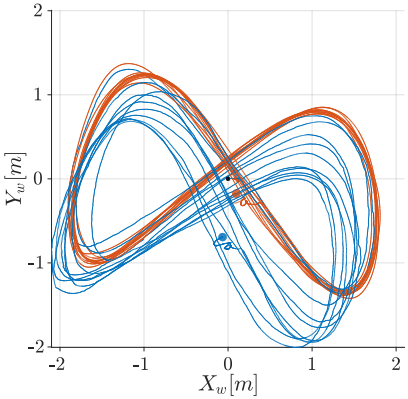


Figure 3.6.: *In flight trajectory estimated by VIO (blue) compared to ground truth (red).*

duration of 2 minutes and 47 seconds. The quadrotor is controlled to move along this trajectory using positional information from a motion capture system. We compare the trajectory as estimated by our algorithm We compare the estimated trajectory with the ground truth in Fig. 3.6. The flown trajectory is $123m$ long and the estimate shows a drift in position of 0.46% and 1.17% yaw drift relative to the ground truth. We note that the position drift in y appears significantly bigger than in x. This difference can be explained by observing that the drift in x and y is not independent of drift in yaw. Drift in y appears significantly bigger than in x, which is due to the fact that positional drift is not independent of drift in yaw. When the estimated yaw angle deviates from the true value, the estimated and true world frames are no longer aligned. This can result in a significant apparent position drift. Inspection of Fig. 3.6 suggests that the under-estimation of the yaw angle causes an under-estimation of the world y position.

Disturbance rejection

The system's capabilities in a control loop of a quadrotor is tested by commanding the quadrotor to hover at a set-point and repeatedly disturbing it. Fig. 3.5 shows still frames from this experiment and we observe that the quadrotor returns to its set-point after each disturbance.

Validation of anchor-centric approach

The anchor-centric parameterization is relevant particularly for longer trajectories. We analyze its effect based on an outdoor trajectory. Fig. 3.7 shows the estimated trajectory with the anchor-centric parameterization in red and with a world centric parametrization in green.

We observe drift in yaw, as seen by the misalignment of the trajectories as well as drift in position of close to three times as much as with the anchor-centric parameterization.

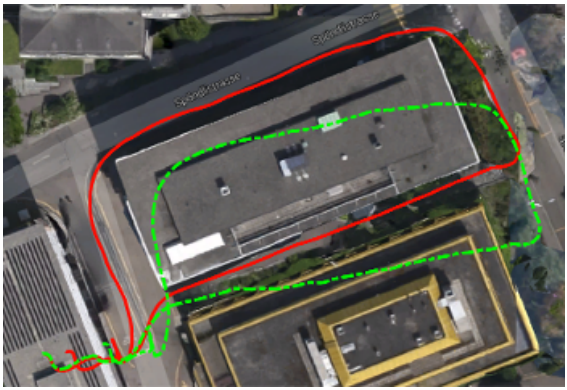


Figure 3.7.: *The trajectory estimated by the anchor-centric parameterization (red) is compared to the estimate with a world centric parameterization (green). The trajectories are aligned with each other and the satellite image at the start.*

3.5. Conclusion

We presented a Visual-Inertial Odometry system that relies solely on off-the-shelf and light-weight components. All computations are performed on-board an embedded ARM processor and no external sensors or beacons are required. We have demonstrated that the system is able to accurately track long trajectories and is robust with respect to fast motions. We have further shown that the system enables the stabilization of a Micro Aerial Vehicle's position in flight. The affordable hardware in combination with the algorithm available as open-source software presents a pose estimation system that is easily incorporated in a wide variety of applications.

Chapter 4.

Semi-Direct EKF-based Monocular Visual-Inertial Odometry

In the last chapter 3 we presented a method to accurately estimate the metric position in real-time given a small baseline stereo camera and an inertial measurement unit. The proposed algorithm uses a separation between feature tracking, using a KLT tracker, and pose estimation. In order allow also using features with bad image gradients on one image axis, we propose a monocular visual inertial odometry algorithm that combines the advantages of EKF-based approaches with those of direct photometric error minimization methods. The method is based on sparse, very small patches and incorporates the minimization of photometric error directly into the EKF measurement model so that inertial data and vision-based surface measurements are used simultaneously during camera pose estimation. We fuse vision-based and inertial measurements almost at the raw-sensor level, allowing the estimated system state to constrain and guide image-space measurements. The proposed formulation allows for an efficient implement-

ation that runs in real-time on a standard CPU and has several appealing and unique characteristics such as being robust to fast camera motion, in particular rotation, and not depending on the presence of corner-like features in the scene. We experimentally demonstrate robust and accurate performance compared to ground truth and show that our method works on scenes containing only non-intersecting lines.

4.1. Introduction

The problem of estimating the motion of a camera relative to a known 3D scene from a set of images or RGB-D (RGB and depth) frames is one of the fundamental problems in computer vision and robotics. Estimating camera motion enables applications such as vehicle or robot localization [Nistér et al., 2004], 3D reconstruction [Tanskanen et al., 2013] and augmented reality [Schöps et al., 2014]. Recently a number of approaches have leveraged dense RGB-D data, available in real-time from depth sensing cameras such as the Kinect [Zhang, 2012], in combination with ICP-like algorithms for pose estimation [Izadi et al., 2011; Kerl et al., 2013; Newcombe et al., 2011a].

Similar in philosophy but using monocular images only, methods for camera pose estimation using dense surface measurements have been demonstrated [Matthies et al., 1988; Newcombe et al., 2011b; Wendel et al., 2012]. These methods use all data available for pose estimation and hence promise high tracking accuracy and robustness. However, they are computationally expensive and typically require powerful GPUs for real-time performance, prohibiting use in mobile and compute restricted setups.

Direct methods, minimizing photometric error for pose estimation, have recently been adapted to sparse formulations [Engel et al., 2013; Forster et al., 2014] with great success. These methods offer higher precision and robustness than traditional feature extraction and tracking based methods [Klein and Murray, 2007] and, in the sparse variant, have comparable or better runtime performance.

Being purely vision based, these methods struggle under fast motion, in particular rotation [Engel et al., 2013], when the camera is moving along it's focal axis, and in scenes with few corner-like features [Forster et al., 2014]. Most direct photometric approaches are formulated as energy minimization problem and leverage Gauss-Newton like methods to solve for camera pose. Therefore, tightly coupling IMU and vision data is non-trivial in these frameworks. On the other hand filter-based approaches to VIO [Davison et al., 2007; Li and Mourikis, 2013; Hesch et al., 2013] tightly couple inertial measurements with visual data and have demonstrated robustness to fast rotation, partial loss of visual tracking and relatively little drift over time. However, we are not aware of exisiting methods to incorporate direct methods (i.e., photometric error minimization) directly in the measurement model of the EKF framework. In this chapter we propose, to our best knowledge, for the first time an algorithm that combines the use of direct photometric error minimization in an extended kalman filter (EKF) framework. Allowing us to fuse vision and inertial data tightly, almost at the raw sensor level. Both signal sources measure the same motion but have different, complementary sensor characteristics which can provide additional constraints during the optimization camera pose. Fusing the complementary data sources at the lowest possible level allows the estimated system state to constrain and guide the image-space measurements, enforcing consistency between image-space feature positions and 6DOF camera motion. Our approach works with very few (10-20) and very small (as small as 3×3) image-patches. This sparsity allows for an efficient and fast implementation. Furthermore, the method can handle scenes that do not have any corner-like features and hence is suitable for scenarios in which other methods fail.

4.1.1. Related Work

Dense methods

Dense direct methods operate on surface measurements directly, either depth estimates of a stereo camera or a RGB-D sensor [Kerl et al.,

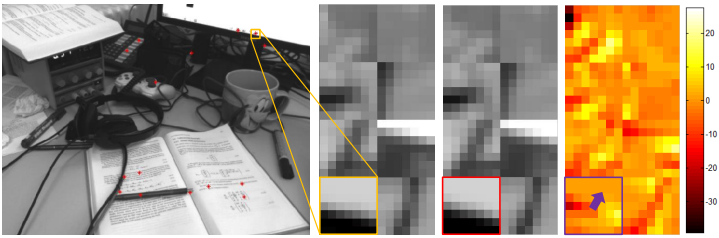


Figure 4.1.: *The left image shows the pixel patches selected for odometry computation on the current camera image. The middle two images show a selection of the pixel patches in the current image and the respective reference patches. The algorithm is optimizing the camera pose and the patch depth by minimizing the intensity residual. The rightmost image shows the intensity residuals with an arrow illustrating the patch motion that is needed to align both patches resulting from the image gradient of the current image.*

2013; Newcombe et al., 2011a], and do not extract sets of features from this data. These approaches require heavy GPU parallelization due to computational cost and tend to have restricted working ranges, due to sensor working principles. Dense monocular methods do not have special sensor requirements but have similar computational costs because they require the build-up of an explicit cost volume[Newcombe et al., 2011b] or on computing constrained scene flow [Newcombe and Davison, 2010].

Semi-dense direct methods

Recently [Engel et al., 2013] proposed to estimate depth only for pixels in textured image areas and introduce an efficient epipolar search, enabling real-time visual odometry and semi-dense point cloud reconstruction on a standard CPU and even on mobile platforms [Schöps et al., 2014]. Photometric alignment on sparse, known 3D points has been used by [Forster et al., 2014] to improve accuracy and robustness of the standard SLAM pipeline of [Klein and Murray, 2007]. Most of these approaches either do not use inertial data or treat both data sources mostly independently and only fuse the two at the camera pose level. For example, to estimate metric scale on top of vision based camera pose [Weiss and Siegwart, 2011].

Visual inertial odometry

To improve feature correlation results, early SLAM approaches have used photometric error and patch-wise normal estimation [Molton et al., 2004] to improve feature correlation but this was done separately from the standard EKF-SLAM steps. Instead of externally optimizing the homography between filter updates, [Jin et al., 2003] estimates the patch normal inside the EKF framework. The drawback with these methods is that the local patches have to be reasonably large (25×25 pixel or larger) for the normal to be estimated robustly. This increases computational cost and introduce problems

with patches near depth discontinuities, where the texture in a patch would not change consistently with camera motion.

4.1.2. System Overview

Our technique is a visual-inertial odometry (VIO) approach, this means that camera pose is estimated only from currently visible regions of the observed 3D scene and we do not maintain a global map of previously extracted feature points, we remove all features from the state space as soon as they leave the field of view of the camera. Note that the proposed approach could easily be extended with standard mapping back-end as for example in [Klein and Murray, 2007]. Following the approach in [Mourikis et al., 2009] We reformulate the EKF framework which has been used successfully for structure from motion [Davison et al., 2007] into an *Error State Extended Kalman Filter* ErKF.

Fig. 7.1 illustrates our approach. A small number of small patches were extracted in previous frames and the corner locations of the patches are projected into a predicted camera pose based on IMU data. An affine warp for the whole patch is computed (cf. Fig. 4.3). The algorithm then jointly optimizes the camera pose and the patch depth by minimizing the intensity residual. One advantage of this approach is that we do not rely on the extraction of features of a specific type (e.g., corners) but can use any patches with sufficient gradient. In particular, patches which lie on lines (see highlighted region in Fig. 7.1) or they can be placed in image areas with good texture, similar to the pixel selection in (semi-)dense approaches [Engel et al., 2013]. Furthermore, we use an inverse depth parametrization for the patch depth which allows us to start tracking without a special initialization sequence as it is necessary with other approaches [Forster et al., 2014; Klein and Murray, 2007].

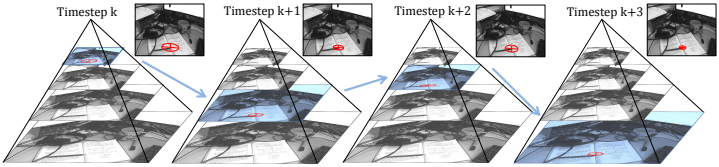


Figure 4.2.: *The optimal pyramid level for pixel patch alignment is selected based on the estimated variance of the projected point location in pixel space. If the camera motion is fast, higher levels are used, if there is low variance on the camera pose lower levels are used for higher precision.*

4.2. Modeling and State Propagation

We use the same feature parametrization and process model as presented in chapter 3. Therefore, we omit the repetition and refer the reader to the previous chapter.

4.3. Photometric Update

The photometric update is different from standard visual odometry approaches that use 2D image positions from an external feature tracker or matcher. In our case the measurement model $h(x)$ is used to directly predict the appearance of a pixel patch (the 1 dimensional intensity values of the pixels) of a reference view given the pixel values in the current camera view (see Figure 4.3).

More specifically, for every pixel patch the current estimate of the 3D location of the center pixel is transformed into the current camera frame. The observation of a feature i defines a ray from the camera optical center to the feature given by:

$$\mathbf{h}_i = \mathbf{R}_c \left(\rho_i (\mathbf{r}_{OA_i} - \mathbf{p}_c) + \mathbf{R}(\bar{\mathbf{q}}_{A_iO}^T) \mathbf{r}_\mu \right), \tag{4.1}$$

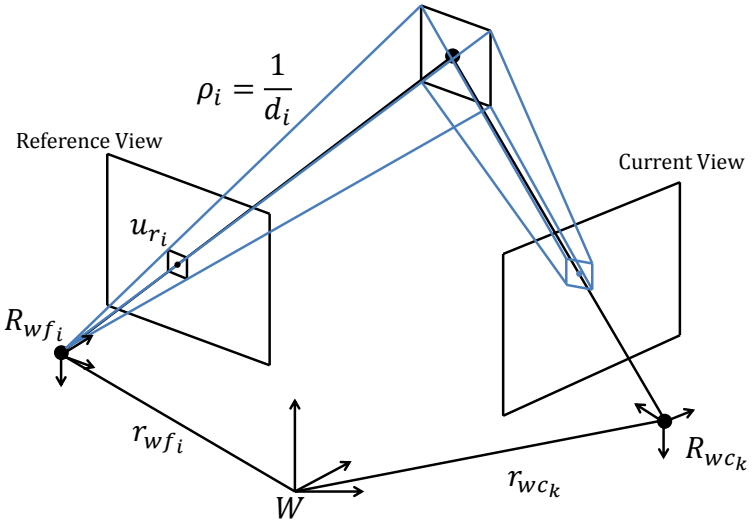


Figure 4.3.: *Estimating per-pixel intensity differences. Given a reference camera pose $R_{wf_i}|r_{wf_i}$ at time i and predicted camera pose $R_{wc_k}|r_{wc_k}$ the center location u_{r_i} of a patch in the reference view is projected into 3D world coordinates and re-projected into the current view. We compute an affine warp to transform the pixel coordinates of all pixels in the small patch around the point location in the current camera view into the reference view. The per-pixel intensity value differences form the residual to minimize.*

where $\mathbf{h}_i$ is a vector from the predicted current camera center towards the 3D location of the patch center. The measurement function $\mathbf{h}_{I_i}(\cdot)$ is then used to compute the appearance of the reference pixels given the current state estimates and the current camera image:

$$\mathbf{h}_{I_i} = I(\pi(\mathbf{h}_i)) \ . \tag{4.2}$$

Here, the measurement model equation $\mathbf{h}_{I_i}(\cdot)$ is given for a single pixel. In order to increase robustness we extend the single measurement to a patch around this point. In doing so we make the assumption that the scene around this point is planar because only the depth of the single point is modeled. However, since we are using small patches (3×3pixels), the assumption of a locally flat scene can be made similar to [Forster et al., 2014]. To further reduce the degrees of freedom, we assume the patch normal to be orthogonal to the image plane in the anchor frame. This assumptions allows us to model the appearance of the pixels surrounding the center point via an affine warp A, encoding in-plane rotation of the patch, the depth dependent size of the patch and some shear caused by a camera observing the patch from a different angle.

The residual $\rho_{\mathbf{I_i}}$, recursively minimized during camera pose estimation by the Kalman filter, is the photometric error between all the pixels in the reference patch and the pixels in the warped patch, extracted from the current camera view:

$$\rho_{\mathbf{I_i}} = I(\mathbf{m}_{x,y}) - I_r(\mathbf{m}_{x,y}) \ , \tag{4.3}$$

with I the current image, I_r the reference image. This residual is computed for all points that are currently in the state space.

Finally, the Kalman filter update step requires the linearization of the measurement function H, computed as the derivative of $h(x)$ with respect to the states x:

$$H = \frac{\partial h(x)}{\partial x} = \nabla I_k \frac{\partial \pi(h_c)}{\partial h_c(x)} \frac{\partial h_c(x)}{\partial x} \ , \tag{4.4}$$

with ∇I_k being the image gradient of the warped patch extracted from the current frame. Assuming familiarity with the EKF framework, the equations given here and in the previous sections should be sufficient to implement the proposed algorithm. However, there are a number of details that can be taken into consideration in order to increase robustness in real-world settings and improve performance. We briefly discuss these in the following subsections.

4.3.1. Patch Extraction

The patches can be selected using many different methods and in particular there is no requirement for patches to be centered on corners. In the experimental section we demonstrate the performance of our technique using only patches that are centered on single, non-intersecting lines. A simple implementation could just extract FAST keypoints [Rosten and Drummond, 2005] on an uniform grid. However, we noticed that selecting image areas based on the Shi-Tomasi score that are stable over the whole scale space of the image pyramid leads to better and more stable results.

4.3.2. Image Pyramid Level Selection

In our method we attain predictions and associated uncertainties for all state variables and their covariances. This can be used to compute the variance on the point location in image space and consequently allows to select the optimal level in the image pyramid such that convergence is guaranteed (see Fig. 4.2). Compared to the standard approach of iterating through the whole image pyramid starting from the highest level, this approach saves computation time while still offering the advantage of a larger convergence radius of the higher pyramid levels and the precision of the lower levels. In addition as the method selects the lowest possible level for convergence, it also reduces the risk of converging towards a wrong local minimum if the optimization on higher levels would not converge towards the correct image location.

The error covariance can be computed by omitting the image gradient when taking the derivative of the measurement function $\frac{\partial h}{\partial x}$:

$$H_\pi = \frac{\partial \pi}{\partial \mathbf{h}_i} \frac{\partial h_c}{\partial x} \ , \tag{4.5}$$

$$S_\pi = H_\pi P_{k-1|k-1} H_\pi^\top \ . \tag{4.6}$$

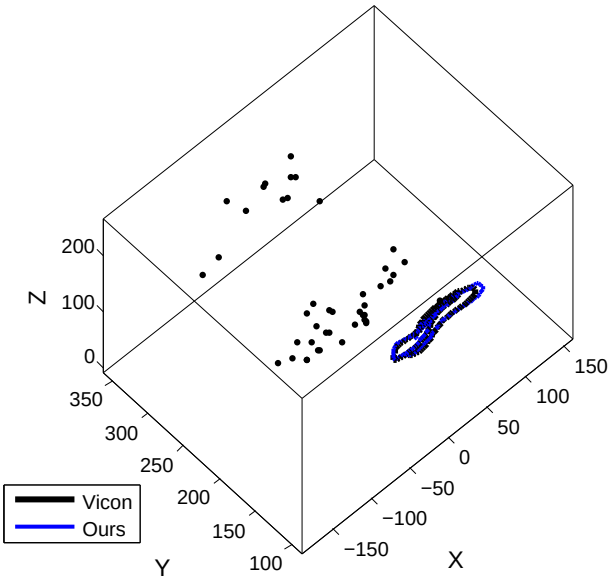
The major axis of the error ellipsoid in image space is then the largest Eigenvalue of the 2×2 matrix S_π. To guarantee convergence the length of this axis should be smaller than 1 pixel at the respective pyramid level. These calculations can be done while computing the derivate H during the update step before the image gradient of the pixel patch is computed. The only overhead is the computation of the 2×2 matrix S and its Eigenvalues.

4.3.3. Iterated Sequential Update

Inherently our formulation requires the processing of many measurements for each update step (every pixel is a measurement). Unfortunately this impacts runtime performance. The size of the Jacobian $\frac{\partial h}{\partial x}$ and as consequence, the size of the innovation covariance matrix S will be $ns \times ns$, where n is the number of patches and s the patch size in pixels. Because S needs to be inverted during every EKF update step, the size of S directly impacts the runtime. We use the same algorithm (see Alg. 4) as proposed in the previous chapter 3. In the case of the linear Kalman filter, sequential updates [Anderson and Moore, 2012] can be utilized to alleviate this situation. We observed that if one iteratively re-linearizes the measurement matrix $H = \frac{\partial h}{\partial x_{seq}}$ around each updated estimated state sequentially, the algorithm produces very good estimates in practice. The sequential update reduces the computations to n inversions of a $s \times s$ matrix which drastically enhances runtime performance.

4.4. Experimental Results

We performed a comparison against ground truth acquired from a Vicon system and two of recently published methods that use a photometric approach in a semi-dense manner [Engel et al., 2013] and patch-based on corner locations [Forster et al., 2014]. We moved the camera around a regular office space, see an example image from the dataset in Figure 4.4 on the left. The top plot shows the 3D view of the final positions of the tracked points during the sequence and the trajectory of our method together with ground truth. The third plot shows the position in all axes, as can be seen, our method tracks the camera pose in typical scenes with equal quality than the compared methods. The initialization for [Engel et al., 2013] was difficult in this particular scene and its performance did not match the expected level. The most compelling advantage of our constrained direct method is that it do not rely on the presence of corner like features. In particular, our implementation works on scenes that *only* contain (non-intersecting) lines. Figure 4.5 shows a demonstration of such a scene. It is clear that methods that rely on external trackers like KLT will fail in this scenario since the tracker is not able to fix the tracked points at a position and thus the point will start to randomly slide along the edge. Since in our implementation the location of the patches are constrained by the model in the filter, the algorithm is able to fully initialize with patches that lie on these kinds of edges even with a patch size of only 3x3 pixels. Figure 4.6 shows the results of a challenging dataset with a camera moving in front of a curtain having almost only line-like structure in view. Only few patches where placed on corner-like areas, this was enough to fix the camera pose from drifting in vertical direction. This demonstrates that the proposed method can be used to track scenes that are rather hard for methods that rely on unconstrained feature correspondences as is the case in many indoor scenes. The runtimes for the photometric update in unoptimized C code from MATLAB on a Core i5 desktop computer is 12 ms, thus already allowing for real-time use. We plan to implement a fully optimized version for mobile ARM CPUs.



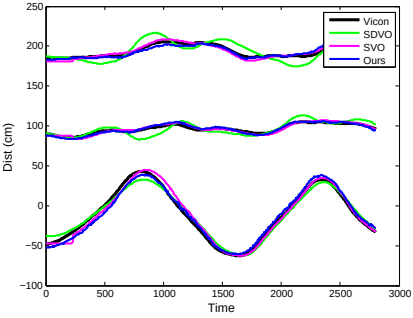


Figure 4.4.: *Blue: Trajectory from the presented algorithm, Magenta: Semi Direct Visual Odometry (SVO) [Forster et al., 2014], Green: Semi-Dense Visual Odometry (SDVO) [Engel et al., 2013], Black: Ground Truth (VICON data). The initialization of SDVO had issues in the selected scene, due to the suboptimal initial map the performance is not as good as can be expected.*

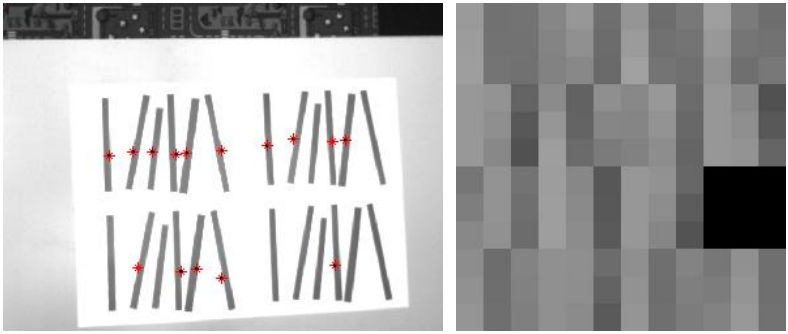


Figure 4.5.: *Thanks to the constraints on the pixel patches, the algorithm is able to initialize even on this difficult scene consisting only of almost vertical lines. On the right side the used 3x3 pixel patches are visible.*

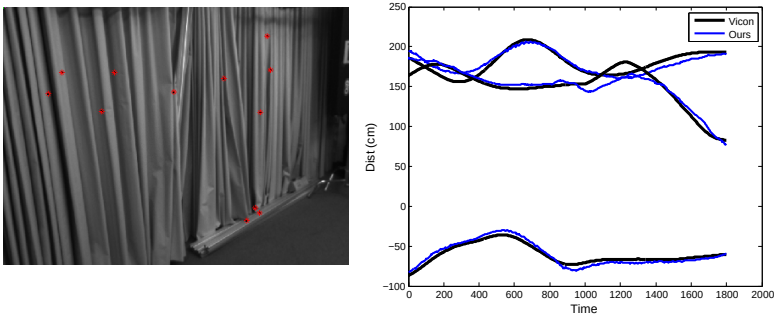


Figure 4.6.: *Visual-Inertial odometry on a scene with lines. Left: patches on lines. Right: comparison with ground truth (VICON data).*

4.5. Conclusion

In this chapter we presented a novel, Kalman filter-based semi-direct visual inertial odometry approach that combines the advantages of a tightly coupled visual-inertial Kalman filter and the robustness and precision of direct photometric methods. We demonstrated how the photometric update can be built into a standard error-state Kalman filter odometry algorithm. We proposed an efficient implementation that reduces the impact of the larger number of measurements when minimizing the photometric residual error. Finally, we demonstated that our proposed algorithm matches the tracking quality of other state of the art approaches, and in addition, thanks to the rigid scene constraints the proposed algorithm can work with pixel patches lying only on line-like structures and is even able to fully initialize without special procedure in such scenes.

Part II.

Real-time Drone Cinematography

Chapter 5.

Multi-Subject Filming and Viewpoint Optimization

In the first part of the thesis we described how we can accurately estimate the position of the flying drone. In the next part, we now assume, the position of the drone and all the subject is known and we describe how we can use this information to develop the necessary algorithms for the proposed intelligent drone cinematography.

In this chapter, we propose a method for real-time motion planning with applications in aerial videography. Taking framing objectives, such as position of targets in the image plane as input, our method solves for robot trajectories and gimbal controls automatically and adapts plans in real-time due to changes in the environment. We contribute a real-time receding horizon planner that autonomously records scenes with moving targets, while optimizing for visibility to targets and ensuring collision-free trajectories. A modular cost function, based on the re-projection error of targets is proposed that allows for flexibility and artistic freedom and is well behaved under numerical optimization. We formulate the minimization problem under constraints as a finite horizon optimal control problem that fulfills aesthetic ob-

jectives, adheres to non-linear model constraints of the filming robot and collision constraints with static and dynamic obstacles and can be solved in real-time. We demonstrate the robustness and efficiency of the method with a number of challenging shots filmed in dynamic environments including those with moving obstacles and shots with multiple targets to be filmed simultaneously.

5.1. Introduction

Robotics and drones in particular are rapidly becoming an end-user facing technology. In particular, the application domain of filming with drones currently receives great interest from industry and consumers. Professional camera teams leverage consumer-grade robots to create stunning visuals that previously required a helicopter and expensive camera gear. However, manually flying quadrotors remains a surprisingly hard task. Furthermore, automated flight modes in current commercial offerings are restricted to simple circling or target following.

Several algorithms have been proposed for the planning of quadcopter trajectories [Gebhardt et al., 2016; Joubert et al., 2015; Roberts and Hanrahan, 2016], taking both aesthetic objectives and the physical limits of the robot into consideration. These methods make the task of trajectory planning easier for non-experts. However, current approaches employ global optimization techniques, planning the entire trajectory a priori. While [Joubert et al., 2015] allows to adjust the velocity along the planned trajectory at execution time, none of the existing methods are capable of re-planning a suitable trajectory in real-time, for example to avoid collisions with dynamic obstacles or to film targets that move in unpredictable ways (e.g., human actors).

In this chapter we propose an approach to the automatic generation of quadrotor and gimbal controls in real-time while ensuring physical feasibility. In particular, we contribute a fast receding horizon planner based on numerical optimization for automatic aerial cinematography. The method takes high-level aesthetic objectives given

by a cameraman as input and automatically records the scene while the targets move in an a priori unknown way. The system fulfils the high-level framing objective, in the least squares sense, while ensuring collision-free paths. The resulting appearance of targets is specified via set-points in screen space (extending ideas outlined in [Gleicher and Witkin, 1992]) and the method minimizes the re-projection error alongside the viewing direction and scale of the target projections in real-time. We formulate this cost minimization problem under constraints as a finite horizon model predictive control (MPC) non-linear program with the following properties: (i) a main utility function to fulfill - as close as possible - the specifications from the user in a dynamic scene; (ii) input and state constraints to respect the dynamics and model constraints of the drone; (iii) constraints for collision avoidance with obstacles and the targets being recorded. The resulting optimization problem can be solved numerically with state-of-the-art solvers in real-time. The inputs computed are applied for the first time step, and the optimization is repeated at the next sampling instance with updated information about the quadrotor state, obstacles and target positions. We believe that the method is general enough and could be adapted to other tasks for drones which include collision avoidance with respect to moving obstacles.

5.2. Related Work

Quadrotor trajectory generation is a well studied problem and various approaches have been proposed, including early work on MPC for collision avoidance applied to aerial vehicles [US et al., 2003; Richards and How, 2004], global forward planning approaches to generate minimum snap trajectories [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] based on a convex MPC formulation. Outside of the field of aerial vehicles, constrained optimization methods have also been proven to be powerful tools for trajectory generation [Schulman

et al., 2014]. We also formulate our problem in the MPC framework to attain real-time performance, using the full non-linear dynamics of the quadrotor and extending trajectory generation to include collision avoidance with dynamic targets and to respect cinematographic objectives. Automatic camera control has been studied extensively in the context of virtual environments in computer graphics. We refer to the comprehensive review in [Christie et al., 2008b], with the majority of methods using discrete optimization formulations. Notably in virtual environments it has become somewhat of a standard to define viewing constraints in screen-space [Gleicher and Witkin, 1992; Drucker and Zeltzer, 1994; Lino et al., 2011]

However, virtual environments are not limited by real-world physics and robot constraints and hence can produce arbitrary camera trajectories, velocities and viewpoints.

Extending the work on trajectory generation [Mellinger and Kumar, 2011], several algorithms for planning aerial video shots of (mostly) static scenes [Gebhardt et al., 2016; Joubert et al., 2015; Roberts and Hanrahan, 2016] exist. Joubert et al.'s method [Joubert et al., 2015] allows an operator to adjust the velocity of the quadrotor at execution time to keep moving targets in-frame. However, the method cannot re-plan the trajectory in cases where the target does deviate from an a-priori known path or in cases were dynamic obstacles temporarily obstruct the original trajectory. Lino et al. [Lino and Christie, 2015] propose a method to generate camera paths in virtual environments that ensure visibility of the faces of two actors in the image. This concept was later extended to quadrotors [Galvane et al., 2016], albeit limited to obstacle free environments. To record a single moving target with multiple quadrotors, [Poiesi and Cavallaro, 2015] introduced a particle swarm method for formation control. The method is again limited to obstacle-free environments and only tested in simulation. We propose a general constrained optimization approach for automatic viewpoint and trajectory computation allowing for multiple moving targets and obstacles in the environment. The approach can be solved in real-time.

Figure 5.1.: *Illustration of cinematographic framing constraints: size, viewing angle and position on screen (from left to right).*

Our method also accounts for occlusions and loosely relates to the fields of target tracking [Hausman et al., 2016], visual servoing [Espiau et al., 1992], active vision [Aloimonos et al., 1988b] and persistent monitoring [Smith et al., 2011], where our focus is on videographing a set of moving targets.

5.3. Preliminaries

5.3.1. Cinematographic Objectives

While film making is a form of art and relies on human creativity and intuition, many aspects of it have been studied and categorized forming a 'grammar' of film [Arijon, 1976]. Fig. 5.1 summarizes the most important aspects. We later formalize these mathematically for use in a cost minimization algorithm. In particular, we are interested in framing objectives – that is formal rules that specify how objects should appear on the screen. The first important notion is that of distance. Shots can be categorized into five types of shots (close-up, close-medium, full and long shots), see Fig. 5.1, left. A further important aspect is that of the relative viewing angle, which can also be categorized into ranges of pan- and tilt-angles), see Fig. 5.1, middle. Finally, the screen position of a filmed target is important in order to create aesthetically pleasing footage. In particular the rule of thirds

(cf. [University of Dallas, 2016]) prescribes the placement of significant vertical and horizontal elements along the horizontal and vertical thirds (Fig. 5.1, right).

Shot framing requirements

From the above film grammar we can extract objectives for an optimization method. In particular, we consider (i) the *screen position* of targets, where we seek to minimize the distance between the desired position on screen and the projection of the actual (3D) target position onto the screen. (ii) To relate shot distances to positions, we seek to optimize the *projected size* of a given target. (iii) Furthermore, we require the algorithm to consider the *relative viewing angle* between target and camera center, for example to keep the face of a person in view. (iv) Finally, we require the algorithm to account for an externally set *camera pose*, which could be specified directly by the user or a high-level global planning algorithm.

5.3.2. Target Model

We denote the position of each actor, or moving target, by $\mathbf{p}_t \in \mathbb{R}^3$, with K the number of targets. We then assume that the human motion is based on a constant velocity model $\ddot{\mathbf{p}}_t = \mathbf{q}_\eta$ where $\mathbf{q}_\eta \sim \mathcal{N}(0, \sigma_t)$ is Gaussian noise. Given the current observations, we predict future positions of the actor with a standard linear Kalman filter [Gibbs, 2011].

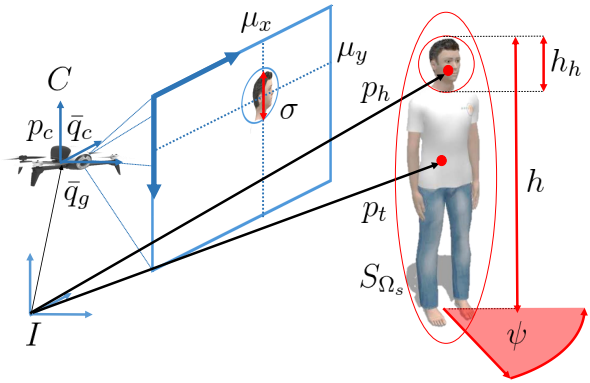


Figure 5.2.: *Coordinate frames and physical quantities used in our method, here shown on the example of a human actor. The pose of the flying camera is given by its position* $\mathbf{p}_c$*, its orientation* $\bar{\mathbf{q}}_c$ *and the orientation of the gimbal* $\bar{\mathbf{q}}_g$*. The target is modeled by an ellipsoid* S_{Ω_s} *centered at* $\mathbf{p}_t$*. The flying camera avoids collisions with this volume. In this case the objective is to record the head of the target, modeled as an oriented sphere with direction given by* ψ*, size* h_h *and position* $\mathbf{p}_h$*. The projection of the head onto the camera image space has size* σ *and is centered at* μ_x*,* μ_y*.*

5.4. Trajectory Generation for Viewpoint Optimization

The goal of our algorithm is to find a feasible and locally optimal trajectory for the quadrotor and gimbal control inputs in real-time. The method produces imagery that minimizes the deviation with respect to the high-level image space specifications of Sec. 5.3.1 and gener-

ates trajectories that guarantee collision-free motion with respect to dynamic obstacles. A scene is formed by K targets to be filmed, e.g. the faces of actors as seen in Fig. 5.2. Each target $i \in \{1, \dots, K\}$ is modeled as a sphere at position $\mathbf{p}_t \in \mathbb{R}^3$, with orientation ψ and of diameter h_h. Note that for clarity we typically omit the additional subscript i unless explicitly dealing with several targets. Without loss of generality, in the following we only consider the orientation ψ around the z-axis, to model the gaze direction of the face (see Fig. 5.2). For collision avoidance and visibility, we consider that each target (not only the face) occupies a physical space defined by an ellipsoid S_{Ω_s} centered at $\mathbf{p}_h = \mathbf{p}_t + (\frac{h_h}{2} - \frac{h}{2})\mathbf{e_z}$, where h denotes the physical height of the target from the ground and $\mathbf{e_z}$ the earth's up vector, illustrated in Fig. 5.2 for the case of a human actor.

5.4.1. Method Overview

We propose a receding horizon MPC formulation for trajectory generation given by the following variables, cost terms and constraints. Denote by Δ_t the time step and by N the time horizon of the MPC problem. At each time step a local trajectory of duration $N\Delta_t$ is computed. The first input is then applied. The optimization problem is re-solved at every sampling instance, leading to closed loop performance. This makes the approach robust against model uncertainties and unpredictable disturbances. At the next timestep $k+1$ we use the estimated states $\hat{\mathbf{x}}_{\mathrm{q}_0}$ of the quadrotor and all targets as the initial state $\mathbf{x}_{\mathrm{q}_0}$ and use the trajectory from timestep k as an initialization for the solver.

Variables

The optimization variables are the following. (a) The states $\mathbf{x}_{\mathrm{q}_{0:N}}$, which include the initial state $\mathbf{x}_{\mathrm{q}_0}$, the state trajectory $\{\mathbf{x}_{\mathrm{q}_1}, \dots, \mathbf{x}_{\mathrm{q}_{N-1}}\}$ and the final state $\mathbf{x}_{\mathrm{q}_N}$. The control inputs (b) $\mathbf{u}_{\mathrm{q}_{0:N-1}} = \{\mathbf{u}_{\mathrm{q}_1}, \dots, \mathbf{u}_{\mathrm{q}_{N-1}}\}$. And (c) the additional slack variables $\lambda_{\mathrm{q}_{0:N}} = \{\lambda_{\mathrm{q}_0}, \dots, \lambda_{\mathrm{q}_N}\}$, one for each collision constraint and described in the forthcoming Sec. 5.4.4.

The set of optimization variables is denoted by

$$\mathbf{z}_{\mathrm{q}} = \{\mathbf{x}_{\mathrm{q}_{0:N}}, \mathbf{u}_{\mathrm{q}_{0:N-1}}, \lambda_{\mathrm{q}_{0:N}}\}. \tag{5.1}$$

Cost terms

The final quantity we seek to minimize is the weighted sum of cost terms consisting of image framing, visibility and pose costs over all N stages of the planned trajectory.

Constraints

We include constraints to avoid collisions with moving obstacles and moving targets and to respect the dynamic model of the quadrotor introduced in Sec. 2.2.1.

5.4.2. Viewpoint Optimization Problem

At the core of our algorithm lie a number of cost terms $c_{\mathrm{image}}(\mathbf{x}_{\mathrm{q}})$, $c_{\mathrm{angle}}(\mathbf{x}_{\mathrm{q}})$ and $c_{\mathrm{size}}(\mathbf{x}_{\mathrm{q}})$ which describe the deviation from the desired position of each target's projection on the image plane μ_{y_d}, from the viewing angle $\mathbf{a}_d$ and from the projection size σ_d. Each of these cost terms is computed for each state $k \leq N$ of the planning horizon and for each target $i \leq K$. For simplicity of exposition, the derivation of the costs is described for a general target. We also define the vector $\mathbf{r}_{ch}$ as the relative vector between the target and the camera as well as the rotation into the camera frame of $\mathbf{r}_{ch}$ which is denoted as $\mathbf{r}^c_{ch}$:

$$\mathbf{r}_{ch} = \mathbf{p}_t - \mathbf{p}_c, \quad \text{and} \quad \mathbf{r}^c_{ch} = R(\bar{\mathbf{q}}_c)\mathbf{r}_{ch},$$

Image space positions

Given the desired position of each target's projection on the image plane μ_{y_d}, the pixel coordinates $\mathbf{m}_{x,y_d}$ are computed via the camera

intrinsics with focal point $C_{x,y}$ and focal length $\mathbf{f} = [f_y, f_x]$:

$$\mathbf{m}_{x,y_d} = \begin{bmatrix} (\mu_{x_d} - C_x)\, f_y \\ (\mu_{y_d} - C_y)\, f_x \end{bmatrix} \quad \text{with } \mu_{(x,y)_d} \in \left\{0, 2C_{(x,y)}\right\}.$$

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

$$c_{\text{image}}(\mathbf{x}_\text{q}) = \|\rho_\mathbf{m}\|_{Q_m} \quad \text{with} \quad \rho_\mathbf{m} = \frac{\mathbf{r}^c_{ch}}{\|\mathbf{r}^c_{ch}\|} - \frac{\mathbf{r}_\mu}{\|\mathbf{r}_\mu\|} \tag{5.2}$$

The size σ of the target in the image plane is computed by projecting the 3D sphere of diameter h_h to a circle in the 2D image plane. The target-size cost $c_{\text{size}}(\mathbf{x}_\text{q})$ is computed using the difference between the projected σ and the desired size σ_d.

$$c_{\text{size}}(\mathbf{x}_\text{q}) = \|\sigma - \sigma_d\|_{Q_\sigma} \quad \text{with} \quad \sigma = \frac{h_h \|\mathbf{f}\|}{\|\mathbf{r}^c_{ch}\|}. \tag{5.3}$$

Relative viewing angle

Given the desired viewing angles θ_d and ψ_d, and the current orientation ψ of the target, we define the desired viewing orientation $\mathbf{a}_d$ between the center of the camera and the target:

$$\mathbf{a}_d = \left[\sin\theta_d \cos\left(\psi_d + \psi\right), \quad \sin\theta_d \sin\left(\psi_d + \psi\right), \quad \cos\theta_d\right]^T.$$

Because the angle ψ_d is relative to the target we add the current rotation of the target ψ to obtain the setpoint. The view angle cost $c_{\text{angle}}(\mathbf{x}_\text{q})$ is computed using the error between the desired (relative to the target's normal) and the current viewing angle:

$$c_{\text{angle}}(\mathbf{x}_\text{q}) = \|\rho_\mathbf{a}\|_{Q_a} \quad \text{with} \quad \rho_\mathbf{d} = -\frac{\mathbf{r}^c_{ch}}{\|\mathbf{r}^c_{ch}\|} - \frac{\mathbf{a}_d}{\|\mathbf{a}_d\|}. \tag{5.4}$$

Camera pose

In addition to local viewpoint optimization we can also consider the residual in the camera's position and orientation dynamics - although this is not a required cost. Here we assume that either the user or a global path planning algorithm can specify a desired pose $\mathbf{p}_{c_d}$ and $\bar{\mathbf{q}}_{c_d}$ for the flying camera, which is compared to the actual pose of the camera given by $\mathbf{p}_c$ and $\bar{\mathbf{q}}_c$. The position residual is given by the euclidean distance. The orientation residual ρ_θ is computed building the error quaternion and projecting it into the tangent space $so(3)$ of the $SO(3)$ manifold:

$$\rho_\theta = \begin{bmatrix} \bar{\mathbf{q}}_{e_x} & \bar{\mathbf{q}}_{e_y} & \bar{\mathbf{q}}_{e_z} \end{bmatrix} \in so(3) \quad \text{with} \quad \bar{\mathbf{q}}_e = \bar{\mathbf{q}}_c \otimes \bar{\mathbf{q}}_{c_d}^{-1}.$$

$$c_{\text{pose}} = \|\rho_{\mathbf{p}}\|_{q_p} + \|\rho_\theta\|_{q_\theta}. \tag{5.5}$$

$$c_{\text{pose}}(\mathbf{x}_{\text{q}}) = \|\mathbf{p}_{c_d} - \mathbf{p}_c\|_{Q_p} + \|\rho_\theta\|_{Q_\theta}. \tag{5.6}$$

5.4.3. Occlusion Minimization

To allow for scenes containing objects that can move into the line-of-sight, we have to account for target visibility. Dynamic objects are modelled as ellipsoids S_{Ω_s} and we observe that point visibility can be decided by 'horizon culling', illustrated in In Fig. 5.4. The lines from the camera center $\mathbf{p}_c$ to the ellipsoid's tangent define the horizon plane H_o. All points on the intersection of H_o and S_{Ω_s} are on the horizon, all points in the shaded region C_o are below the horizon and therefore not visible from the camera's viewpoint. To this end we adopt a fast visibility test [Cozzi and Stoner, 2010], which can be summarized as: (i) determine if a point is in front H_o (visible) and (ii) for those behind H_o whether they are within the infinite cone formed by the tangent lines (C_o). To determine if a target $\mathbf{t}_i$ is in front of H_o, defined by $\mathbf{t}_j$,

we compute:

$$\begin{aligned} \mathbf{r}_{ch_i} &= \mathbf{p}_{t_i} - \mathbf{p}_c, \\ \mathbf{r}_{ct_j} &= \mathbf{p}_{t_j} - \mathbf{p}_c, \\ p_{\text{proj}} &= \mathbf{r}_{ch_i}^T \mathbf{r}_{ct_j}. \end{aligned}$$

Where $\mathbf{r}_{ch_i}$ is the vector to the target, $\mathbf{r}_{ct_j}$ the vector to the center of ellipsoid S_{Ω_s} and p_{proj} is the component of $\mathbf{r}_{ch_i}$ in the direction of $\mathbf{r}_{ct_j}$. If $p_{\text{proj}} > \mathbf{r}_{ct_j}^T \mathbf{r}_{ct_j} - 1$ then the point is behind the plane.For these cases we can determine if the point falls into the cone by comparing the opening angles α and β, see Fig. 5.4. Avoiding costly trigonometric functions we use directly the approach given in [Cozzi and Stoner, 2010]. This gives us the visibility score:

$$d_v = \frac{p_{\text{proj}}}{\mathbf{r}_{ch_i}^T \mathbf{r}_{ch_i}} > \mathbf{r}_{ct_j}^T \mathbf{r}_{ct_j}.$$

And we define the visibility cost $c_{\text{vis}}(\mathbf{x}_{\text{q}})$:

$$c_{\text{vis}}(\mathbf{x}_{\text{q}}) = \begin{cases} \|d_v\|_{Q_v} & \text{if } d_v > 0 \text{ and } p_{\text{proj}} > \mathbf{r}_{ct_j}^T \mathbf{r}_{ct_j} - 1, \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

5.4.4. Collision Avoidance

A final concern of the planning algorithm is to generate collision-free trajectories. Here we assume known geometry of the scene and real-time position information of moving obstacles (targets are also considered as moving obstacles). To improve safety and at the same time guarantee performance and responsiveness of the system we adopt a two layered approach: (i) a potential field to repel the robot from obstacles and (ii) a hard constraint to stay outside a smaller enclosing ellipsoid S_{Ω_s} to enforce collision-free motion. We denote the relative vector between the center of the target and the camera by $\mathbf{r}_{ch} = \mathbf{p}_t - \mathbf{p}_c$.

Collision avoidance potential field

First, we employ a potential field cost which becomes active as soon as the camera enters an ellipsoid S_{Ω_l} containing the target and with a buffer zone. This term helps to maintain a safe distance from moving obstacles and leaves some buffer for the un-modeled dynamics of the quadrotor and human motion. Formally, the distance to this ellipsoid is given by

$$d_c = \mathbf{r}_{ch}^T \Omega_l \mathbf{r}_{ch} - 1.$$

Which produces the cost term:

$$c_{\text{coll}}(\mathbf{x}_{\text{q}}) = \begin{cases} \|d_c\|_{Q_c} & \text{if } d_c > 0 \\ 0 & \text{else} \end{cases}. \tag{5.8}$$

Collision avoidance constraint

Second, we introduce a non-linear constraint which becomes active within the collision ellipsoid S_{Ω_s}. This constraint guarantees collision-free motion, i.e. the position of the quadrotor must remain outside of the S_{Ω_s} at all times. Formally,

$$\mathbf{r}_{ch}^T \Omega_s \mathbf{r}_{ch} > 1 - \lambda_c. \tag{5.9}$$

We introduce the slack variables λ_c and, therefore, we adopt a soft-constrained approach for the collision-avoidance constraints to ensure that the optimizer always returns an answer in practice. It can be shown that under sufficiently high penalization of (a linear norm of) the slack variables λ_c, the solution of the hard-constrained problem is recovered when it exists; otherwise, a plan with minimum deviation will be computed by the optimizer [Kerrigan and Maciejowski, 2000].

5.4.5. MPC Formulation

The locally optimal trajectory and inputs for the quadrotor are then computed by solving a constrained optimization problem, which con-

sists of the cost terms introduced in Eq. (7.10)-(5.8) and the constraints of Eq. (6.3), stacked together over all targets, obstacles and time-steps of the controller. Without loss of generality, we consider the obstacle set to be equal to the set of targets, since an obstacle can be treated as a target with only the collision avoidance constrained active - and no other cost term. The full constrained optimization problem solved as MPC non-linear program is then given by:

$$
\begin{aligned}
\min_{\mathbf{x}_\mathrm{q},\mathbf{u}_\mathrm{q},\lambda_\mathrm{q}} \quad & w_N^T c(\mathbf{x}_{\mathrm{q}_N},\mathbf{u}_{\mathrm{q}_N}) + \Sigma_{k=0}^{N-1} w^T c(\mathbf{x}_{\mathrm{q}_k},\mathbf{u}_{\mathrm{q}_k}) + \lambda \|\lambda_k\|_\infty \\
& \mathbf{x}_{\mathrm{q}_0} = \hat{\mathbf{x}}_{\mathrm{q}_0} && \text{(Initial State)} \\
\text{s.t.} \quad & \mathbf{x}_{\mathrm{q}_{k+1}} = f(\mathbf{x}_{\mathrm{q}_k},\mathbf{u}_{\mathrm{q}_k}), && \text{(Dynamics)} \\
& \mathbf{r}_{ch}^T \Omega_s \mathbf{r}_{ch} > 1 - \lambda_k, && \text{(Collision Avoidance)} \\
& \mathbf{r}_{ch} = g(\mathbf{x}_{\mathrm{q}_k}), \\
& \mathbf{x}_{\mathrm{q}_k} \in \mathcal{X}, && \text{(State Constraints)} \\
& \mathbf{u}_{\mathrm{q}_k} \in \mathcal{U}, && \text{(Input Constraints)} \\
& \lambda_k \geq 0, && \text{(Slack Constraints).}
\end{aligned}
\tag{5.10}
$$

where the stage cost function $c(\mathbf{x}_{\mathrm{q}_k},\mathbf{u}_{\mathrm{q}_k})$ is defined as:

$$
c(\mathbf{x}_{\mathrm{q}_k},\mathbf{u}_{\mathrm{q}_k}) = \begin{bmatrix} c_\mathrm{image} & c_\mathrm{size} & c_\mathrm{angle} & c_\mathrm{coll} & c_\mathrm{vis} & c_\mathrm{pose} \end{bmatrix}^T
$$

Where w are weights that can be set interactively by the user to control influence of the different framing constraints. This problem can readily be formulated in standard software, e.g. FORCES Pro [Domahidi et al., 2012; Domahidi and Jerez, 2016], and efficient code be generated to solve it in real-time.

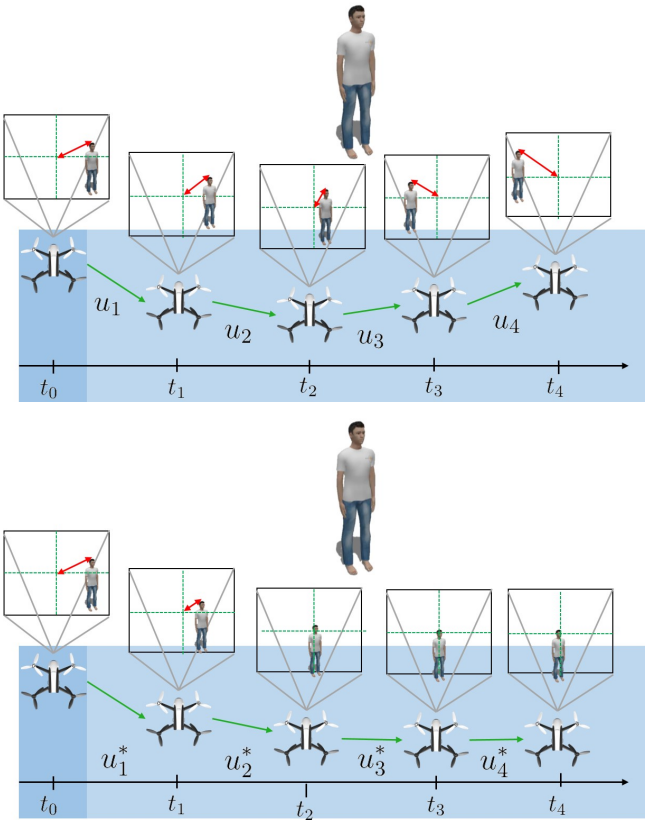


Figure 5.3.: *Top: Schematic explanation of the MPC framework. The MPC algorithm starts at t_0 and sums all defined costs (red) in the time window up. Bottom: After a numeric optimization, the optimal inputs u^* are computed and the costs (red) are minimized.*

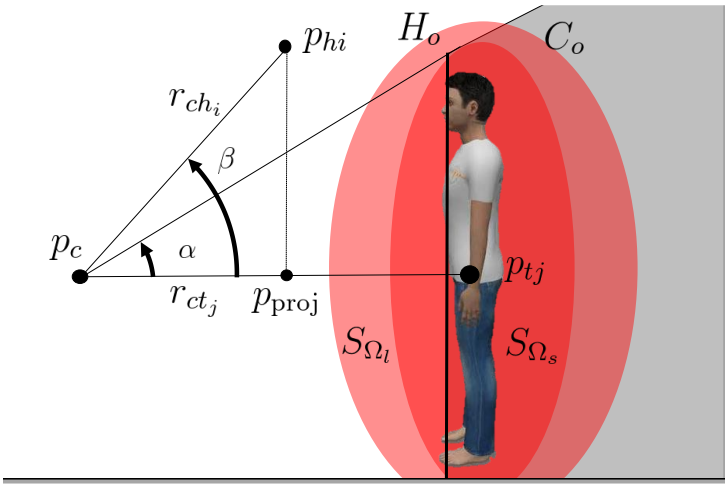


Figure 5.4.: *Schematic illustration of occlusion minimization based on viewpoint dependent horizon culling.*

Figure 5.5.: *Exp. 1: The effect of viewpoint optimization under varying set-points: 3/4 Frontal right, 3/4 frontal left, right screen position, left screen position (from left to right).*

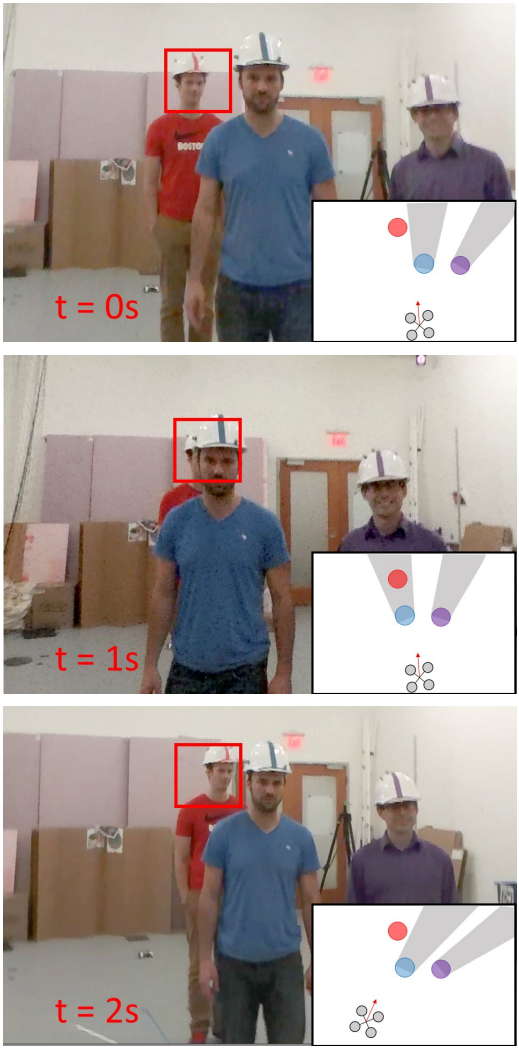


Figure 5.6.: *Exp. 2: Effect of occlusion handling (onboard view). Entire duration is 3s. Inset shows robot position and non-visible areas.*

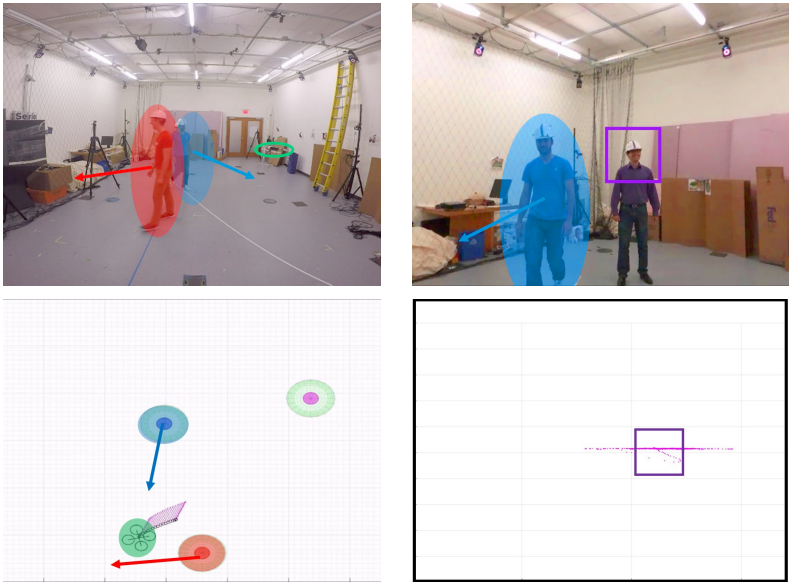


Figure 5.7.: *Exp. 3: Framing and collision avoidance. From left to right: Offboard view (robot in green). Onboard view (target in purple). Visualization of collision ellipsoids. Target position residual, gimbal reduces error in x but yawing the robot is slow (large error in y).*

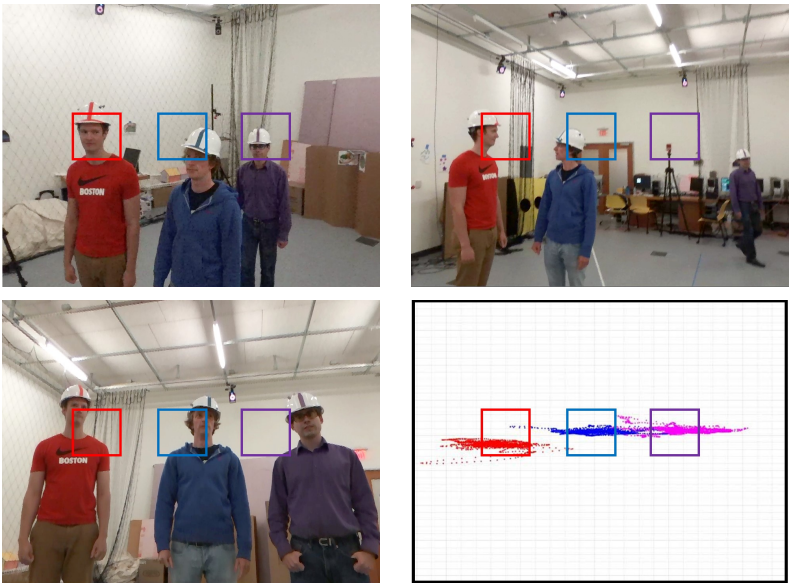


Figure 5.8.: *Exp. 4: The Camera tries to keep the three targets in view. Distribution of the re-projection is shown (right).*

5.5. Experimental Results

5.5.1. Hardware Setup

We use a Parrot Bebop2 with an integrated electronic gimbal. Robot and targets are tracked with a Vicon motion capturing system. Experiments were conducted on a standard desktop PC (Quadcore Intel i7 CPU@3.5 GHz).

5.5.2. Experiments

We conducted five experiments to evaluate the performance of our proposed method under dynamic conditions. In our experiments we used a maximum number of three moving targets, although this is not an inherent limitation of the method. In the following results the boxes rendered on the images represent the setpoints. The size of a box determines the desired size of the projected target ellipse. The color coding is: Target 1 (red), Target 2 (purple) and Target 3 (blue).

Experiment 1 (setpoint change): First, we illustrate the effect of different framing objectives. A single target is filmed with different setpoints, including transitions from sitting to standing. Horizon length $N = 25$.

Experiment 2 (single target with occlusion): Our second experiment demonstrates effectivity of the framing objectives (Eq. (7.10)-(7.12)) and occlusion handling (Eq. (5.7)). One face is set as target and two further actors move in and out of the line-of-sight. The goal is then to keep the main target always in view while minimizing occlusions from the other two targets. Horizon length $N = 25$.

Experiment 3 (single target with collision avoidance): In the third experiment we modify above setup so that the two other targets walk

randomly, including directly towards the quadrotor. Again the algorithm will try to keep the target in frame while avoiding collisions (Eq. (5.8)). Horizon length $N = 25$.

Experiment 4 (multi target framing objective): In our fourth setup we declare all three actors to be targets and the algorithm tries to keep all three faces appearing along a single line in the image. Horizon length $N = 25$.

Experiment 5 (execution time): To evaluate the real-time performance we conducted two experiments. We measure the execution time per timestep along a full trajectory (solving over the entire horizon per timestep). First, we use a constant number of targets $K = 2$ and vary horizon length ($N = 25, N = 40, N = 55$). Second, we keep the horizon fixed $N = 25$ and vary the number of targets.

5.5.3. Results

Experiment 1: Fig. 5.5 shows the algorithm working with a single target $K = 1$ and a horizon length of $N = 25$. With a single target we change the setpoints to: 3/4 Frontal right, 3/4 frontal left, right screen position, left screen position. The accompanying video also shows the target moving.

Experiment 2: The sequence in Fig. 5.6 illustrates how at time $t = 0s$ the initial condition is met and the face is visible according to the framing objective. At time $t = 1s$ target 3 moves into the line-of-sight and occludes target 2. The robot moves smoothly to the closest pose in terms of framing objectives but restores line-of-sight $t = 3s$.

Experiment 3: Fig. 5.7 shows representative views with the framing objective of creating a frontal shot of target 2. Although the targets 1 and 3 force the robot to perform multiple collision avoidance moves,

the target remains in view and its screen space position remains relatively stable. Due to safety reasons collision avoidance takes the highest priority. Furthermore, collision avoidance (hard constraint) restricts the quadrotor motion stronger than the occlusion minimization (soft constraint).

Experiment 4: Fig. 5.8 shows representative frames from a shot with multi-target framing objective. Although there exists, except of some degenerated cases, a camera position with a zero re-projection error, the algorithm has to respect state constraints. In this case the quadrotor has to stay inside the physical room limits, reducing the range of motion drastically. Nevertheless all targets remain in view and the framing requirements are fulfilled as closely as possible.

Experiment 5: Fig. 5.9 plots the computation time for a trajectory with 2000 timesteps at each of which we solve Eq. (5.10) over the horizon length resulting in a total of $2000 \times N$ calls to the solver. We randomly vary the setpoints, ensuring that we don't initialize with unrealistic values too close to the solution. Furthermore, collision avoidance is turned on. The two experiments are parametrized: (i) First, a constant target number $K = 2$ and variable horizon length Fig. 5.9, top. The mean solve-times are: 0.01s, 0.015s and 0.025s respectively. (ii) Second, a constant horizon $N = 25$ and a variable number of targets K Fig. 5.9, bottom. The mean solve-times are: 0.009s, 0.011s and 0.011s. The computational time grows approximately linear with the length of the horizon. This result is expected according to the design of FORCES Pro. In general we see that the computation time for longer horizons and more targets also tend to have more variability in the solve-time. We think this is due to the existence of a unique solution for a single target, while for multiple targets, depending on the setpoints, the solution may only exist in the least squares sense.

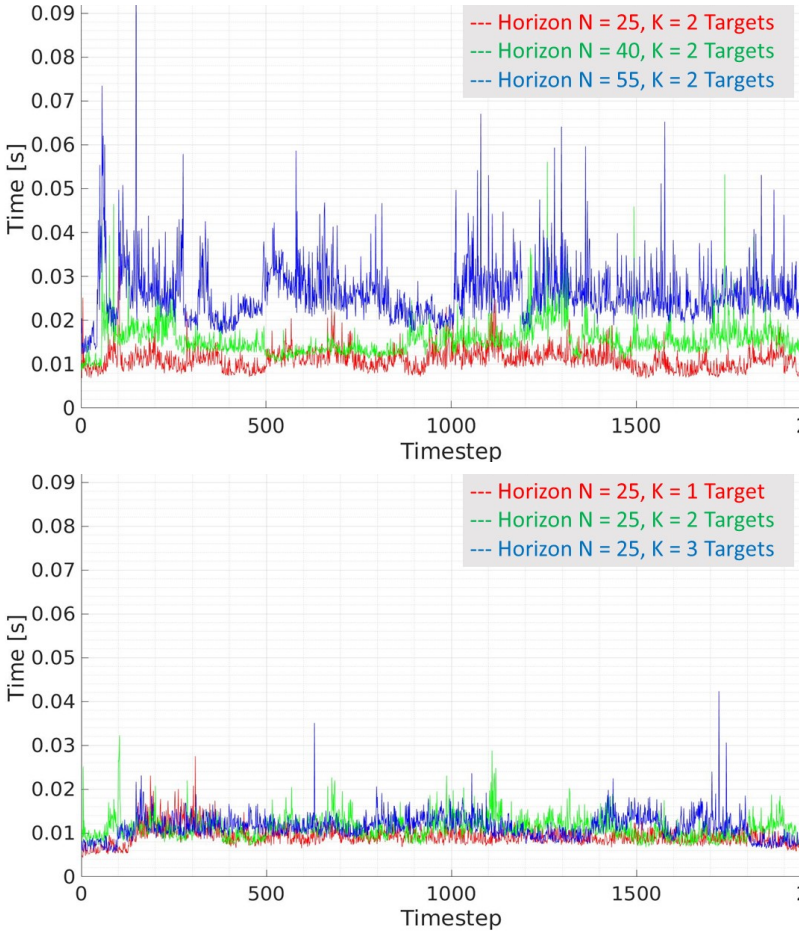


Figure 5.9.: *Plots show solve time per evaluation of the MPC-problem (Eq. (5.10)) as function of horizon length (N) and number of targets (K). Note that we solve over N stages for each timestep (y-axis).* Top*: We change desired setpoints randomly throughout the trajectory but keep number of targets fixed. Colors indicate different horizon length.* Bottom*: Fixed horizon length, while changing number of targets.*

5.6. Conclusion

In this chapter we presented a trajectory generation method that solves for trajectory parameters from set-points defined in image space, and a formulation of this problem that lends itself to an implementation as a receding horizon optimal control program with non-linear constraints which can be solved numerically with state-of-the art solvers in real-time. We have demonstrated in experiments with an aerial vehicle and multiple targets that the algorithm can satisfy framing constraints, derived from cinematographic rules, continuously minimize occlusions from dynamic objects in the environment and avoid collisions with these. Currently our algorithm accepts position and framing setpoints from a user. In future work this could be handled by a global planning algorithm that generates inputs for entire scenes rather than individual shots. We believe that the algorithm is general and could be adapted to other tasks for aerial vehicles which include collision avoidance with respect to moving obstacles.

Chapter 6.

Real-time Planning for Multi-View Drone Cinematography

In this chapter, 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 quadrotor 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.

6.1. 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 quadrotor trajectories [Gebhardt et al., 2016; Joubert et al., 2015, 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 chapter, 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. 3.1, 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 visib-

ility 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.

6.2. 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 way-points 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 pre-determined 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 quadrotor 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. [2015] compute 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. [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 [Joubert et al., 2016; Galvane 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., 2008a], with the majority of methods using discrete optimization formulations. Many of these methods define viewing constraints in screen-space for single subjects [Gleicher and Witkin, 1992; Drucker and Zeltzer, 1994; 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. [Christie et al., 2008b] 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 [Witkin and Kass, 1988; Rose et al., 1996] 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 drone flight [Mellinger and Kumar, 2011; Bry et al., 2015], 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.

6.3. Method

Our real-time method, summarized in Alg. 5, 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 camera-man) 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. 6.2).
- 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.

6.3.1. Dynamical Models

We introduce the models used in our formulation. Let $\mathbf{p}_t \in \mathbb{R}^3$ denote the position of a subject to be filmed and $\dot{\mathbf{p}}_t \in \mathbb{R}^3$ its velocity. The full state of the subject is then denoted by $\mathbf{x}_t = (\mathbf{p}_t, \dot{\mathbf{p}}_t) \in \mathbb{R}^6$, with simple linear dynamics

$$\dot{\mathbf{x}}_t = (\dot{\mathbf{p}}_t, 0),$$

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].

Algorithm 5 Compute drone state

$\mathbf{x}_0 \leftarrow$initialize_horizon(nr. subjects)
loop
  retrieve measurements and predict states: ▷ Sec.6.3.1
  $\mathbf{x}_\mathrm{q} \leftarrow$KalmanFilter($z_\mathrm{quad}$)
  $[\mathbf{p}_t, \dot{\mathbf{p}}_t] \leftarrow$KalmanFilter($z_\mathrm{sub}$)
  retrieve dynamic inputs from user:
  $S_s \leftarrow$framing setpoints from UI ▷ Sec.6.3.2
  $S_c \leftarrow$input trajectories from UI ▷ Sec.6.3.4
  solve for path and quadrotor configuration:
  $s(x_\theta) \leftarrow$compute_virtual_rail($\mathbf{p}_t, \dot{\mathbf{p}}_t, S_s, S_c$) ▷ Sec.6.3.4
  update cost & constraints, solve MPCC Eq. (7.14)
  apply_inputs($\mathbf{u}_{\mathrm{q}_0}$)
end loop

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,

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

where n_x are the dimensions of the states $\mathbf{x}_\mathrm{q} \in \mathbb{R}^{n_x}$ and n_u is the dimensionality of inputs $\mathbf{u}_\mathrm{q} \in \mathbb{R}^{n_u}$ and k denotes the discrete time instant. Typically, the state $\mathbf{x}_\mathrm{q}$ 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 $n_u = 6$.

6.3.2. 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 [Arijon, 1976]. 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 $c_{\mathrm{i}} : \mathbb{R}^{n_x+6} \to \mathbb{R}_+$ on the residual $\rho_{\mathbf{m}}$ between the actual and desired viewing directions of the camera:

$$c_{\mathrm{i}}(\mathbf{x}_{\mathrm{q}}, \mathbf{x}_t) = \|\rho_{\mathbf{m}}\|_{\mathbf{Q}_m} \quad \text{with} \quad \rho_{\mathbf{m}} = \frac{\mathbf{r}_{ch}}{\|\mathbf{r}_{ch}\|} - \frac{\mathbf{r}^c_{chd}}{\|\mathbf{r}^c_{chd}\|}, \tag{6.1}$$

where $\mathbf{r}_{ch}$ is the ray from the camera to the target and $\mathbf{r}^c_{chd} = (\mathbf{m}_{x,y}, 1) \in \mathbb{R}^3$ is the vector through the desired screen position.

The pixel coordinates $\mathbf{m}_{x,y_d}$ are given by the camera intrinsics.

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

$$c_{\mathrm{s}}(\mathbf{p}_t, \mathbf{p}_c, \sigma_d) = \| \, \|\mathbf{p}_t - \mathbf{p}_c\|_2 - \sigma_d \, \|_{\mathbf{Q}_\sigma} \, . \tag{6.2}$$

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.

6.3.3. 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 $S_{\Omega_s}(\mathbf{p}_t)$ around each subject and ask the method to compute quadrotor positions $s()$ that lie *outside* or on its boundary, i.e. the non-convex set

$$\begin{aligned} \bar{S_{\Omega_s}}(\mathbf{p}_t, S_{\Omega_\lambda}) &:= \mathbb{R}^3 \setminus S_{\Omega_s}(\mathbf{p}_t) \\ &:= \{s(\in)\mathbb{R}^3 \mid (s(-)\mathbf{p}_t)^T \Omega_s (s(-)\mathbf{p}_t) \geq 1 - S_{\Omega_\lambda}\}, \quad (6.3) \end{aligned}$$

is the admissible region for camera positions $s()$ for some positive definite matrix $\mathbf{E}$. The scalar $S_{\Omega_\lambda} \geq 0$ 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. $S_{\Omega_\lambda} \equiv 0$, 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.

6.3.4. 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 [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. 6.2. 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.

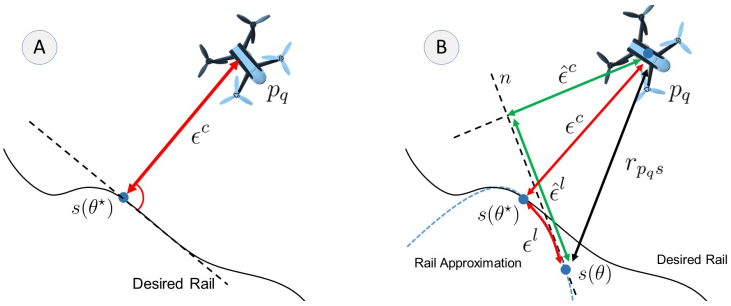


Figure 6.1.: *Exact projection of the quadrotor position onto the virtual rail (A) and it's approximation in the local linearized frame (B).*

Smooth path approximation $s(x_\theta)$ of virtual rails

At each time-step we compute a differentiable path $s(x_\theta)$ 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 $s()$ onto the path:

$$\theta^* = \arg\min_{x_\theta} \|s(x_\theta) - s(\|). \tag{6.4}$$

This projection yields the closest distance to the path which is commonly denoted as the contouring error $\rho_\mathbf{c}$, illustrated Fig. 6.1, 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 2D MPCC it is common practice to approximate $\rho_\mathbf{c}$ via separation of contouring and lag-error. The lag-error $\rho_\mathbf{l} = \int_{\theta^*}^{\theta_k} s(x)dx$ is the integral over the path segment between the desired location on the path θ^* and the location x_θ found by solving the final MPCC problem (Eq. (7.14)).

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 $\rho_{\mathbf{l}}, \rho_{\mathbf{c}}$ by projecting the current quadrotor position $s()$ onto the tangent vector $\mathbf{n}$, with origin at the current path position $s(()x_\theta)$. The relative vector between $s()$ and the tangent point $s()$ can be written as $\mathbf{r}_{\mathbf{p}_q s} := s(x_\theta) - \mathbf{p}_q$. Further the derivative of the path $s(x_\theta)$ with respect to the path parameter x_θ is defined as: $\mathbf{s}' := \frac{\partial s(()x_\theta)}{\partial x_\theta}$ which defines the normalized tangent vector $\mathbf{n} = \frac{\mathbf{s}'}{\|\mathbf{s}'\|}$. The approximation of the lag error is then given by:

$$\hat{\rho}_{\mathbf{l}}(\mathbf{p}_q, x_\theta) = \|\mathbf{r}_{\mathbf{p}_q s}^T \mathbf{n}\|, \tag{6.5a}$$

The approximations of the contour error is computed by:

$$\hat{\rho}_{\mathbf{c}}(\mathbf{p}_q, x_\theta) = \|\mathbf{r}_{\mathbf{p}_q s} - \left(\mathbf{r}_{\mathbf{p}_q s}^T \mathbf{n}\right) \mathbf{n}\|, \tag{6.5b}$$

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 $\dot{x_\theta}$ along the path:

$$c_{\mathrm{p}}(s(,)x_\theta, \dot{x_\theta}) = \begin{bmatrix} \hat{\rho}_{\mathbf{l}}(\mathbf{p}_q, x_\theta) \\ \hat{\rho}_{\mathbf{c}}(\mathbf{p}_q, x_\theta) \end{bmatrix}^T \mathbf{Q} \begin{bmatrix} \hat{\rho}_{\mathbf{l}}(\mathbf{p}_q, x_\theta) \\ \hat{\rho}_{\mathbf{c}}(\mathbf{p}_q, x_\theta) \end{bmatrix} - \beta \dot{x_\theta}, \tag{6.6}$$

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.

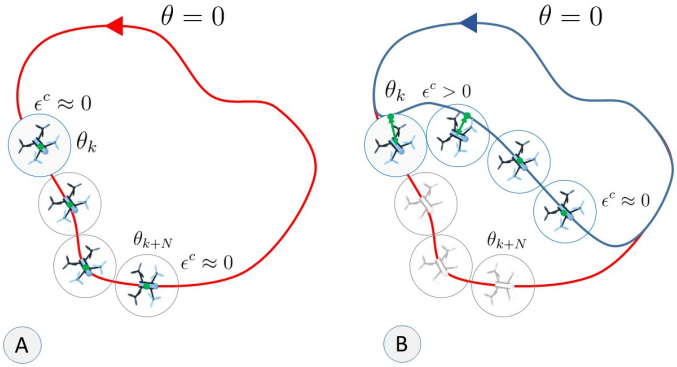


Figure 6.2.: *Online warping of camera reference path. The original reference (red) is changed by the user (blue) at runtime, causing the contour error $\rho_\mathbf{c}$ (in green) to increase for the initial stages of the planning horizon (B). $\rho_\mathbf{c}$ quickly converges to the changed reference path (B).*

Fig. 6.1, B illustrates that as $\hat{\rho}_\mathbf{l}(\mathbf{p}_q, x_\theta)$ becomes small the approximation quality of the contour error increases. In particular when $\hat{\rho}_\mathbf{l}(\mathbf{p}_q, x_\theta) \to 0$ then $\hat{\rho}_\mathbf{c}(\mathbf{p}_q, x_\theta) \approx \rho_\mathbf{c}$. We therefore typically chose a high penalty on $\hat{\rho}_\mathbf{l}(\mathbf{p}_q, x_\theta)$. 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. 6.2, B). The relative weight of the objectives can be set by the user via an appropriate tuning of the cost function Eq. (6.6).

We take a linear combination of the error measures for image location, Eq. (7.10), and size, Eq. (7.11), and the path following cost, Eq. (6.6), to define a stage cost that serves as a performance index for the path generation, following and videography goals of the method:

$$J_k = a_i c_\mathrm{i}(\mathbf{x}_{\mathrm{q}_k}, x_{\theta_k}) + a_d c_\mathrm{s}(\mathbf{p}_{t_i}, \mathbf{p}_{c_k}, \sigma_d) + a_p c_\mathrm{p}(s(x_{\theta_k}), x_{\theta_k}, \dot{x_{\theta_k}}), \quad (6.7)$$

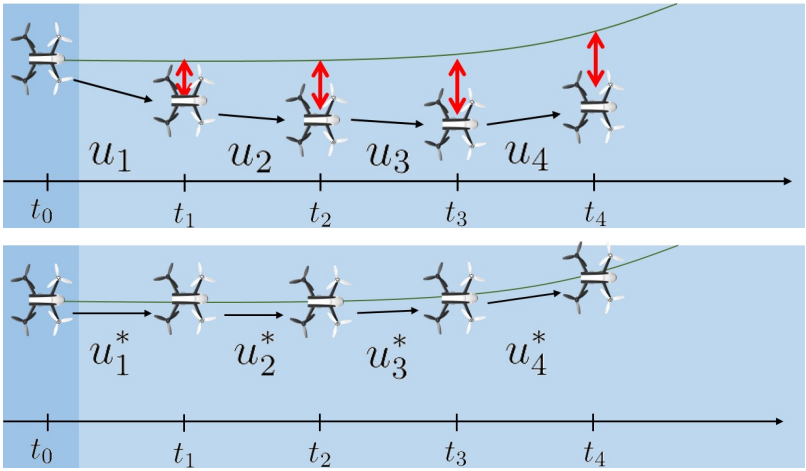


Figure 6.3.: *Top: Schematic explanation of the MPCC framework. The MPCC algorithm starts at t_0 and sums all defined costs (red) in the time window up. Bottom: After a numeric optimization, the optimal inputs u^* are computed and the costs (red) are minimized.*

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 con-

strained non-linear optimization problem at time instant t:

$$\begin{aligned}
\underset{\mathbf{u}_\mathrm{q},\mathbf{x}_\mathrm{q},x_\theta,\dot{x_\theta},\Omega_\lambda}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \left(J_k + \mathbf{u}_{\mathrm{q}_k}^T \mathbf{R} \mathbf{u}_{\mathrm{q}_k}\right) + a_N J_N \quad + \lambda \|\Omega_\lambda\|_\infty && \text{(6.8a)} \\
\text{subject to} \quad & \mathbf{x}_{\mathrm{q}_0} = \hat{\mathbf{x}}_{\mathrm{q}_t} && \text{(Initial state)} \\
& x_{\theta_0} = \hat{\mathbf{x}}_{\theta_t} && \text{(Initial path parameter)} \\
& \mathbf{u}_{\mathrm{q}_{k+1}} = f(\mathbf{u}_{\mathrm{q}_k}, \mathbf{u}_{\mathrm{q}_k}) && \text{(Dynamics)} \\
& x_{\theta\, k+1} = x_{\theta k} + \dot{x_\theta}_k T_s && \text{(Progress virtual rail)} \\
& s(\theta)k \notin S_{\Omega_s}(\mathbf{p}_{tk}, S_{\Omega_\lambda\, k}) \; \forall s && \text{(Collision avoidance)} \\
& 0 \leq x_{\theta_k} \leq L && \text{(Path length)} \\
& \mathbf{x}_{\mathrm{q}_k} \in \mathcal{X}, && \text{(State constraints)} \\
& \mathbf{u}_{\mathrm{q}_k} \in \mathcal{U}, && \text{(Input constraints)} \\
& S_{\Omega_\lambda\, k} \geq 0, && \text{(Slack positivity)}
\end{aligned}$$

where $\mathbf{R} \in \mathbb{S}_+^{n_u}$ is a positive definite penalty matrix avoiding excessive use of the control inputs. The vector $\mathbf{x}_{\mathrm{q}_t}$ and the scalar $\hat{x_\theta}(t)$ denote the (estimated or measured) values of the current states $\mathbf{x}_\mathrm{q}$ and x_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 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 $\mathbf{u}_{\mathrm{q}_0}$. Importantly, a new trajectory is recomputed at each time-step, taking updated sensor data, rail configurations and user-specified viewpoints into consideration.

6.4. 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. 6.3.3, 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. 6.4, 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. (6.3) 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. 6.4, B.

$$\bar{S_{\Omega_s}}(\mathbf{p}_{qk}^j, S_{\Omega_\lambda}^j) := \mathbb{R}^3 \setminus S_{\Omega_s}(\mathbf{p}_{qk}^j) \tag{6.9}$$

where scalar $S_{\Omega_\lambda}^j \geq 0$ again are slack variables ensuring that a solution is found in practice.

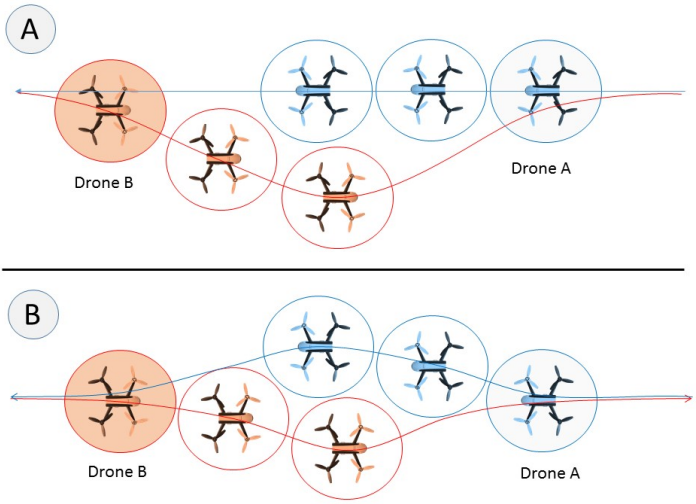


Figure 6.4.: *Schematic of collision between two quadrotors. (A) shows classical priority based collision avoidance where the blue quadrotor has higher priority than the red. (B) shows our solution with iterative sharing of planned trajectories. This results in cooperative collision avoidance.*

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. (7.13) with an additional term c_{v} for each pair of drones, penalizing mutual visibility. The computation of this cost is schematically illustrated in Fig. 6.5.

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

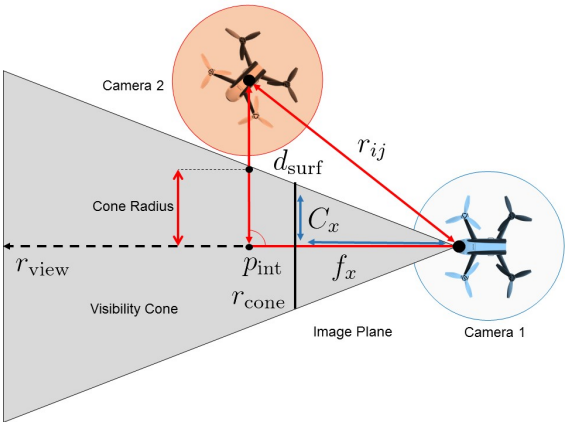


Figure 6.5.: *Explanation of the mutual visibility cost Eq.* (6.10).

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

$$c_{\text{v}}(\mathbf{x}_{\text{q}_i}, \mathbf{x}_{\text{q}_j}) = \begin{cases} Q_v d_{\text{surf}}^2 & \text{if } d_{\text{surf}} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{6.10}$$

where $\mathbf{x}_{\text{q}_i}$ is the state of drone i. The cost c_{v} is added to Eq. (7.13) with a tunable weight Q_v. The resulting behavior is shown in Fig. 6.6. Algorithm 6 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. (7.14). For each neighboring drone, we add: a) the collision constraints of Eq. (6.9), and b) the mutual visibility cost of Eq. (6.10). Note that *communicating* motion plans, rather than relying on *estimates*, can enable highly dynamic maneuvers.

Algorithm 6 Multi-drone algorithm

Require: Trajectories $\mathcal{T}_i$ for all drones $i \in \{1, M\}$ at time $t-1$.
Require: Inputs $[S_s, S_c, \mathbf{p}_t, \dot{\mathbf{p}}_t]$ 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. (7.14) with new constraints and costs.
end for

6.5. Evaluation and Discussion

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

6.5.1. 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 et al., 2012; Domahidi and Jerez, 2016] 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 $\mathbf{u}_{\mathrm{q}_0}$ computed in Eq. (7.14), *without* an additional feedback controller for trajectory tracking on the drone.

Initialization

The problem of Eq. (7.14) 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.

6.5.2. Quantitative and Qualitative Experiments

Computational performance

To assess the computational performance of the method we record overall solve time of the method (Algorithm 5 and 6). 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. 6.7 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. 6.6 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.

6.5.3. 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 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. 6.8 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.

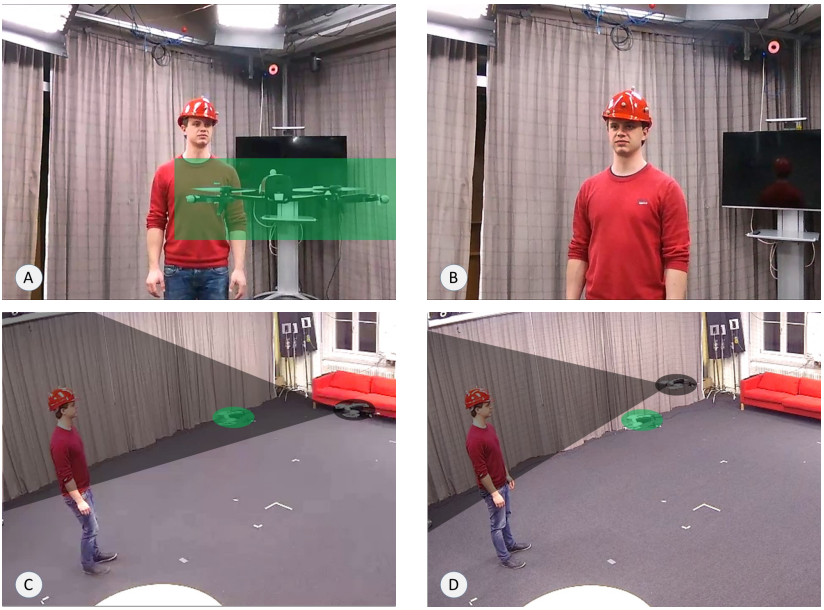


Figure 6.6.: *Influence of penalizing mutual visibility. A and C: single subject is filmed by two drones simultaneously, the first drone (green) is in the field of view of the second drone. B and D: Enabling the mutual visibility cost triggers re-planning of trajectory resulting in unobstructed view with correctly framed subject.*

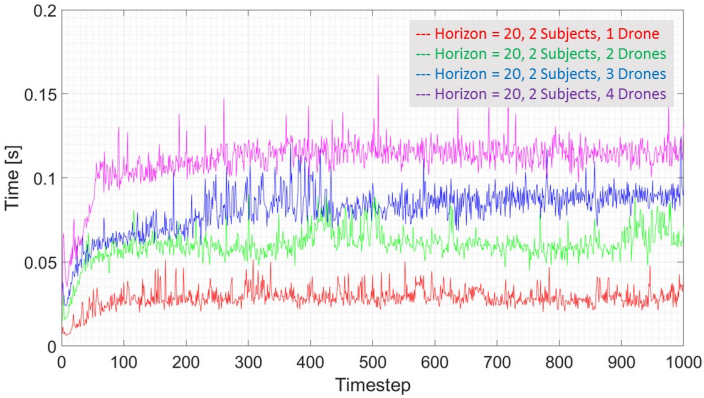


Figure 6.7.: *Solve times for horizon length $N = 20$ and 2 subjects with number of drones varying from 1 - 4.*

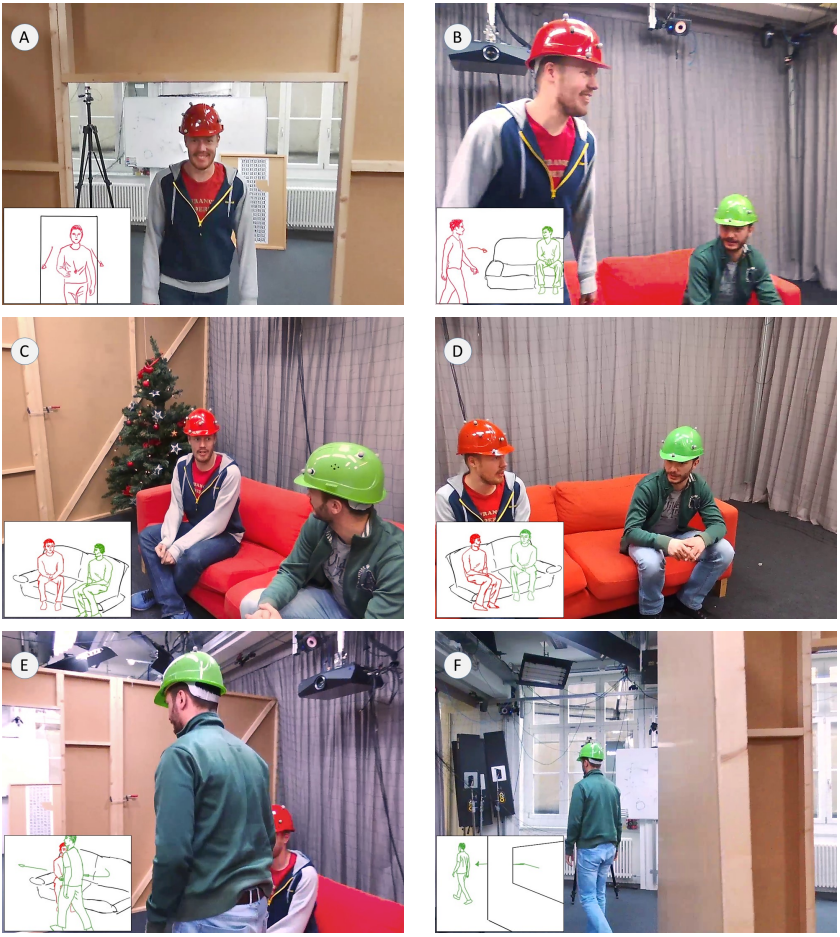


Figure 6.8.: *Figures A - F: Multi-view, multi-person shot, transcribed form story board (insets). Two drones enter and leave the buildin to frame the subjects. The user interactively refocuses the subjects during filming. Full sequence as real-time cut in the accompanying video.*

Figure 6.9.: *Figures A - D: Multi-view, single-person shot outdoors. A+B: A person rides a bike along a street and is filmed by camera 1 (red). C+D: Person gets of the bike, switch to camera 2 (purple). camera 1 goes out of view of camera 2.*

6.6. Conclusions

In this chapter we present 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.

Part III.

Real-time Drone-based Motion Capturing

Chapter 7.

Environment-independent Human Pose Estimation with Drones

In the last technical chapter of this thesis, we present a real-time method for the infrastructure-free estimation of articulated human motion, as for example needed in animation movies. The approach leverages a swarm of camera-equipped flying robots and *jointly* optimizes the swarm's and skeletal states, which include the 3D joint positions and a set of bones. Our method allows to track the motion of human subjects, for example an athlete, over long time horizons and long distances, in challenging settings and at large scale, where fixed infrastructure approaches are not applicable. The proposed algorithm uses active infra-red markers, runs in real-time and accurately estimates robot and human pose parameters online without the need for accurately calibrated or stationary mounted cameras. Our method i) estimates a global coordinate frame for the quadrotor swarm, ii) jointly optimizes the human pose and relative camera positions, and iii) estimates the length of the human bones. The entire swarm is then

controlled via a model predictive controller to maximize visibility of the subject from multiple viewpoints even under fast motion such as jumping or jogging. We demonstrate our method in a number of difficult scenarios including capture of long locomotion sequences at the scale of a triplex gym, in non-planar terrain and while climbing.

7.1. Introduction

Many graphics applications such as character animation for games, sports, biomechanics, VR, and AR rely on accurate human pose information, and virtually every modern movie production leverages Motion Capture (Mocap) systems for special effects. Most commonly, such systems are camera based, either relying on body-worn markers, or more recently even work markerless. Multi-view approaches can now be highly accurate and sometimes provide dense surface reconstructions. The maturing of camera based motion capture technology in turn leads to a desire to use it in increasingly challenging scenarios such as with fast moving actors, large scale scenes and even in outdoors settings. However, practically all existing approaches require a set of environment mounted, accurately calibrated cameras looking into a capture area of fixed size. This requirement for stationary cameras makes application in these settings very tedious, costly and sometimes entirely infeasible.

In this chapter we propose an environment-independent approach to multi-view human motion capture that leverages an autonomous swarm of quadrotors. They carry cameras trained on the subject of interest, who wears a sparse set of active LED markers. The 2D positions of these markers are extracted from the images and the 3D joint positions of the human skeleton are estimated in real-time.

Our approach we address several challenges: First, and in contrast to traditional camera localization approaches that make rigid scene assumptions, the 3D joint locations move in an articulated non-rigid fashion. Second, the cameras move relative to the human and their configuration changes dynamically, which is in contrast to typical hu-

man pose estimation approaches where the cameras are assumed to be stationary and calibrated. Third, we do not rely on any external signal, such as GPS for positioning, making our approach applicable both indoors and outdoors. Our method enables motion capture in previously difficult or entirely infeasible scenarios such as continuously reconstructing the full body pose of an athlete throughout an entire workout or capturing actors in remote and difficult to reach locations, for example while climbing.

More concretely we propose a completely self-contained method for the *joint* estimation and control of the states of multiple quadrotors and of 3D human skeletal configuration. The proposed algorithm runs in real-time and accurately estimates the positions of the robot swarm and the human pose parameters. Furthermore, we compute in real-time drone trajectories to keep the cameras trained on the subject and therefore the markers in view of the cameras.

Our algorithm is inspired by recursive filtering techniques used in robot localization problems. However, in contrast to classical scene reconstruction and camera localization algorithms, the tracked 3D points are not static but move in a complex, articulated fashion. To make this nonlinear state estimation problem of a discrete-time stochastic system tractable in real-time, we pose it as an indirect iterated extended Kalman filter (IEKF) which computes the state estimates as maximum a posteriori (MAP) estimates. In typical camera localization formulations, states are estimated relative to a global world reference frame, which causes the uncertainty with respect to the origin to grow as one moves further away [Castellanos et al., 2004]. To avoid this uncertainty growth, we use a formulation where 3D points and the world origin are expressed with respect to a *moving* reference frame (the lead drone of the swarm). During state propagation and update, linearization is performed around the estimated lead camera frame. In consequence, little linearization error is accumulated over time. This allows us to follow the subject over long distances without drift or loss in pose estimation accuracy.

To our knowledge, we are the first to frame localization and optimal control of a robotic swarm and the estimation of human articulated

motion as a *joint* optimization problem and to provide a real-time implementation. Our method, at every frame, i) collects images from all drones, detects and labels 2D joint positions, ii) estimates the state of a *leader* robot from onboard sensors (e.g., IMU, down-facing optical flow sensor), iii) estimates the joint positions of the human skeleton (and the bone lengths) and optimizes the relative positions and orientations of the multi-robot swarm; iv) finally, it computes control inputs for the drones via model-predictive control (MPC) to keep markers observable under subject motion.

We demonstrate our method in a number of compelling usage scenarios that include fast motion, such as running or jumping jacks, and that capture long trajectories (hundreds of meters). Furthermore, we demonstrate the benefits of environment independence by following a subject over different elevations and in difficult terrain such as a climbing wall (see Fig. 3.1, middle).

7.2. Related Work

Our work brings together state-of-the-art robotics research on quadrotor state estimation and control and algorithms for motion capture from the computer graphics and vision literature. Here we briefly review the most pertinent work.

Camera-based motion capture: Camera-based capture of articulated human motion is at the core of many graphics and related application domains. Commercial solutions require wearing of marker suits or gloves and depend on multiple calibrated cameras mounted in the environment. To overcome these constraints much research has been devoted to developing marker-less approaches from *multiple cameras* (cf. [Moeslund et al., 2006]). Often such methods trade-in high quality results with offline processing [Bregler and Malik, 1998; Ballan et al., 2012; Starck and Hilton, 2003] but recently real-time approaches [Rhodin et al., 2015; de Aguiar et al., 2008; Stoll et al., 2011; Elhayek et al., 2017] have been proposed. Such approaches typ-

ically fit a skeletal model to image data or represent the human as a collection of Gaussians [Rhodin et al., 2015]. Other approaches to real-time performance include combining discriminative and generative approaches [Elhayek et al., 2017; Oikonomidis et al., 2012]. However, such multi-view approaches always assume stationary, well calibrated cameras and are therefore not suitable in mobile and outdoors scenarios. More recently pose estimation methods have exploited deep convolutional networks (ConvNets) for body-part detection in fully unconstrained *monocular* images [Chen and Yuille, 2014; Newell et al., 2016; Tompson et al., 2014; Toshev and Szegedy, 2014; Wei et al., 2016]. However, these methods only capture 2D skeletal information. Predicting 3D pose directly from 2D RGB images has been demonstrated using offline [Bogo et al., 2016; Tekin et al., 2016; Zhou et al., 2016] methods and in online settings [Mehta et al., 2017]. Monocular *depth* cameras provide additional information and have been shown to aid robust skeletal tracking [Ganapathi et al., 2012; Taylor et al., 2012; Shotton et al., 2013] and enable dense surface reconstruction even under deformation [Zollhöfer et al., 2014; Newcombe et al., 2015; Dou et al., 2016]. Multiple, specialized structured light scanners have been used to capture high-fidelity dense surface reconstructions of humans [Pons-Moll et al., 2015]. Our approach relies on multiple cameras to estimate skeletal motion and we believe much of the above work is complementary to ours in that marker-less techniques could serve as input to our joint camera and human pose estimation pipeline. In contrast to the above work, our method does not require any infrastructure or calibrated cameras. Because the cameras are airborne all measurements are noisy, unreliable and measure only relative quantities, making the task significantly harder. Yet, our method achieves good accuracy over long sequences.

Inertial measurement units: Attaching sensors directly onto the body overcomes the need for line-of-sight and enables use without infrastructure. Inertial measurements units (IMU) are the most prominent type of sensor used for pose estimation. Commercial systems rely

on 17 or more IMUs, which fully constrain the pose space, to attain accurate skeletal reconstructions via inverse kinematics [Roetenberg et al., 2007]. It has been shown that good performance can be achieved with fewer sensors by exploiting data-driven methods [Tautges et al., 2011; Liu et al., 2011; Schwarz et al., 2009] or by taking temporal consistency into account, albeit at the cost of high computational cost and therefore offline processing [von Marcard et al., 2017]. While IMUs provide mobility and accuracy, above approaches inherently require user instrumentation. Furthermore, they rely on sophisticated models of the human and hence can not easily be generalized to other subjects. Our implementation currently also requires body-worn markers but in principle can work markerless. More importantly, we optimize 3D point coordinates and only model the human by connecting adjacent joints, thus reducing computational cost of the optimization and making the method applicable to all kinds of articulated motion.

Drones in graphics and vision: With the consumerization of aerial robots, the graphics community has recently proposed a number of tools and algorithms for the planning of physically feasible quadrotor camera trajectories for aerial videography. 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 modeling constraints are considered. The methods of [Joubert et al., 2015] and [Gebhardt et al., 2016] generate quadrotor trajectories given user-defined space-time keyframes, whereas the method proposed in [Roberts and Hanrahan, 2016] takes physically infeasible trajectories and computes the closest possible feasible trajectory by re-timing the velocities subject to a non-linear quadrotor model. All of the above methods are *offline* and cannot generate control inputs for use in *dynamic* environments. Using a Model Predictive Control (MPC) formulation, [Nägeli et al., 2017] optimizes cinematographic constraints, such as visibility and position on the screen, subject to robot constraints for a single quadrotor. [Nägeli et al., 2017] extends

this work to multiple drones and allows actor-driven tracking on a geometric path. The robotics literature has proposed methods to recover the 3D trajectory of a moving person from a quadrotor mounted camera while mapping the environment [Lim and Sinha, 2015; Li et al., 2016]. In contrast, the objective of this work is to reconstruct the full 3D body pose of a moving subject while planning the quadrotor trajectories to always follow the actor and to keep markers in view. For this task, multiple quadrotors are necessary and their position has to be estimated alongside the skeletal joint positions.

In this sense our work is most closely related to [Xu et al., 2017] who leverage depth-cameras mounted on three drones together with a deformable surface energy for dense surface reconstruction of a dynamic user. However, the proposed method relies on depth data, a prescanned template mesh (which is deformed and used for data fitting) and only target tracking is performed in real-time via [Li et al., 2016], whereas pose reconstruction is reported to run at 3 frames per *minute* on a high-end PC. Our method runs entirely in real-time, while it tracks articulated motion and controls the position of the quadrotors. Our method estimates the articulated motion of the user from monocular imagery only and thus can work indoors and outdoors, where depth cameras struggle in direct sunlight.

Multi-robot systems: Multi-robot teams are widely studied in robotics, including groups of aerial [Lupashin et al., 2011; Michael et al., 2010; Lima and Floreano, 2013] robots. To stabilize a formation, each agent requires exact positional knowledge [Pugh and Martinoli, 2006]. Existing approaches to formation flight therefore rely either on low precision sensors, which result in large inter-robot distances, or on external infrastructure. Methods for infrastructure-free formation control have been proposed by [Nägeli et al., 2014], albeit requiring the cameras to be trained on the other members of the swarm, rendering it unsuitable for subject tracking. We do not rely on any infrastructure or external tracking and jointly estimate the drone position and human pose in a single, combined optimization framework.

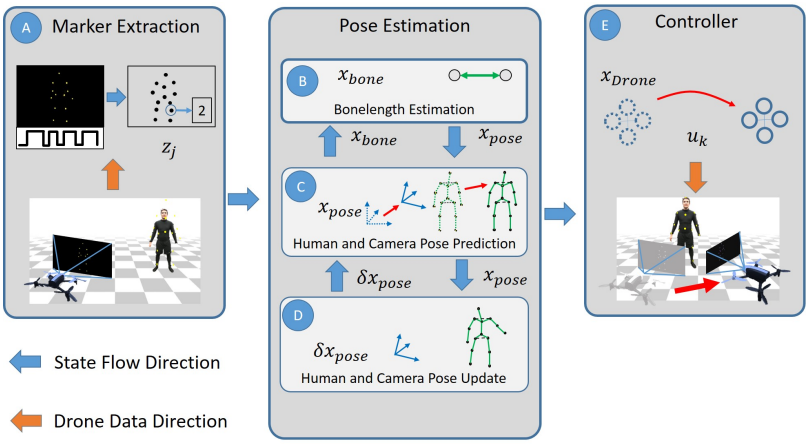


Figure 7.1.: *Method Overview. Left to right: A subject wears a sparse set of active LEDs from which we extract 2D joint detections z_j. A recursive* error-state *filter formulation jointly estimates the position and orientation of multiple flying cameras, and the positions of the 3D joints and the length of bones. Finally we compute feasible trajectories and corresponding control inputs u_k for the quadrotors to keep the human in view.*

7.3. Overview

We propose a drone based real-time method for infrastructure-free estimation of articulated human motion. The approach leverages a swarm of flying camera-equipped robots and *jointly* optimizes the swarm's and skeletal states, consisting of 3D joint positions and a set of bones (see Fig. 7.2) in real-time. This allows to track the motion of human subjects, for example, an athlete over long time horizons and long distances in challenging settings such as outdoors where traditional multi-view approaches are not feasible.

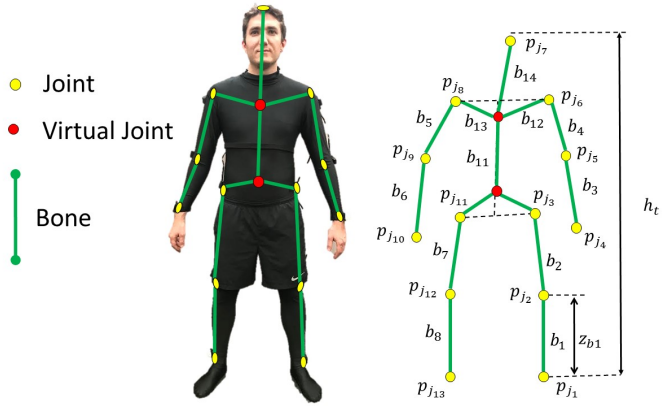


Figure 7.2.: *Schematic of the states used to model the human skeleton* $\mathbf{x}_h$*. The estimated skeleton constists of 13 real joint markers (yellow), two virtual markers (red) and 14 bones (green). The virtual markers are computed using the physikal markers and are introduced for better bone estimates.*

This is a difficult problem because i) in contrast to traditional camera localization approaches, no rigid scene assumption can be made and ii) in contrast to traditional human pose estimation approaches, no rigid camera assumption can be made. In our setting the quadrotor swarm and the 3D joint positions move non-rigidly and relative to each other. While previous work has demonstrated *offline* human pose estimation from several handheld cameras [Hasler et al., 2009] via a conceptually related structure from motion (SfM) approach, this is not feasible here since the swarm is controlled relative to the human and hence real-time performance is mandatory.

To solve this challenging problem of online human pose estimation using quadrotor's in unstructured environments, we make the following key *assumptions*:

1. **Fast sampling:** The camera frame rates and our algorithm are fast ($30Hz$) with respect to human motion. Hence, we assume that the pixel displacement from image to image is small for all marker positions.

2. **Constant bone-length:** Adjacent joints are linked via bones of constant, yet unknown, length. Since the markers are not rigidly attached to the bones, we allow small changes and estimate bone-lengths online, without any prior calibration.

3. **Observability:** Marker's seen from at least two cameras are called *observable*. The location of individual *unobservable* markers can be predicted via the bone-length constraint.

4. **Predictive control:** We assume that trajectories can be generated to accurately track the human and to keep it in the camera's frame (see Sec.7.6.2). This allows for initialization of the pose *a priori* estimate from the drone trajectory.

We formulate this optimization problem in a recursive filtering framework that allows us to naturally link states and measurements over time and provides a straightforward integration of sensor data as priors for each iteration of the optimization.

Additionally, we accurately estimate the states of all quadrotors by fusing the optimized camera poses with odometry measurements attained from onboard sensors, which include downward looking optical flow sensors and IMUs. The drone positions are then controlled to maximize visibility of the subject. Our algorithm, illustrated in Fig. 7.1 iteratively performs the following steps:

(A) Collect images from all drones, detect and label joints.

(B) Predicts and estimates the state of a *leader* robot from onboard sensors (e.g., IMU, down-facing optical flow sensor)

(C) Perform a joint reconstruction of the articulated human pose and the camera states to obtain the position of the joints and the

position of each drone-mounted camera. Update the pose state $\mathbf{x}_{\text{pose}}$. Fuse the camera pose estimate with proprioceptive sensor data (IMU, optical flow) to estimate the full drone state.

(D) Estimate the length of the bones online.

(E) Compute drone inputs, via a receding horizon controller.

7.4. Modeling

We now provide a brief overview of the model of the human multi-robot swarm used in our non-linear estimation and control formulation. The states can be grouped into two sets corresponding to the skeleton (see Sec.7.4.2) and the quadrotor cameras (see Sec.7.4.3).

7.4.1. Terminology

In this work we define the term *Pose* as the pose (joint-angle configuration) of the human, together with the pose (position and orientation) of all cameras. If we talk about a specific pose, we specify this by writing *Camera Pose* or *Human Pose*. We call all quadrotors together a swarm. The swarm together with the human forms a formation.

7.4.2. Human Pose

The pose of the human is defined by a set of m joints, modelled as 3D points, and their connecting bones. Fig. 7.2 shows the assumed mapping between bones and joints. The state of all *joints* is denoted by $\mathbf{x}_{\text{joint}}$ which contains the position of the m individual *joints* that define the human pose: $\mathbf{x}_{\text{joint}} = [\mathbf{x}_{\text{feat}1}, \ldots, \mathbf{x}_{\text{feat}m}] \in \mathbb{R}^{3m}$.

All joints are connected by bones of a certain length. We denote with $\mathbf{x}_{\text{bone}}$ the *bone-lengths* state vector: $\mathbf{x}_{\text{bone}} = [b_1, \ldots, b_{m-1}] \in \mathbb{R}^{(m-1)}$. The bone-lengths are assumed to be constant, but unknown, and therefore treated as *bias states* for which the exact values are

estimated online. The full human (or skeleton) state vector is then given by:

$$\mathbf{x}_t = [\,\text{Joints}\,|\,\text{Bonelengths}\,] = [\mathbf{x}_{\text{joint}}, \mathbf{x}_{\text{bone}}] \in \mathbb{R}^{3m+(m-1)} \quad .$$

7.4.3. Drones and Cameras

We consider n drones, each of them equipped with a camera. The state of each quadrotor is given by its position $\mathbf{p}_q \in \mathbb{R}^3$, its velocity $\dot{\mathbf{p}}_q \in \mathbb{R}^3$and its orientation, i.e. roll Φ_q, pitch Θ_q and yaw Ψ_q. For drone i, its camera is attached to the drone with a gimbal of controllable pitch θ_g and yaw ψ_g. For brevity, we assume the camera position and the quadrotor position to be identical. The full state vector of a drone is defined as:

$$\mathbf{x}_{\text{q}} = [\,\text{Quadrotor}\,|\,\text{Camera}\,] = [\mathbf{p}_q, \dot{\mathbf{p}}_q, \Phi_q, \Theta_q, \Psi_q, \theta_g, \psi_g] \in \mathbb{R}^{11},$$

where the camera state is

$$\mathbf{x}_{\text{cam}} = [\mathbf{p}_q, \theta_g, \psi_g] \in \mathbb{R}^5,$$

and the additional states for the quadrotor are

$$\mathbf{x}_q = [\dot{\mathbf{p}}_q, \Phi_q, \Theta_q, \Psi_q] \in \mathbb{R}^6.$$

Note that the Parrot Bebop's SDK demands angles as input and hence we represent rotations as such for the *control* of the robot and its gimbal. For our *optimization* we always represent rotations as quaternions to avoid gimbal locking. For instance, the 3D camera orientation is denoted by the quaternion $\bar{q}_c = \bar{q}(\theta_g, \psi_g) \in SO(3)$ and the orientation of the drone by $\bar{q}_q = \bar{q}(\Phi_q, \Theta_q, \Psi_q) \in SO(3)$.

7.4.4. State-space Structure and Filtering Strategy

Since all robots and the human move dynamically, solving the problem considered here requires the estimation of the full system state, which consists of the drone states and the human state. This leads to a

very large state-space of $11n + 3m + (m - 1)$. In our implementation this dimensionality is 73. Since the computational cost of a single filter iteration grows cubically with the number of states, a naive implementation would not run in real-time. We leverage two key ideas to reduce the computational cost and render this problem tractable in real-time. (1) We separate the constant, but unknown *bias sates* from the state-space. This technique is known as separate-bias or two-stage estimation [Friedland, 1969; Hsieh, 2000]. See Sec.7.5.4 for details of the online bone length estimation $\mathbf{x}_{\text{bone}}$. (2) We separate the drone states that are not necessary for the human pose estimation, but that have fast dynamics, from the overall state-space. Following [Gibbs, 2011] we refer to these separable states as *control* states. See Sec.7.5 for details on human and drone state estimation.Based on concepts (1) and (2), we can structure the state space into three groups:

- **Pose**: States used for the human pose estimation.
- **Bias**: Bone lengths that are constant but unknown.
- **Control**: Additional states used for quadrotor control.

We now restructure the state space accordingly:

$$\begin{aligned}\mathbf{x} &= [\underbrace{\text{Cameras, Joints}}_{\text{Pose State}} \mid \underbrace{\text{Quadrotors}}_{\text{Control State}} \mid \underbrace{\text{Bonelength}}_{\text{Bias State}}] \\ &= [\underbrace{\mathbf{x}_{c_1}, \ldots, \mathbf{x}_{c_n}, \mathbf{x}_{\text{joint}}}_{\text{Pose State}} \mid \underbrace{\mathbf{x}_{q_1}, \ldots, \mathbf{x}_{q_n}}_{\text{Control State}} \mid \underbrace{\mathbf{x}_{\text{bone}}}_{\text{Bias State}}],\end{aligned}$$

where the Pose state is

$$\mathbf{x}_{\text{pose}} = [\mathbf{x}_{c_1}, \ldots, \mathbf{x}_{c_n}, \mathbf{x}_{\text{joint}}]. \tag{7.1}$$

To solve this problem, we apply an *error state* Kalman filtering (ESKF) strategy to pose state estimation. This allows us to circumvent dynamic modeling errors [Roumeliotis et al., 1999], singularities in the estimation of the covariance matrices of the camera poses [Lefferts et al., 1982b] and filter inconsistencies caused by unobservable states [Castellanos et al., 2004].

Furthermore, the entire constellation of poses is relative to each other and hence, if not taken care of properly, the solution is free to drift arbitrarily. For consistency in the estimation, a global position reference for the human-multi-robot team is required. To address this issue we first estimate the global pose of one drone, which subsequentially is used as reference frame to express all other poses and feature locations. However, even this reference drone has no access to drift-free positional information and hence a recursive filter would incur in growing uncertainty in the pose estimate. To alleviate this issue, we adopt a robo-centric EKF formulation inspired by [Castellanos et al., 2004; de Palézieux et al., 2016]. In our formulation, the world reference frame and feature locations are expressed with respect to a moving reference frame that is updated to the current estimated *leader* pose after every filter update. The (unobservable) uncertainty of the absolute camera position, traditionally associated with the current estimate, is now associated with the world reference pose. Linearization is now performed around the low uncertainty current estimate of the camera pose, avoiding accumulation of error.

The above robo-centric estimation lends itself to a formalization as error-state filter [Castellanos et al., 2004; Roumeliotis et al., 1999]. Since we assume small motion, we can decouple the absolute, yet unknown, pose state $\mathbf{x}_{\text{pose}}$ into an estimated prior state and an additional small error state $\delta_{\mathbf{x}_{\text{pose}}}$:

- **Prior state**: The prior state $\mathbf{x}_{\text{prior}}$ is the *a priori* estimate of the pose $\mathbf{x}_{\text{pose}}$ using all available onboard sensors (imu, optical flow).
- **Error state**: The error state $\delta_{\mathbf{x}_{\text{pose}}}$ describes the residual between the *a priori* and the *a posteriori* estimate of the pose state $\mathbf{x}_{\text{pose}}$ after fusing prior estimates and camera measurements (i.e., marker locations).

We can now write the *a posteriori* estimate of the pose state:

$$\mathbf{x}_{\text{pose}} := \mathbf{x}_{\text{prior}} \otimes \delta_{\mathbf{x}_{\text{pose}}}. \tag{7.2}$$

where $\otimes$ denotes the fusion of the *a priori* total state and the *a posteriori* error state. Linear quantities are updated additively, while rotational entries are updated multiplicatively. Note that $\mathbf{x}_{\text{pose}}$ is the desired quantity we seek to optimize. That is, at the end of the procedure detailed in Alg 7, $\mathbf{x}_{\text{pose}}$ will contain the estimate of the camera swarm and the skeletal configuration.

Algorithm 7 Joint Skeleton and Camera Pose Estimation

loop
  get camera images and label joint positions: ▷ Sec.7.7
  for every camera **do**
    $\mathbf{z}_{\text{Blobs}} \leftarrow$MarkerDetection(Image)
    $\mathbf{z}_j \leftarrow$MarkerLabeling(Blobs)
  end for

  estimate human and drone pose: ▷ Sec.7.5
  $[\mathbf{x}_{\text{pose}}] \leftarrow$JointPoseEstimation$(\mathbf{z}_j, \mathbf{x}_{\text{bone}}, \mathbf{x}_{\text{cam}})$
  $[\mathbf{x}_{\text{bone}}] \leftarrow$BonelengthEstimation$(\mathbf{x}_{\text{joint}})$

  for every drone **do**
    full drone state estimation:
    $[\mathbf{x}_{\text{q}}] \leftarrow$quadStatesEstimation$(\mathbf{z}_{odo}, \mathbf{x}_{\text{cam}})$

    compute drone inputs: ▷ Sec.7.6.2
    update cost & constraints, solve MPC Eq. (7.14)
    apply_inputs$(\mathbf{u}_{\text{q}_0})$ to drone
  end for
end loop

7.5. Joint Camera and Human State Estimation

Given the above filtering structure, recovering the skeletal configuration of the subject alongside the position of the camera drones now boils done to estimating the Pose state $\mathbf{x}_{\text{pose}}$ accurately. This Pose state estimate is then used to compute the control inputs for the swarm for the next timestep (see Sec. Sec.7.6.2) to ensure observability of the human skeleton. We attain this estimate online via recursive estimation repeating a *prediction* and *update* steps alternatingly.

7.5.1. Pose State Propagation

To accurately estimate the humans 3D joint positions we first need to establish where the cameras are relative to the subject - this itself is in the absence of global positioning an underconstraint problem. To initialize our optimization we use the sensors of the drones to get an a-priori estimate $\mathbf{x}_{\text{prior}}$ of the pose state $\mathbf{x}_{\text{pose}}$. We denote by $\mathbf{z}_{\text{odo}i}$ the estimated position of drone i, given by an onboard optical flow estimation algorithm [Bristeau et al., 2011]. Note that at this point neither of the drone has any information about the location of the remaining $n-1$ drones.

To establish the relative transformations, the robot-human constellation requires an absolute position reference. Since the position dynamics of n have only $3(n-1)$ independent degrees of freedom [Nägeli et al., 2014], we require only one absolute position estimate. To approximate this global reference, we pick of one drone, which we refer to as the *leader* drone and use the associated odometry estimate $\mathbf{z}_{\text{odo}1}$ and the resulting position $\mathbf{p}_{q_1}$, as the entire constellations global position estimate. Note that this estimate drifts over time but experimentally we found it to be sufficiently accurate even over long distances and time horizons (see accompanying video).

Camera pose propagation: Following Sec. 7.3, we assume (4) that all drones and the human are *approximately* translating with the global frame, defined by the lead camera. For the lead drone we consider that its position estimate is given by its odometry,

$$\mathbf{p}_{q1}^{k+1} \leftarrow \mathbf{z}_{\text{odo}1} \qquad \text{Drone 1 position propagation.}$$

We can then compute the translation δ of the lead drone in one time step, $\delta := \mathbf{z}_{\text{odo}1} - \mathbf{p}_{q_1} = \mathbf{p}_{q1}^{k+1} - \mathbf{p}_{q_1}$.

From assumption (4), the position estimate of the remaining drones can be initialized by adding the position change δ of the lead drone to the latest position estimate. For drone $i > 1$,

$$\mathbf{p}_{q_i}^{k+1} \leftarrow \mathbf{p}_{q_i} + \delta \qquad \text{Drone i>1 position propagation.}$$

Following the covariance update proposed in [de Palézieux et al., 2016; Castellanos et al., 2004], we marginalize out the position error covariance of the position dynamics for the leader drone, $P_{p1}^{k+1} = \mathbf{0} \in \mathbb{R}^{3\times3}$, and for all the remaining drones, $P_{p_i}^{k+1} = P_{p_i}^{k} + Q_{pos} \in \mathbb{R}^{3\times3}$. The parameter Q_{pos} is a diagonal matrix containing the standard deviation of the expected position change from the initial state, and is a tunable parameter.

Again applying assumption (4), the estimated center of mass of the human is also translated by δ,

$$\mathbf{x}_j^{k+1} \leftarrow \mathbf{x}_j + \delta. \tag{7.3}$$

The covariance of the skeletal joints state is then given by $P_j^{k+1} = P_j^k + Q_j$. The parameter Q_j is a diagonal matrix containing the standard deviation of the expected position change from the initial state, and is a tunable parameter.

7.5.2. Filter measurements

After having established an initialization of the different positions, we use the two following quantities to estimate the δ and hence to update the pose state $\mathbf{x}_{\text{pose}}$:

- **Camera measurements**: Pixel-coordinates of the measured marker positions from each camera, denoted by $\mathbf{z}_j$.
- **Bone-length measurements**: denoted by $\mathbf{x}_{\text{bone}}$, and obtained with the estimated bias state, which we discuss in Section 7.5.4.

Joint residual: At each iteration we receive new camera measurements. In our implementation these are 2D marker positions extracted from the images. With these measurements we perform an *update* step of the filter. More specifically, we use the pixel measurements of all markers seen by all cameras as data term in order to minimize the resulting residual between the *estimated* marker positions and the incoming measurements. The estimate is attained via the prior state $\mathbf{x}_{\text{prior}}$. Without loss of generality, but slight abuse of notation, we describe the measurement residual of a joint a seen by camera i. To build the residual $\rho_{\mathbf{j}}$ between the joint measurement and the *estimated* joint measurement, we project the *estimated* joint position $\mathbf{p}_{\text{joint}} \subset \mathbf{x}_{\text{joint}} \subset \mathbf{x}_{\text{prior}}$ into the camera frame using the prior camera position $\mathbf{p}_q \subset \mathbf{x}_{\text{prior}}$ and orientation $\bar{\mathbf{q}}_q \subset \mathbf{x}_{\text{prior}}$. The projection is performed via a standard pinhole camera model [Hartley and Zisserman, 2003]. The estimated 2D joint position is then attained via a projection into undistorted pixel coordinates,

$$\mathbf{h}_j(\mathbf{x}_{\text{prior}}) = \begin{bmatrix} \mathbf{m}_x f_y + C_x \\ \mathbf{m}_y f_x + C_y \end{bmatrix} \quad \text{with} \quad \mathbf{m} = \frac{1}{\mathbf{r}_z^c} \begin{bmatrix} \mathbf{r}_x^c \\ \mathbf{r}_y^c \end{bmatrix}, \tag{7.4}$$

where $\mathbf{r} = \mathbf{p}_{\text{joint}} - \mathbf{p}_q$ is the relative vector between the joint estimate and the camera center, and $\mathbf{r}^c = R(\bar{\mathbf{q}}_q)\mathbf{r}$ is the vector $\mathbf{r}$ rotated into the camera frame. Pixel coordinates m are computed via the camera intrinsics $f = [f_y, f_x]$ and $C = [C_x, C_y]$.

The residual for joint a seen by camera i is then given by

$$\rho_{\mathbf{j}} = \mathbf{v} - \mathbf{h}_j(\mathbf{x}_{\text{prior}}) \in \mathbb{R}^2, \tag{7.5}$$

where $\mathbf{v} \subset \mathbf{z}_j$ denotes the measurement element for joint a and camera i.

Bone length residual: Conceptually we treat the bone lengths as constant in this filter. However, for modelling convenience we follow [Friedland, 1969] and include them as measurements affected by zero mean Gaussian noise into the filtering framework.

An individual bone-length prediction can be computed as the Euclidean distance between two adjacent 3D joint positions $\mathbf{p}_{j_a}$ and $\mathbf{p}_{j_b}$. It is therefore given by $h_b = \|\mathbf{p}_{j_a} - \mathbf{p}_{j_b}\|$. For a single bone i, the bone-length residual ρ_b is then

$$\rho_b = b_i - h_b(\mathbf{x}_{\text{prior}}) \in \mathbb{R}^1, \tag{7.6}$$

where $b_i \subset \mathbf{x}_{\text{bone}}$ is the constant, but a-priori unknown, bone-length. We estimate this quantity online, see Sec.7.5.4 for details. Intuitively, this residual will ensure that the solution converges to a skeletal configuration in which bones have a constant length. While the physical bone does not change its length at all, this formulation allows for slight variation in relative joint distances. This is due to the difficulties of integrating hard-constraints into recursive filters and due to the fact that the markers and their detections may move relative to the actual joint. Finally, all the individual residuals are stacked into a singe residual vector $\rho = [\rho_{\mathbf{j}}, \rho_b]^T$. This residual vector is then used to update the total state.

7.5.3. Filter Update

To update the total state $\mathbf{x}_{\text{pose}}$, via Eq. (7.2), we first compute the error state δ by performing a Kalman iteration, thus minimizing the residuals ρ. We compute the Kalman gain $\mathbf{K}$ with respect to the measurement models $\mathbf{h}_j(\cdot)$ and $\mathbf{h}_b(\cdot)$, evaluated at the current state estimate. We then compute the Jacobian, denoted by $\mathbf{H}$, of Eq. (7.5) and Eq. (7.6) with respect to the error state δ and linearized around its expected value $\mathbb{E}[\delta] = \mathbf{0}$. Note that this step in practice is highly involved and involves computation of derivatives for the quaternions in $\mathbf{x}_{\text{pose}} \in SO(3)$, with respect to the error state δ.

The *a posteriori* error state is then computed by

$$\delta = \mathbf{K}\rho, \tag{7.7}$$

and the estimated total state is updated such that the expected error state is once again zero $\mathbb{E}[\delta] = \mathbf{0}$. This allows us to rewrite Eq. (7.2):

$$\mathbf{x}_{\text{pose}}^{k+1} = \mathbf{x}_{\text{pose}}^{k} \otimes \delta. \tag{7.8}$$

To make this nonlinear state estimation problem of a discrete-time stochastic system tractable in real-time, we have posed it as an error-state extended Kalman filter (EKF), which computes the state estimates as maximum a posteriori (MAP) estimate. The computed *a posteriori* error state δ is thus only a first order approximation of the true error state. The accuracy of the state estimate can be improved by repeatedly performing an update with a single set of measurements, this is known as an iterated state update (ISEKF) [Gibbs, 2011]. Via re-linearization of the measurement equation around the updated state the IEKF avoids issues with filter convergence, due to accumulated linearization error.

We now have attained an estimate of the joint state of the multi-robot human team including the desired human pose configuration via optimizing $\mathbf{x}_{\text{pose}}$. The covariance matrix is updated with the standard Kalman Filter equation:

$$\mathbf{P}^{k+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\,\mathbf{P}^{k} \tag{7.9}$$

7.5.4. Bone Length Estimation

The bias or bone-length states $\mathbf{x}_{\text{bone}}$ of our filter remain constant over time, but are unknown a-priori. We use an additional linear Kalman filter with a zero order state propagation model [Gibbs, 2011] to estimate the bonelength, given the estimated joint positions $\mathbf{p}_j \in \mathbf{x}_{\text{prior}}$. We only perform the filter update of a bone if both corresponding joints $\mathbf{p}_{ja}$ and $\mathbf{p}_{jb}$ are seen at least by two cameras.

7.6. Camera Control

For accurate human pose estimation we must ensure that the human is always in the field of view of each drone and that each drone in the swarm records the human from a different viewpoint. To achieve this, we build upon the control method of [Nägeli et al., 2017], defining a N-step finite-horizon constrained non-linear optimization problem at time instant k. Note that for here we assume known drone and human states as well as 2D marker positions.

Robot model: To generate correct control inputs, a mathematical model of the drone in form of a non-linear differentiable function $f : \mathbb{R}^{n_x \times n_u} \rightarrow \mathbb{R}^{n_x}$, discretized using a standard forward Euler approach is needed. The discrete-time state update equation of the drone is denoted by

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

where n_x is the dimension of the state $\mathbf{x}_{\mathrm{q}} \in \mathbb{R}^{n_x}$, n_u is the dimension of the input $\mathbf{u}_{\mathrm{q}} \in \mathbb{R}^{n_u}$ and superindex k denotes the discrete time instant. In our experiments we use a Parrot Bebop2 and include dynamics of the (software) gimbal. This results in $n_x = 11$ and $n_u = 6$. With this model in place we define a number of cost terms to constrain the camera motion relative to the user. For given state $\mathbf{x}_{\mathrm{q}_k}$, an input $\mathbf{u}_{\mathrm{q}_k}$ and the predicted position of the human at time-step k, we can compute a cost J_k that penalizes deviations from a desired relative orientation, a desired position and a desired size of the human in the image. This is equivalent to maintaining an approximately constant distance and viewing angle with respect to the human and is defined in image space. A terminal cost J_N, which only depends on the state $\mathbf{x}_{\mathrm{q}_N}$ and the predicted position of the human at time-step N, is computed analogously. The desired values are defined by the user and can vary between drones. For the human prediction we currently employ a constant velocity model.

7.6.1. Marker Visibility

To ensure that each drone can observe as many of the markers as possible we ask each drone to keep the bounding box of the detected and labeled marker positions at a desired 2D position on-screen. Furthermore, we control the relative distance to the human via the size of the projected bounding box and the viewing direction of each drone with respect to the orientation of the human to maximize marker coverage. Via a constant velocity model we predict the human states $\mathbf{x}_t$ into the future. These include the position $\mathbf{p}_t$ of the center of the bounding box and its orientation.

$$c_{\mathrm{i}}(\mathbf{x}_{\mathrm{q}}, \mathbf{x}_t) = \|\rho_{\mathbf{m}}\|_{\mathbf{Q}_m} \quad \text{with} \quad \rho_{\mathbf{m}} = \frac{\mathbf{r}_{ch}}{\|\mathbf{r}_{ch}\|} - \frac{\mathbf{r}^c_{chd}}{\|\mathbf{r}^c_{chd}\|}, \tag{7.10}$$

where $\mathbf{r}_{ch}$ is the ray from the camera to the human and $\mathbf{r}^c_{chd} = \left(\mathbf{m}_{x,y_d}, 1\right) \in \mathbb{R}^3$ is the vector through the desired screen-space position, where pixel coordinates $\mathbf{m}_{x,y_d}$ are computed via the camera intrinsics. The screen-size of the bounding box is controlled via the quadratic error function $c_{\mathrm{s}} : \mathbb{R}^7 \rightarrow \mathbb{R}_+$ on the residual between the actual σ and desired σ_d Euclidean distance between user's position $\mathbf{p}_t$, extracted from $\mathbf{x}_t$, and the camera's $\mathbf{p}_q$:

$$c_{\mathrm{s}}(\mathbf{p}_t, \mathbf{p}_q, \sigma_d) = \| \, \|\mathbf{p}_t - \mathbf{p}_q\|_2 - \sigma_d \, \|_{\mathbf{Q}_\sigma} \, . \tag{7.11}$$

Similarly, the relative viewing angle per drone is controlled via the quadratic error function $c_{\mathrm{a}} : \mathbb{R}^{n_x+6} \rightarrow \mathbb{R}_+$ on the residual $\rho_{\mathbf{a}}$ of the camera relative to the orientation of the human:

$$c_{\mathrm{a}}(\mathbf{x}_{\mathrm{q}}, \mathbf{x}_t) = \|\rho_{\mathbf{a}}\|_{\mathbf{Q}_a} \quad \text{with} \quad \rho_{\mathbf{a}} = \frac{\mathbf{r}_{ch}}{\|\mathbf{r}_{ch}\|} - \frac{\mathbf{a}_d}{\|\mathbf{a}_d\|}, \tag{7.12}$$

where $\mathbf{r}_{ch}$ is the vector from the center of the camera to the human in the global frame, and $\mathbf{a}_d$ is the desired relative viewing orientation, given by

$$\mathbf{a}_d = \begin{bmatrix} \sin\theta_d \cos\left(\psi_d + \psi_h\right), & \sin\theta_d \sin\left(\psi_d + \psi_h\right), & \cos\theta_d \end{bmatrix}^T ,$$

where ψ_h is the current orientation of the human and θ_d and ψ_d are the desired viewing angles, both specified by the user and different for each drone in order to observe the human from different view points.

7.6.2. Trajectory Optimization

For a given drone, and in a slight abuse of notation, we denote by $\mathbf{x}_\mathrm{q} = [\mathbf{x}_{\mathrm{q}_0}, \ldots, \mathbf{x}_{\mathrm{q}_N}]$ and $\mathbf{u}_\mathrm{q} = [\mathbf{u}_{\mathrm{q}_0}, \ldots, \mathbf{u}_{\mathrm{q}_{N-1}}]$ the computed trajectory and inputs, where $\mathbf{u}_{\mathrm{q}_0}$ and $\mathbf{u}_{\mathrm{q}_0}$ are the initial states and inputs for the drone. We take a linear combination of the error measures for image location Eq. (7.10), size Eq. (7.11) and viewing angle Eq. (7.12) to define a stage cost for trajectory optimization:

$$J_k = a_l c_\mathrm{i}(\mathbf{x}_{\mathrm{q}_k}, \mathbf{x}_{t_k}) + a_d c_\mathrm{s}(\mathbf{p}_{qk}, \mathbf{x}_{t_k}\sigma_d) + a_a c_\mathrm{a}(\mathbf{x}_{\mathrm{q}_k}, \mathbf{x}_{t_k}) \quad , \tag{7.13}$$

where the scalar weight parameters $a_l, a_d, a_a > 0$ 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 nonlinear optimization problem at time instant t.

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{u}_\mathrm{q}} \quad & \sum_{k=0}^{N-1} \left(J_k + \mathbf{u}_{\mathrm{q}_k}^T \mathbf{R} \mathbf{u}_\mathrm{q} k \right) + a_N J_N && && (7.14)\\
\text{subject to} \quad & \mathbf{x}^0 = \hat{\mathbf{x}}_{\mathrm{q}_t} && \text{(Initial state)}\\
& \mathbf{x}_{\mathrm{q}_{k+1}} = f(\mathbf{x}_{\mathrm{q}_k}, \mathbf{u}_{\mathrm{q}_k}), && \text{(Dynamics)}\\
& \mathbf{x}_{\mathrm{q}_k} \in \mathcal{X}, && \text{(State constraints)}\\
& \mathbf{u}_{\mathrm{q}_k} \in \mathcal{U}, && \text{(Input constraints)}\\
& && \forall k \in \{0, \ldots, N-1\}\\
& \mathbf{x}_{\mathrm{q}_N} \in \mathcal{X}, && \text{(State constraints)}
\end{aligned}$$

where $\mathbf{R} \in \mathbb{S}_+^{n_u}$ is a positive definite penalty matrix to avoid excessive use of the control inputs. 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 to mimic long horizons, approximating the infinite horizon solution. The vector $\hat{\mathbf{x}}_{\mathrm{q}_t}$ denotes the estimated value of the current state $\mathbf{x}_{\mathrm{q}}$. Finally, the sets $\mathcal{X}$ and $\mathcal{U}$ denote the sets of feasible states and inputs for drone, respectively. 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.

Additional constraints for avoiding collisions between the drones and between each drone and the tracked human could also be added, analogously to [Nägeli et al., 2017].

The drone is actuated using the optimal inputs from the first step $\mathbf{u}_{\mathrm{q}_0}$. Importantly, a new trajectory is recomputed at each time-step, taking updated sensor data into consideration.

7.7. Implementation

Our experiments are conducted on a standard desktop PC (Quadcore Intel i7 CPU@3.5 GHz). The subjects are tracked directly by the drones via a custom active LED marker scheme *no* external motion capture system was used. We implement the recursive estimation algorithm using Matlab.

Quadrotor hardware:

We use Parrot Bebop2 quadrotors in all our experiments with an integrated electronic gimbal the camera has been modified to remove daylight illumination but record IR illumination (cf. Fig. 7.4). 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 $\mathbf{u}_0$ *without* an additional feedback controller

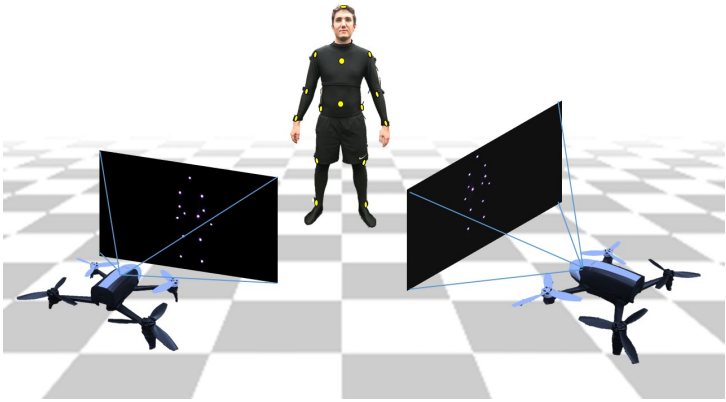


Figure 7.3.: *Our system consists of two drones observing 13 active LED markers worn on the users body. The controller, described in Sec. 7.6.2, computes control inputs fror the drones in order to see as many markers as possible. From the 2D detections observed by the quadrotors, the human pose is estimated in real-time.*

for trajectory tracking on the drone.

7.7.1. Active Markers

Our method takes 2D joint detections as input. While body-part detection in monocular images is possible [Chen and Yuille, 2014; Newell et al., 2016; Tompson et al., 2014; Toshev and Szegedy, 2014; Wei et al., 2016] we leave full integration of such methods for future work. Instead we utilize body-worn markers allowing for simple detection and unique labeling of joints. Our setup consists of 13 active IR-LED markers attached to a morphsuit (see 7.3).

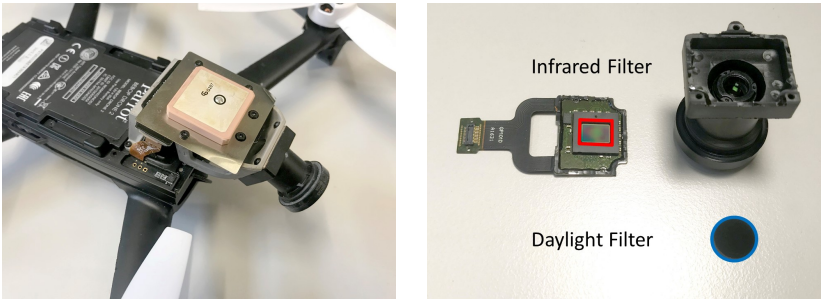


Figure 7.4.: *In order to detect our proposed active marker detection scheme only requires little modification to the Parrot Bebop's hardware and is hence cheap. The camera has to be removed first (left). First, the lens-mount has to be removed using a heat-gun. Then, the infrared filter (read) can be removed and the daylight filter (blue) can be put on the lens mound. In the last step, the camera has to be reasembeld and recalibrated.*

A band-pass filter was added to the camera lenses, removing daylight but letting IR illumination pass. Each marker is composed of two LEDs, one illuminated permanently and the other displaying a unique temporal pattern. Markers are segmented from the background via simple image processing operations. The temporal pattern creates varying image intensities which are converted into a bit-stream which is used to uniquely identify markers and to track them over time (see Appendix 9.1 for details.)

7.8. Experiments

Our method enables motion capture in scenarios that are difficult or entirely infeasible with traditional techniques. Hence, direct quantitative evaluation of accuracy is difficult. Furthermore, the accuracy is

affected by the placement of markers on the body and processing of the resulting images as well as camera calibration. We demonstrate the feasibility and robustness of our proof-of-concept implementation in five experimental evaluations where we continuously reconstruct full body pose of a subject during fast movements, moving through large-scale scenes, in difficult to reach locations, indoors and outdoors.

Fast Motion (Experiment 1): In a first experiment, a participant performs jumping jacks. The sequence in Fig. 7.5 shows one half-cycle of a jumping jack (duration: 0.15 seconds). The maximal joint velocity is limited by the camera sampling rate (30Hz in our experiment). The reconstructed joint positions are indicated in yellow, the skeleton is projected into the image and rendered in green.

Climbing (Experiment 2): To demonstrate the location independence we show results form our system being deployed in a difficult to reach location. A subject climbs up, across, and down a climbing wall, while the drones track the position. Extracts from this sequence can be seen in Fig. 7.10. The drone positions (indicated in red) are optimized to see all markers and hence automatically follow the subject, adjusting their height above ground without external control.

Long trajectory (Experiment 3): Long-range trajectories are a particularly challenging scenario for traditional motion capture approaches. To demonstrate the environment independence of our approach, we ask a participant to walk in large circles in an area that exceeds typical motion capture spaces significantly. Fig. 7.6, top shows different snapshots from the sequence, drones highlighted in red. Fig. 7.6, bottom illustrates the corresponding estimated skeletal configurations. Note that in this experiment the system tracks the user over time period of 3 min and over a trajectory length of 170 meters. We observed an absolute position drift of about 2m, caused by the integrating nature of the optical flow estimates. In Fig. 7.8 we show the cumulated joint

positions (red), relative to the person's center of mass over the long walking sequence.

Ground-truth comparison (Experiment 4): To compute the expected accuracy of our method we performed an experiment with a motion capture system. In particular, we compare the estimated distance between the hand of the subject and one of the drone cameras with that obtained from the vicon based ground truth. In Fig. 7.9 we show the distance from the left hand to the first drone camera. Over a 30 second sequence We obtained a standard deviation of 2.2cm.

Outdoor test (Experiment 5): We assess the environment independence of the proposed approach via an additional outdoor experiment. Fig. 7.7 shows a motion sequence with the live reconstruction of the skeleton (green) in overlay. The drones are highlighted in red for better visibility.

Computational complexity: In principle, the our method could track multiple sets of markers (multiple subjects). However, the complexity of the algorithm grows cubically with the number of measurements and quadratically with the number of states. In our proof of concept implementation the number of states is $(6n + 3m)$ and the number of measurements is $(2m \times n)$. Yet, the current image frame-rate (30Hz) is well below the filter update speed (100Hz), providing enough margin to increase the number of drones or subjects tracked.

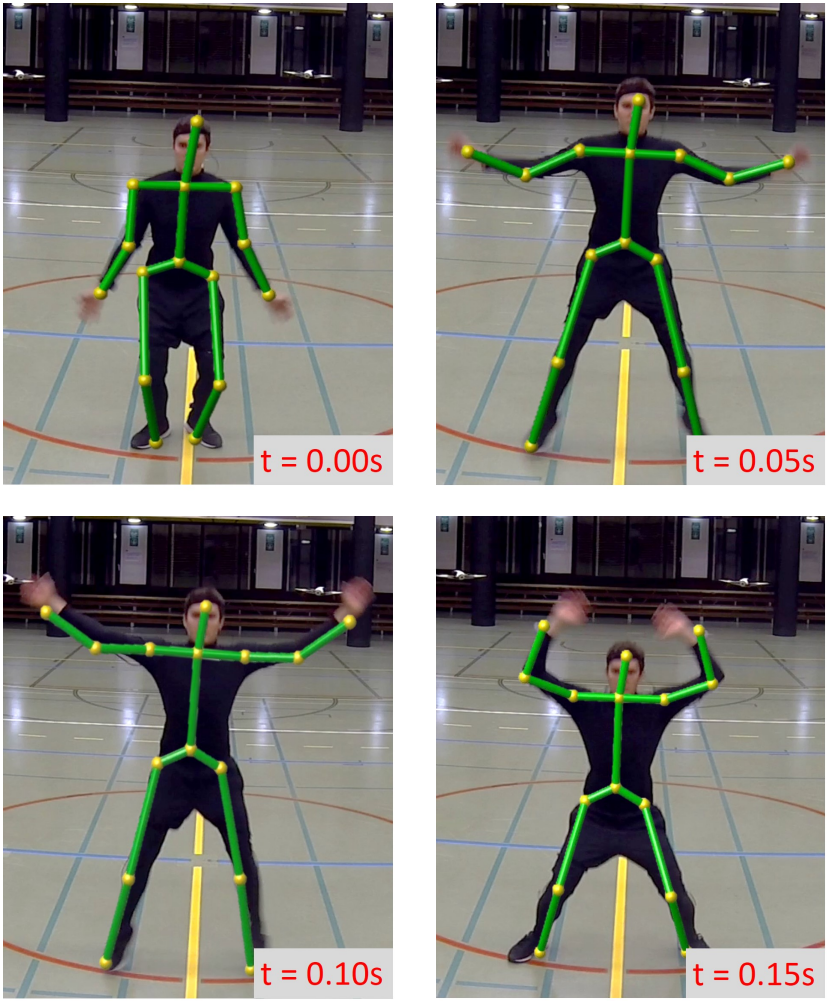


Figure 7.5.: *Experiment 1: A subject performing jumping jacks. The sequence is captured with a camera and the reconstruction of our system is overlayed. The speed limitation of motions we can track is limited by the frame-rate of the Parrot Bebop 2 live image stream and therefore by the marker tracking. The estimated joint positions are indicated in yellow, the estimated skeleton is marked in green.*

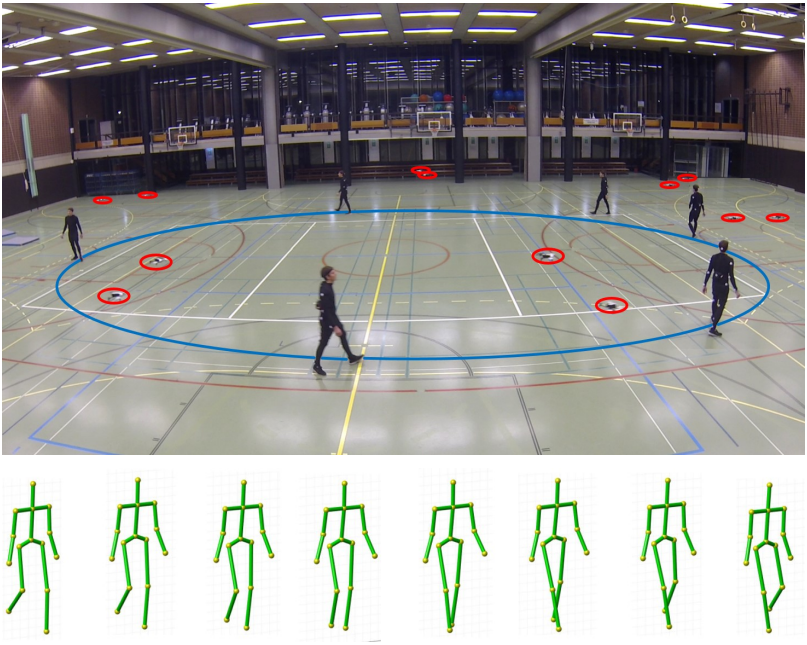


Figure 7.6.: *Experiment 3: Top: Subject walking over a long distance and time period in circles - indicated in blue - with a walking speed around* $1.5\frac{m}{s}$ *(top). The drones follow the subject and always position themselves to optimally observe the markers, mounted on the back of the subject. Bottom: reconstructed gait cycle recorded during the experiment.*

Figure 7.7.: Experiment 5: Our system works indoors and outdoors. Here we show extract from a long walking trajectory in varied terrain.

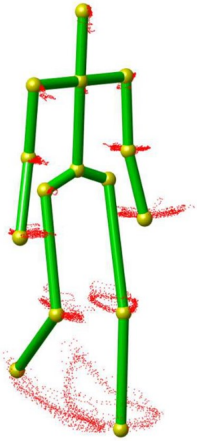


Figure 7.8.: *Experiment 4: The joint positions (yellow) are plotted over time(red) with respect to the skeleton center. The noise distribution is 3cm with respect to the mean joint-trajectory.*

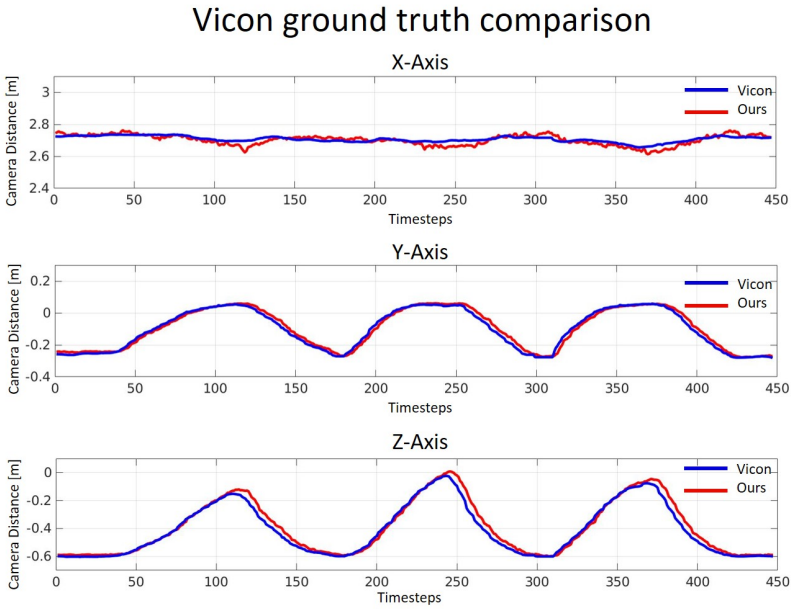


Figure 7.9.: Experiment 4: Ground truth comparison between the hands and the camera while walking. In the plot we show the relative distance (x top, y middle and z bottom) between the left hand and the first camera as a representative result. The ground truth is blue, our estimate is red. The standard deviation is 2.2cm.

Figure 7.10.: *Experiment 2: A subject is climbing up a wall. The drones follow the subject over different elevations and locations. The markers are indicated in yellow, the estimated skeleton in green. The drones are indicated in red.*

7.9. Limitations and Conclusions

In this chapter we presented a real-time method for infrastructure-free estimation of articulated human motion. The approach leverages a swarm of camera-equipped quadrotors and *jointly* optimizes the swarm's and skeletal states, including 3D joint positions and a set of bones, in real-time. The problem is phrased as a non-linear recursive filtering estimation, namely IESKF, allowing us to naturally link state estimates and measurements over time. Furthermore, a robo-centric formulation minimizes accumulation of error due to linearizion around the last state and uncertainty about the global transform. Therefore the method provides robust long term predictions of the global pose of the multi-robot swarm and the human skeletal configuration. We demonstrated the method in a number of challenging settings where traditional multi-view methods are not applicable. Our work lies the foundation for a host of exciting avenues of future work. Foremost we currently rely on active LED-markers to detect 2D joint locations. The framework would naturally admit 2D detections stemming from a deep-learning method that extracts these joint detections from natural images alone (e.g., [Wei et al., 2016; Toshev and Szegedy, 2014]) or are even directly from videos (e.g., [Song et al., 2017]). However, note that our method requires accurate tracking of the human and makes small-motion assumptions, hence integration of a deep-learning approach into our pipeline would have to address several interesting challenges and would require strict real-time performance. Environment features could be automatically extracted and tracked to enable accurate localization of the quadrotors and tracking of the human. Another interesting aspect is to extend our method to work with learning-based approaches that directly predict 3D-pose from images (e.g., [Mehta et al., 2017]). Another interesting challenge is to incorporate our method into a pipeline that capture dense surface deformation via model-fitting or related approaches (e.g., [Rhodin et al., 2015; Robertini et al., 2016]). Finally, we are keen to explore applications of method in graphics, AR/VR and bio-mechanics.

Chapter 8.

Conclusion

This chapter provides a summary of the contributions developed and presented in this thesis. In addition, we also show limitations of our methods and propose potential future research directions.

In this thesis, we investigated the problem of how the boundless possibilities of flying cameras can be transferred to creative people on film sets. Moreover, we wanted to achieve the challenging goal to allow non-drone experts to be able to use drones – without having to know how to control a drone. Two major issues that currently prevent the use of drones as everyday tools on film sets are identified at the beginning of this thesis as:

1. The cognitive load to simultaneously control a drone and produce artistic shot compositions is incredibly high.

2. A film set has an unpredictable and unstructured nature where actors are walking around and scenes may change very quickly.

In contrast to Roberts and Hanrahan [2016] as well as Gebhardt et al. [2016, 2018], we proposed a two-step approach, we suggested a separation planning problem into a real-time local feasible controller and a

high-level planner. In our main contribution of this thesis, we present the two building blocks needed to realize a local feasible controller, which are the basis for intelligent drone cinematography. Our second core contribution is the use of our presented building blocks and development of a real-time drone based human motion capturing system. In order to achieve these goals, we also contributed two methods for GPS less drone navigation.

Keyframing and feasible path following: We provided a very flexible model predictive control formulation that allows to abstract a drone as a flying camera. In contrast to the method presented by Galvane et al. [2018], our solution allows multi subject framing and an easy way to include a wide range of cinematographic concepts by defining key frames. The controller steers the drone in a way that the desired key frame – the frame the user wants to see the resulting camera footage – is fulfilled. Cinematographic concepts such as shot size, subject framing and relative view angles can easily be integrated.

Further, we present a method for drones that allows to track arbitrary high level trajectories with guaranteed constraints. The input trajectories do not need to be physically feasible in the sense of Roberts and Hanrahan [2016]. In addition, we show extensions of the proposed algorithm by adding the possibility to do multi-camera productions using drones. The algorithms have a real-time inter drone collision avoidance, and can film with a reciprocal visibility minimization.

Drone-based motion capturing: In order to show the flexibility of our proposed method, we developed a flying motion capturing system, which is our second core contribution. We developed the first real-time human motion capturing system, which uses drones. We provided a completely self-contained method that allows human skeleton tracking in real-time and controlling of the states of multiple drones in real-time. We demonstrate the provided system in a wide range of compelling experiments, including in and outdoor scenarios

on larger scales. Using the proposed method, for the first time it is possible to do large-scale human motion tracking independent of the environment.

GPS-less position estimation: In addition to the above presented core contributions, we provided a very lightweight and fast Visual Inertial Odometry (VIO) algorithm to accurately estimate the position of a drone. In addition, we present an extension to the presented VIO algorithm that directly includes pixel intensity for position estimation.

8.1. Limitations and Future Research Avenues:

Some of the limitations and future research directions of the presented methods were already mentioned and discussed in the individual sections. Here, we summarize and generalize them and show possible solutions to overcome the limitations.

Real-time drone cinematography: Our method for key-framing and path following addresses the issues we identified to prevent non-experts to use drones on a film set: Too high cognitive load and non-static environment. Although we showed that the presented methods are a step forward in the right direction, they still cannot fully solve the problem: It is relatively easy to over constrain the presented methods. We can illustrate this with two examples: It is not possible for the drone to stay on a path, while defining a shot size of an actor if he moves around. Either, the drone has to leave the path and follow the subject to keep the shot size constant *or* the drone stays on a path and cannot film the subject with the desired shot size. To give another example: We want to film two subjects from the front e.g. to see both faces from the front but they are not exactly looking in the same direction. A single drone cannot fulfill this shot. Either, the drone films the first subject from the front or the second sub-

ject. These two examples show the main limitation of the proposed method and the need of a high-level guidance controller, which is able to provide feasible set points. With feasible we mean set points which are not resulting in an over constraint optimization problem. A possible solution to the above problem of filming two subjects can either be:

(a) Film only one subjects from the front. The shot is fulfilled for one subject, but not for the second.

(b) Use two drones.

It is not a priori clear how to solve this issue. It is a high-level optimization problem where the number of drones or human preferences are part of the optimization objective. However, it is not clear how to set up a cost function. A possible solution is to develop a combined method with learning based cost objectives. This means it could be possible to learn human preferences for example from classical films.

Drone-based motion capturing: In chapter 7 we presented a real-time human motion capturing system using drones and active IR-LED markers. The presented method is the first real-time motion capturing system using drones. One of the biggest limitations of the system is the marker tracking. In order to do a proper labeling of the markers, light patterns have to be detected over time which limits the tracking frame-rate. Therefore, we would directly suggest using a marker less approach using 2D joint estimation algorithms. In addition, to robustify the detection and to reduce visibility problems, motion patterns can be analyzed and drones can be optimally placed around the subject. This will minimize observability issues and therefore lead to a robust motion estimation. In addition, a full body mesh estimation could be added in order to provide a performance capturing system.

Further research directions: In the following, we add further research directions and possible, more detailed follow up projects.

Environment independent position: Providing an absolute position of a drone without the use of GPS is still an unsolved problem. Although there are drones on the market which have a certain degree of autonomy, a precise and absolute position estimate is still missing. In order to use drones on a film set, maybe a marker-based solution could be much more interesting than a complete infrastructure free solution. This would allow providing the necessary flight accuracy. Flying precise, accurate lower to the ground and with a velocities >100km/h is still very challenging but would deliver completely new view perspectives. *VirtualRail Interface:* To define the virtual camera rails presented in chapter 6 a proper interface would be needed. Although we used a Google maps based GUI interface for indoor experiments, this is still not enough if you want e.g. fly accurately in a forest. To fly in such narrow environments with lots of unmapped obstacles such as trees, another interface is needed such as e.g. defining the virtual rail on the ground using markers. *Human Filming Imitation* After talking to producers[1] they mentioned that drone shots are sometimes too perfect. This means, industry as well as research are focusing on perfect and smooth drone shots. However, sometimes the human factor is wanted. E.g., if a producer wants to shoot a scene "out of the eyes of a human". These shots are normally filmed using a camera on the shoulders of a camera operator. This human "motion" could be learned and added on the smooth drone movement. This could give the impression of a human camera operator.

Wildlife filming and inspection: We see also potential in applying the proposed building blocks for cinematography and add new cost terms such as noise or distraction. This can be used e.g. to track animals in the wild without disturbing them.

[1] Singorell.com, .stories.ch

Chapter 9.

Appendix

9.1. Active LED Markers

For reproducibility we detail how we extract marker IDs from the drones onboard camera streams.

Marker labelling; To produce labeled measurements of marker positions in the image, bright blobs corresponding to markers are segmented from the background. Each marker blinks with a distinct bit pattern and when tracked over time the pixel intensities may be converted into a bitstream indicating the on- or off-state of the modulated LED. The extracted bit pattern allows for unique identification of each marker.

Marker detection: Intensity blobs corresponding to our marker candidates are segmented from the background by applying an intensity threshold on input frame I_i. A morphological opening operation is used for noise removal and the remaining connected components are marker candidates $\mathcal{C}_i$. Marker candidates are tracked via a KLT tracker and associated with the newly detected marker candidates.

Bitstream conversion: Each frame I_i provides a sample s_i^j of marker M_j's current signal bit state $b_{cur}^j \in \{0, 1, 0.5\}$, where $b_{cur}^j = 0.5$ denotes a corrupted signal.

Since apparent marker intensity depends on the current state of the blinking LED and extraneous influences, dynamic thresholding is used to classify the state as on (logical 1) or off (logical 0). This marker-specific threshold is computed as the moving average of a marker's intensities over a sample size of N_{window} frames.

Note that our system clocks are not synchronized and hence the time at which the LEDs state switches $t_{transition}$ has to be approximated by a transition window $[t_s, t_e]$. This is done by finding the frames I_i and I_{i+1} where the sample bits of multiple markers change their value, i.e. find i such that $|\{j \,|\, s_i^j \neq s_{i+1}^j\}| > 3$. Because the signal pattern frequency f is known, it can be assumed that $t_{transition} = t_{transition} + \frac{1}{f}$ and the transition window can be updated accordingly. The transition time window allows us grouping samples belonging to the same signal bit b_{cur}^j . Namely, we choose samples:

$$\{s_l^j, ..., s_{l+k}^j\} = \{s_i^j \,|\, timestamp(I_i) > t_e \wedge timestamp(I_i) < t_s + \frac{1}{f}\}$$

Having multiple samples per signal leads to increased robustness of the bit classification process. The resulting signal bit for a Marker j is computed as $b_{cur}^j = f(s_l^j, ...s_{l+k}^j))$ with

$$f(s_l^j, ...s_{l+k}^j)) = \begin{cases} 1, & \text{if } \frac{1}{K}\sum_{i=0}^{K} s_l^{j+i} > 0.6 \\ 0, & \text{if } \frac{1}{K}\sum_{i=0}^{K} s_l^{j+i} < 0.4 \\ 0.5, & \text{otherwise} \end{cases}$$

The resulting bitstream can be used to match the extracted to the known patterns which correspond to unique marker labels.

9.2. 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 $[f_y, f_x]$, the focal point $[C_x, C_y]$, the center of the camera $\mathbf{p}_c$ and its orientation $\mathbf{R}_q$, see Fig. 6.5. For each other drone j, we test if it is inside the cone C. We first compute the view direction $\mathbf{r}_{\text{view}} = \mathbf{R}_q[0,0,1]^T$ of the camera and the intersection point $\mathbf{p}_{\text{int}} = c_{\text{int}}\mathbf{r}_{\text{view}}$, where $c_{\text{int}} = \mathbf{r}_{ij}^T\mathbf{r}_{\text{view}}$ and $\mathbf{r}_{ij} = \mathbf{p}_{qj} - \mathbf{p}_{c_i}$ 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 $\mathbf{p}_{\text{int}}$,

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

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 $\mathbf{x}_{\text{q}}i$ and $quadStatesij$, the cost term

$$c_{\text{v}}(\mathbf{x}_{\text{q}}i, \mathbf{x}_{\text{q}_j}) = \begin{cases} Q_v d_{\text{surf}}^2 & \text{if } d_{\text{surf}} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{9.1}$$

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

Bibliography

(2015). 3DR Solo. http://3drobotics.com/solo.

(2015). APM Autopilot Suite. http://ardupilot.com.

(2015). DJI Ground Station. http://www.dji.com/product/pc-ground-station.

(2015). Parrot SDK. http://developer.parrot.com/.

(2015). VC Technology Litchi Tool. https://flylitchi.com/.

Abramowitz, M., Stegun, I. a., and Miller, D. (1965). *Handbook of Mathematical Functions With Formulas, Graphs and Mathematical Tables (National Bureau of Standards Applied Mathematics Series No. 55)*. Courier Corporation.

Aguiar, A. P., Hespanha, J. P., and Kokotović, P. V. (2008). Performance limitations in reference tracking and path following for nonlinear systems. *Automatica*, 44(3):598–610.

Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988a). Active vision. *International journal of computer vision*, 1(4):333–356.

Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988b). Active vision. *Journal of computer vision*.

Alonso-Mora, J., Naegeli, T., Siegwart, R., and Beardsley, P. (2015). Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots*, 39(1):101–121.

Altuğ, E., Ostrowski, J. P., and Taylor, C. J. (2003). Quadrotor control using dual camera visual feedback. In *ICRA*.

Altuğ, E., Ostrowski, J. P., and Taylor, C. J. (2005). Control of a quadrotor helicopter using dual camera visual feedback. *The International Journal of Robotics Research*.

Anderson, B. D. and Moore, J. B. (2012). *Optimal filtering*. Courier Dover Publications.

Arijon, D. (1976). Grammar of the film language.

Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE*, 76(8):966–1005.

Ballan, L., Taneja, A., Gall, J., Van Gool, L., and Pollefeys, M. (2012). Motion capture of hands in action using discriminative salient points. *Computer Vision–ECCV 2012*, pages 640–653.

Bell, B. M. and Cathey, F. W. (1993). The iterated kalman filter update as a gauss-newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297.

Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Siam.

Bloesch, M., Omari, S., Hutter, M., and Siegwart, R. (2015). Robust visual inertial odometry using a direct ekf-based approach. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 298–304. IEEE.

Blösch, M., Weiss, S., Scaramuzza, D., and Siegwart, R. (2010). Vision based mav navigation in unknown and unstructured environments. In *ICRA, 2010 IEEE international conference on*.

Bogo, F., Kanazawa, A., Lassner, C., Gehler, P., Romero, J., and Black, M. J. (2016). Keep it smpl: Automatic estimation of 3d human pose and shape from a single image. In *European Conference on Computer Vision*, pages 561–578. Springer.

Borrelli, F., Bemporad, A., and Morari, M. (2017). *Predictive control for linear and hybrid systems*. Cambridge University Press.

Breckenridge, W. (1979). Quaternions proposed standard conventions. *Jet Propulsion Laboratory, Pasadena, CA, Interoffice Memorandum*.

Bregler, C. and Malik, J. (1998). Tracking people with twists and exponential maps. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 8–15. IEEE.

Bristeau, P.-J., Callou, F., Vissiere, D., and Petit, N. (2011). The navigation and control technology inside the ar. drone micro uav. *IFAC Proceedings Volumes*, 44(1):1477–1484.

Bristeau, P.-J., Martin, P., Salaün, E., and Petit, N. (2009). The role of propeller aerodynamics in the model of a quadrotor uav. In *Control Conference (ECC), 2009 European*, pages 683–688. IEEE.

Brown, D. C. (1971). Close-range camera calibration.

Bry, A., Richter, C., Bachrach, A., and Roy, N. (2015). Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal of Robotics Research*, 34(7):969–1002.

Castellanos, J. A., Neira, J., and Tardos, J. D. (2004). Limits to the consistency of EKF-based SLAM.

Chen, X. and Yuille, A. L. (2014). Articulated pose estimation by a graphical model with image dependent pairwise relations. In *NIPS*, pages 1736–1744.

Chen, Y., Chien, S. Y.-P., and DESROCHERS, A. A. (1992). General structure of time-optimal control of robotic manipulators moving along prescribed paths. *International Journal of Control*, 56(4):767–782.

Christie, M., Olivier, P., and Normand, J.-M. (2008a). Camera Control in Computer Graphics. *Computer Graphics Forum*, 27(8):2197–2218.

Christie, M., Olivier, P., and Normand, J. M. (2008b). Camera control in computer graphics. *Computer Graphics Forum*, 27(8):2197–2218.

Civera, J., Grasa, O. G., Davison, A. J., and Montiel, J. M. M. (2009). 1-point RANSAC for EKF-based structure from motion. *IROS*.

Cozzi, P. and Stoner, F. (2010). Gpu ray casting of virtual globes. In *ACM SIGGRAPH 2010 Posters*, SIGGRAPH '10, New York, NY, USA. ACM.

Davison, A. J. (2003). Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE.

Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*.

de Aguiar, E., Stoll, C., Theobalt, C., Ahmed, N., Seidel, H.-P., and Thrun, S. (2008). Performance capture from sparse multi-view video. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 98:1–98:10, New York, NY, USA. ACM.

de Palézieux, N., Nägeli, T., and Hilliges, O. (2016). Duo-vio: Fast, light-weight, stereo inertial odometry. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2237–2242.

Denham, W. F. and Pines, S. (1966). Sequential estimation when measurement function nonlinearity is comparable to measurement error. *AIAA journal*, 4(6):1071–1076.

Domahidi, A. and Jerez, J. (2016). FORCES Pro: code generation for embedded optimization. `https://www.embotech.com/FORCES-Pro`.

Domahidi, A., Zgraggen, A. U., Zeilinger, M. N., Morari, M., and Jones, C. N. (2012). Efficient interior point methods for multistage problems arising in receding horizon control. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 668–674. IEEE.

Dou, M., Khamis, S., Degtyarev, Y., Davidson, P., Fanello, S. R., Kowdle, A., Escolano, S. O., Rhemann, C., Kim, D., Taylor, J., Kohli, P., Tankovich, V., and Izadi, S. (2016). Fusion4d: Real-time performance capture of challenging scenes. *ACM Trans. Graph.*, 35(4):114:1–114:13.

Drucker, S. M. and Zeltzer, D. (1994). Intelligent camera control in a virtual environment. In *In Proceedings of Graphics Interface '94*, pages 190–199.

Elhayek, A., de Aguiar, E., Jain, A., Thompson, J., Pishchulin, L., Andriluka, M., Bregler, C., Schiele, B., and Theobalt, C. (2017). Marconi—convnet-based marker-less motion capture in outdoor and indoor scenes. *IEEE transactions on pattern analysis and machine intelligence*, 39(3):501–514.

Engel, J., Sturm, J., and Cremers, D. (2012). Camera-based navigation of a low-cost quadrocopter. In *Proc. of the International Conference on Intelligent Robot Systems*.

Engel, J., Sturm, J., and Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE.

Engel, J., Sturm, J., and Cremers, D. (2014). Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems (RAS)*, 62(11):1646—1656.

Espiau, B., Chaumette, F., and Rives, P. (1992). A new approach to visual servoing in robotics. *Robotics and Automation, IEEE Transactions on*, 8(3):313–326.

Faulwasser, T., Kern, B., and Findeisen, R. (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*, pages 8642–8647. IEEE.

Fischler, M. a. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*.

Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE.

Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., and Pollefeys, M. (2012). Vision-based autonomous mapping and exploration using a quadrotor mav. In *IROS*.

Friedland, B. (1969). Treatment of bias in recursive filtering. *IEEE Transactions on Automatic Control*, 14(4):359–367.

Galvane, Q., Christie, M., Lino, C., and Ronfard, R. (2015). Camera-on-rails: automated computation of constrained camera paths. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, pages 151–157. ACM.

Galvane, Q., Christie, M., Ronfard, R., Lim, C.-K., and Cani, M.-P. (2013). Steering behaviors for autonomous cameras. In *Proceedings of Motion on Games*, pages 93–102. ACM.

Galvane, Q., Fleureau, J., Tariolle, F.-L., and Guillotel, P. (2016). Automated cinematography with unmanned aerial vehicles. In *Eurographics Workshop on Intelligent Cinematography and Editing*. The Eurographics Association.

Galvane, Q., Fleureau, J., Tariolle, F.-L., and Guillotel, P. (2017). Automated cinematography with unmanned aerial vehicles. *arXiv preprint arXiv:1712.04353*.

Galvane, Q., Lino, C., Christie, M., Fleureau, J., Servant, F., Tariolle, F., Guillotel, P., et al. (2018). Directing cinematographic drones. *ACM Transactions on Graphics (TOG)*, 37(3):34.

Ganapathi, V., Plagemann, C., Koller, D., and Thrun, S. (2012). Real-time human pose tracking from range data. In *European conference on computer vision*, pages 738–751. Springer.

Gebhardt, C., Hepp, B., Naegeli, T., Stevsic, S., and Hilliges, O. (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, New York, NY, USA. ACM.

Gebhardt, C., Stevsic, S., and Hilliges, O. (2018). Optimizing for Aesthetically Pleasing Quadrotor Camera Motion. *ACM Trans. Graph.*, 37(4):90:1–90:11.

Geijtenbeek, T. and Pronost, N. (2012). Interactive character animation using simulated physics: A state-of-the-art review. *Comput. Graph. Forum*, 31(8):2492–2515.

Gibbs, B. P. (2011). *Advanced Kalman filtering, least-squares and modeling: a practical handbook*. John Wiley & Sons.

Gleicher, M. and Witkin, A. (1992). Through-the-lens camera control. volume 26, New York, NY, USA. ACM.

Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition.

Hasler, N., Rosenhahn, B., Thormahlen, T., Wand, M., Gall, J., and Seidel, H.-P. (2009). Markerless motion capture with unsynchronized moving cameras. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 224–231. IEEE.

Hausman, K., Kahn, G., Patil, S., Müller, J., and Goldberg, K. (2016). Cooperative Occlusion-Aware Multi-Robot Target Tracking using Optimization. *rll.berkeley.edu*.

Hesch, J. A., Kottas, D. G., Bowman, S. L., and Roumeliotis, S. I. (2013). Towards consistent vision-aided inertial navigation. In *Algorithmic Foundations of Robotics X*. Springer.

Honegger, D., Greisen, P., Meier, L., Tanskanen, P., and Pollefeys, M. (2012). Real-time velocity estimation based on optical flow and disparity matching. In *IROS*.

Hsieh, C.-S. (2000). Robust two-stage kalman filters for systems with unknown inputs. *IEEE Transactions on Automatic Control*, 45(12):2374–2378.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. (2011). KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proc. ACM User Interface Software and Technologies*, UIST '11, pages 559–568.

Jin, H., Favaro, P., and Soatto, S. (2003). A semi-direct approach to structure from motion. *The Visual Computer*.

Jones, E. S. and Soatto, S. (2011). Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research*.

Joubert, N., Jane, L. E., Goldman, D. B., Berthouzoz, F., Roberts, M., Landay, J. A., and Hanrahan, P. (2016). Towards a drone cinematographer: Guiding quadrotor cameras using visual composition principles. *CoRR*, abs/1610.01691.

Joubert, N., Roberts, M., Truong, A., Berthouzoz, F., and Hanrahan, P. (2015). An interactive tool for designing quadrotor camera shots. volume 34, pages 238:1–238:11, New York, NY, USA. ACM.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894.

Katz, S. D. (1991). *Film directing shot by shot: visualizing from concept to screen*. Gulf Professional Publishing.

Kerl, C., Sturm, J., and Cremers, D. (2013). Robust odometry estimation for rgb-d cameras. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Kerrigan, E. C. and Maciejowski, J. M. (2000). Soft constraints and exact penalty functions in model predictive control. In *Control 2000 Conference, Cambridge*.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*.

Lam, D., Manzie, C., and Good, M. (2010). Model predictive contouring control. In *49th IEEE Conference on Decision and Control (CDC)*, pages 6137–6142. IEEE.

Lefferts, E. J., Markley, F. L., and Shuster, M. D. (1982a). Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*, 5(5):417–429.

Lefferts, E. J., Markley, F. L., and Shuster, M. D. (1982b). Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*.

Leutenegger, S., Furgale, P. T., Rabaud, V., Chli, M., Konolige, K., and Siegwart, R. (2013). Keyframe-based visual-inertial slam using nonlinear optimization. In *Robotics: Science and Systems*.

Li, M. and Mourikis, A. I. (2013). High-precision, consistent EKF-based visual-inertial odometry. *International Journal of Robotics Research*.

Li, R., Pang, M., Zhao, C., Zhou, G., and Fang, L. (2016). Monocular long-term target following on uavs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 29–37.

Lim, H. and Sinha, S. (2015). Monocular localization of a moving person onboard a quadrotor mav.

Lima, P. and Floreano, D. (2013). Audio-based Relative Positioning System for Multiple Micro Air Vehicle Systems. *RSS2013*, (266470).

Liniger, A., Domahidi, A., and Morari, M. (2015). Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647.

Lino, C. and Christie, M. (2015). Intuitive and efficient camera control with the toric space. *ACM Transactions on Graphics (Proc. SIGGRAPH 2015)*, 34(4):82.

Lino, C., Christie, M., Ranon, R., and Bares, W. (2011). The director's lens: An intelligent assistant for virtual cinematography. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 323–332, New York, NY, USA. ACM.

Liu, H., Wei, X., Chai, J., Ha, I., and Rhee, T. (2011). Realtime human motion control with a small number of inertial sensors. In *Symposium on Interactive 3D Graphics and Games*, pages 133–140. ACM.

Lupashin, S., Schollig, A., Hehn, M., and D'Andrea, R. (2011). The flying machine arena as of 2010. In *IEEE ICRA '11*, pages 2970–2971.

MacAllister, B., Butzke, J., Kushleyev, A., Pandey, H., and Likhachev, M. (2013). Path planning for non-circular micro aerial vehicles in constrained environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3933–3940. IEEE.

Mademlis, I., Mygdalis, V., Nikolaidis, N., and Pitas, I. (2018). Challenges in autonomous uav cinematography: An overview.

Madgwick, S. (2010). An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25:113–118.

Martinez-Cantin, R. and Castellanos, J. a. (2006). Bounding uncertainty in EKF-SLAM: The robocentric local approach. *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*.

Mascelli, J. V. (1965). *The five C's of cinematography*. Grafic Publications.

Matthies, L., Szeliski, R., and Kanade, T. (1988). Incremental estimation of dense depth maps from image sequences. In *Computer Vision and Pattern Recognition, 1988. Proceedings CVPR'88., Computer Society Conference on*. IEEE.

Mehta, D., Sridhar, S., Sotnychenko, O., Rhodin, H., Shafiei, M., Seidel, H.-P., Xu, W., Casas, D., and Theobalt, C. (2017). Vnect: Real-time 3d human pose estimation with a single rgb camera. volume 36.

Mellinger, D. and Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation*

(ICRA), 2011 IEEE International Conference on, pages 2520–2525. IEEE.

Michael, N., Mellinger, D., Lindsey, Q., and Kumar, V. (2010). The grasp multiple micro-uav testbed. *Robotics Automation Magazine, IEEE*, 17(3):56–65.

Moeslund, T. B., Hilton, A., and Krüger, V. (2006). A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2):90–126.

Molton, N., Davison, A. J., and Reid, I. (2004). Locally planar patch features for real-time structure from motion. In *BMVC*.

Montiel, J., Civera, J., and Davison, A. (2006). Unified inverse depth parametrization for monocular SLAM. *Analysis*.

Mourikis, A. I. and Roumeliotis, S. I. (2007). A multi-state constraint kalman filter for vision-aided inertial navigation. In *Robotics and automation, 2007 IEEE international conference on*, pages 3565–3572. IEEE.

Mourikis, A. I., Trawny, N., Roumeliotis, S. I., Johnson, A. E., Ansar, A., and Matthies, L. (2009). Vision-aided inertial navigation for spacecraft entry, descent, and landing. *Robotics, IEEE Transactions on*.

Mueller, M. W. and D'Andrea, R. (2013). A model predictive controller for quadrocopter state interception. In *Control Conference (ECC), 2013 European*, pages 1383–1389. IEEE.

Nägeli, T., Alonso-Mora, J., Domahidi, A., Rus, D., and Hilliges, O. (2017). Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization. *IEEE Robotics and Automation Letters*, 2(3):1696–1703.

Nägeli, T., Meier, L., Domahidi, A., Alonso-Mora, J., and Hilliges, O. (2017). Real-time planning for automated multi-view drone cinematography. *ACM Trans. Graph.*, 36(4):132:1–132:10.

Newcombe, R. A. and Davison, A. J. (2010). Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE.

Newcombe, R. A., Davison, A. J., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., Molyneaux, D., Hodges, S., Kim, D., and Fitzgibbon, A. (2011a). Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE.

Newcombe, R. A., Fox, D., and Seitz, S. M. (2015). Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352.

Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011b). Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE.

Newell, A., Yang, K., and Deng, J. (2016). Stacked hourglass networks for human pose estimation. In *ECCV*, pages 483–499.

Nägeli, T., Conte, C., Domahidi, A., Morari, M., and Hilliges, O. (2014). Environment-independent formation flight for micro aerial vehicles. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1141–1146.

Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. IEEE.

Oikonomidis, I., Kyriazis, N., and Argyros, A. A. (2012). Tracking the articulated motion of two strongly interacting hands. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1862–1869. IEEE.

Pietzsch (2008). Efficient Feature Parameterisation for Visual SLAM Using Inverse Depth Bundles. *Bmvc*.

Poiesi, F. and Cavallaro, A. (2015). Distributed vision-based flying cameras to film a moving target. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2453–2459. IEEE.

Pons-Moll, G., Romero, J., Mahmood, N., and Black, M. J. (2015). Dyna: A model of dynamic human shape in motion. *ACM Trans. Graph.*, 34(4):120:1–120:14.

Pugh, J. and Martinoli, A. (2006). Relative localization and communication module for small-scale multi-robot systems. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 188–193. IEEE.

Qin, S. J. and Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *IEEE ICRA Workshop on Open Source Software*.

Rhodin, H., Robertini, N., Richardt, C., Seidel, H.-P., and Theobalt, C. (2015). A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 765–773.

Richards, A. and How, J. (2004). Decentralized model predictive control of cooperating UAVs. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4286–4291 Vol.4. IEEE.

Robertini, N., Casas, D., Rhodin, H., Seidel, H.-P., and Theobalt, C. (2016). Model-based outdoor performance capture. In *Proceedings of the 2016 International Conference on 3D Vision (3DV 2016)*.

Roberts, M. and Hanrahan, P. (2016). Generating dynamically feasible trajectories for quadrotor cameras. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)*, 35(4).

Roetenberg, D., Luinge, H., and Slycke, P. (2007). Moven: Full 6dof human motion tracking using miniature inertial sensors. *Xsen Technologies, December*, 2(3):8.

Rose, C., Guenter, B., Bodenheimer, B., and Cohen, M. F. (1996). Efficient generation of motion transitions using spacetime constraints. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 147–154. ACM.

Rosten, E. and Drummond, T. (2005). Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision (ICCV)*.

Roumeliotis, S. I., Sukhatme, G. S., and Bekey, G. A. (1999). Circumventing dynamic modeling: evaluation of the error-state kalman filter applied to mobile robot localization. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1656–1663 vol.2.

Sakane, S., Ish, M., and Kakikura, M. (1987). Occlusion avoidance of visual sensors based on a hand-eye action simulator system: Heaven. *Advanced robotics*, 2(2):149–165.

Saunders, J. B., Call, B., Curtis, A., Beard, R. W., and McLain, T. W. (2005). Static and dynamic obstacle avoidance in miniature air vehicles. *AIAA Infotech@ Aerospace*, 96.

Schöps, T., Engel, J., and Cremers, D. (2014). Semi-dense visual odometry for AR on a smartphone. In *ISMAR*.

Schulman, J., Duan, Y., Ho, J., Lee, A., and Awwal, I. (2014). Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*.

Schwarz, L., Mateus, D., and Navab, N. (2009). Discriminative human full-body pose estimation from wearable inertial sensor data. *Modelling the Physiological Human*, pages 159–172.

Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., and Moore, R. (2013). Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124.

Smith, S. L., Schwager, M., and Rus, D. (2011). Persistent monitoring of changing environments using a robot with limited range sensing. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5448–5455. IEEE.

Song, J., Wang, L., Van Gool, L., and Hilliges, O. (2017). Thin-slicing network: A deep structured model for pose estimation in videos. *arXiv preprint arXiv:1703.10898*.

Srikanth, M., Bala, K., and Durand, F. (2014). Computational rim illumination with aerial robots. In *Proceedings of the Workshop on Computational Aesthetics*, pages 57–66. ACM.

Starck, J. and Hilton, A. (2003). Model-based multiple view reconstruction of people. In *null*, page 915. IEEE.

Stoll, C., Hasler, N., Gall, J., Seidel, H.-P., and Theobalt, C. (2011). Fast articulated motion tracking using a sums of gaussians body model. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 951–958. IEEE.

Swain, M. J. and Stricker, M. A. (1993). Promising directions in active vision. *International Journal of Computer Vision*, 11(2):109–126.

Tanskanen, P., Kolev, K., Meier, L., Camposeco, F., Saurer, O., and Pollefeys, M. (2013). Live metric 3d reconstruction on mobile phones. In *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE.

Tanskanen, P., Naegeli, T., Pollefeys, M., and Hilliges, O. (2015). Semi-direct ekf-based monocular visual-inertial odometry. *IROS*.

Tarabanis, K., Tsai, R. Y., and Allen, P. K. (1991). Automated sensor planning for robotic vision tasks. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 76–82. IEEE.

Tautges, J., Zinke, A., Krüger, B., Baumann, J., Weber, A., Helten, T., Müller, M., Seidel, H.-P., and Eberhardt, B. (2011). Motion reconstruction using sparse accelerometer data. *ACM Transactions on Graphics (TOG)*, 30(3):18.

Taylor, J., Shotton, J., Sharp, T., and Fitzgibbon, A. (2012). The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 103–110. IEEE.

Tekin, B., Márquez-Neila, P., Salzmann, M., and Fua, P. (2016). Fusing 2d uncertainty and 3d cues for monocular body pose estimation. *arXiv preprint arXiv:1611.05708*.

Tompson, J. J., Jain, A., LeCun, Y., and Bregler, C. (2014). Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS*, pages 1799–1807.

Toshev, A. and Szegedy, C. (2014). Deeppose: Human pose estimation via deep neural networks. In *CVPR*, pages 1653–1660.

Trawny, N. and Roumeliotis, S. I. (2005). Indirect Kalman Filter for 3D Attitude Estimation. *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep.*

Tsotsos, K., Chiuso, A., and Soatto, S. (2014). Robust inference for visual-inertial sensor fusion. *arXiv preprint arXiv:1412.4862*.

University of Dallas (2016). Elements of Cinematography: Camera. `http://www.utdallas.edu/atec/midori/Handouts/camera.htm`. Accessed: 2016-09-02.

US, D. S., Kim, H., and Sastry, S. (2003). *Decentralized reflective model predictive control of multiple flying robots in dynamic environment.*

von Marcard, T., Rosenhahn, B., Black, M. J., and Pons-Moll, G. (2017). Sparse inertial poser: Automatic 3d human pose estimation from sparse imus. *Comput. Graph. Forum*, 36(2):349–360.

Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional pose machines. In *CVPR*, pages 4724–4732.

Weiss, S. and Siegwart, R. (2011). Real-time metric state estimation for modular vision-inertial systems. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE.

Wendel, A., Maurer, M., Graber, G., Pock, T., and Bischof, H. (2012). Dense reconstruction on-the-fly. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. to appear.

Witkin, A. and Kass, M. (1988). Spacetime constraints. *ACM Siggraph Computer Graphics*, 22(4):159–168.

Xu, L., Liu, Y., Cheng, W., Guo, K., Zhou, G., Dai, Q., and Fang, L. (2017). Flycap: Markerless motion capture using multiple autonomous flying cameras. *IEEE Transactions on Visualization & Computer Graphics*, (1):1–1.

Zhang, W., Mueller, M. W., and D'Andrea, R. (2016). A controllable flying vehicle with a single moving part. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3275–3281. IEEE.

Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE MultiMedia*.

Zhou, X., Liu, S., Pavlakos, G., Kumar, V., and Daniilidis, K. (2018). Human motion capture using a drone. *arXiv preprint arXiv:1804.06112*.

Zhou, X., Zhu, M., Leonardos, S., Derpanis, K. G., and Daniilidis, K. (2016). Sparseness meets deepness: 3d human pose estimation from monocular video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4966–4975.

Zollhöfer, M., Nießner, M., Izadi, S., Rehmann, C., Zach, C., Fisher, M., Wu, C., Fitzgibbon, A., Loop, C., Theobalt, C., et al. (2014). Real-time non-rigid reconstruction using an rgb-d camera. *ACM Transactions on Graphics (TOG)*, 33(4):156.