Photorealistic Sensor Simulation for Perception-driven Robotics using Virtual Reality

by

Winter Joseph Guerra

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Author .....
Department of Electrical Engineering and Computer Science
Aug 23, 2019

Certified by .....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Photorealistic Sensor Simulation for Perception-driven Robotics using Virtual Reality

by

Winter Joseph Guerra

Submitted to the Department of Electrical Engineering and Computer Science
on Aug 23, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In recent years, intensive research has centered around using small, perception-driven robotic systems (*e.g.* quadrotor vehicles) for complex tasks at operational speeds. Although much progress has been made towards that end in the fields of online-planning, fast-perception, and agile-control, most robotic systems are still confined to controlled laboratory settings due to cost, safety, and repeatability.

In this thesis, we introduce a few novel contributions that we believe could assist the greater robotics community to bring their robotics systems out of the lab and into the real world. First, we introduce *FlightGoggles*, a photorealistic sensor simulator for perception-driven robotic vehicles. FlightGoggles provides photorealistic exteroceptive sensor simulation using graphics assets generated with photogrammetry and provides the ability to combine synthetic exteroceptive measurements generated *in silico* in real time and vehicle dynamics and proprioceptive measurements generated *in motio* by vehicle(s) in flight in a motion-capture facility. Second, we present *The Blackbird Dataset*, a large-scale dataset for UAV perception in aggressive flight containing over 10 hours of flight data across 171 flights at velocities up to $13.8\,\mathrm{m\,s^{-1}}$ in 5 environments with some dynamic elements. Third, we introduce a virtual reality framework for FlightGoggles that enables multi-agent or robot-human interaction in a safe manner by superimposing position data from multiple motion capture spaces into a unified virtual reality environment. Fourth, we propose an extension of FlightGoggles using augmented reality for aircraft-in-the-loop experiments that aims to aid sim-to-real transfer from simulated to real-world cameras. Lastly, we study applications of FlightGoggles in the greater robotics community through the AlphaPilot autonomous drone racing challenge and survey approaches and results from the top AlphaPilot teams, which may be of independent interest.

Thesis Supervisor: Sertac Karaman
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

First and foremost, I'd like to thank my advisor Sertac Karaman for his continued support and guidance. His unbridled excitement and interest toward real-world issues in robotics is infectious and has helped motivate me to explore a constant stream of new ideas during my Masters degree. In addition, I'd like to thank him for always being available when most needed – his guiding voice of reason has helped me overcome many hurdles over the years.

I'd also like to thank my colleagues at the MIT FAST Lab for their advice, companionship, and assistance during countless late-night paper writing sessions. In particular, I'd like to thank Varun Murali for his expert knowledge on SLAM and willingness to humor my outlandish ideas, Thomas Sayre-McCord for introducing me to the world of robotic academia, Igor Spasojevic for his camaraderie and amazing ability to disentangle large mathematical systems, Amado Antonini for his help laying the ground work for The Blackbird Dataset, Ezra Tal for his ability to make quadrotors do incredible acrobatic feats, Murat Bronz for his encyclopedic knowledge of aircraft design, Gilhyun Ryou for his assistance with experiments and demos, and Dave McCoy for his electrical engineering wizardry. Additionally, I'd like to thank Jin Gao for helping the lab run smoothly and handling my last-minute equipment requests. My heartfelt gratitude goes to Anne Hunter at the MIT EECS office, who has been a constant advocate for me. Finally, I'd like to thank my close friends Dong-Ki Kim, Schuyler Gaillard, Jon Spyreas, Mai Li Goodman, and Diana Wofk for their kind words of encouragement.

The work in this thesis was supported in part by NVIDIA, MIT Lincoln Laboratory, the Office of Naval Research (ONR), and partly by the Army Research Lab's DCIST program. Their support is greatly appreciated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation & Related Work

Simulation systems have long been an integral part of the development of robotic vehicles. They allow engineers to identify errors early on in the development process, and allow researchers to rapidly prototype and demonstrate their ideas. Despite their success in accelerating development, many researchers view results generated in simulation systems with skepticism, as any simulation system is some abstraction of reality and will disagree with reality at some scale. This skepticism towards results generated exclusively in simulations studies is exemplified by Rodney Brooks' well-known quote from 1993: "[experiment] simulations are doomed to succeed ... [because] simulations cannot be made sufficiently realistic"[1].

Despite the skepticism towards simulation results, several trends have emerged in recent years that have driven the research community to develop better simulation systems out of necessity. A major driving trend towards realistic simulators stems from the emergence of data-driven algorithmic methods in robotics, for instance, based on machine learning methods that require extensive data. Simulation systems provide not only massive amounts of data, but also the labels required for training algorithms. For example, simulation systems can provide a safe environment for learning from experience useful for reinforcement learning methods [2], [3]. This driving trend has posed a critical need to develop better, more realistic simulation

systems.

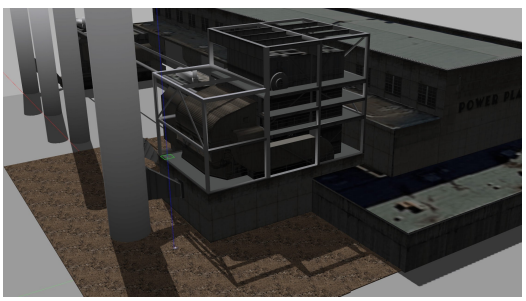
Several enabling trends have also recently emerged that allow for better, more-realistic simulation systems to be developed. The first enabling trend is the development of new computing resources that enable realistic rendering. The rapid evolution of game engine technology, particularly 3D graphics rendering engines, has made available advanced features such as improved material shaders, real-time reflections, volumetric lighting, and advanced illumination through deferred rendering pipelines. Particularly, the maturation of off-the-shelf software packages such as Unreal Engine [4] and Unity [5], makes them suitable for high-fidelity rendering in applications beyond video games, such as robotics simulation. Simultaneously, next-generation graphics processors simply pack more transistors, and the transistors are better organized for rendering purposes, *e.g.*, for real-time ray tracing. In addition, they incorporate computation cores that utilize machine learning, for instance, trained with pictures of real environments to generate realistic renderings [6]. This trend is an opportunity to utilize better software and hardware to realize realistic sensor simulations. The second enabling trend stems from the proliferation of motion capture facilities for robotics research, enabling precise tracking of robotic vehicles and humans through various technologies, such as infrared cameras, laser tracking, and ultra-wide band radio. These facilities provide the opportunity to incorporate real motion and behavior of vehicles and humans into the simulation in real time. This trend provides the potential to combine the efficiency, safety, and flexibility of simulation with real-world physics and agent behavior.

Traditionally, simulation systems embody "models" of the vehicles and the environment, which are used to emulate what the vehicles sense, how they move, and how their environment adapts. In this paper, we present two concepts that use "data" to drive realistic simulations. First, we heavily utilize *photogrammetry* to realistically simulate exteroceptive sensors. For this purpose, we photograph real-world objects, and reconstruct them in the simulation environment. Almost all objects in our simulation environment are, in fact, a rendering of a real-world object. This approach allows realistic renderings, as shown in Fig. 1-1. Second, we utilize a novel *virtual-*

Figure 1-1: FlightGoggles renderings of the *Abandoned Factory* environment, designed for autonomous drone racing. Note the size of the environment and the high level of detail.

reality system to realistically embed inertial sensors, vehicles dynamics, and human behavior into the simulation environment. Instead of modeling these effects, we place vehicles and human actors in motion-capture facilities. We acquire the pose of the vehicles and the configuration of the human actors in real time, and create their avatars in the simulation environment. For each autonomous vehicle, its proprioceptive measurements are acquired using on-board sensors, *e.g.*, inertial measurement units and odometers, and exteroceptive sensors are rendered photorealistically in real time. In addition, the human behavior observed by the vehicles is generated by humans reacting to the simulation. In other words, vehicles embedded in the FlightGoggles simulation system experience real dynamics, real inertial sensing, real human behavior, and synthetic exteroceptive sensor measurements rendered photorealistically effectively by transforming photographs of real-world objects.

The combination of real physics and data-driven exteroceptive sensor simulation that FlightGoggles provides is not achieved in traditional simulation systems. Such systems are typically built around a physics engine that simulates vehicles and the environment based on a "model", most commonly a system of ordinary or partial differential equations. While these models may accurately exemplify the behavior of a general vehicle or actor, this is not sufficient to ensure that simulation results transfer to the real world. Complicated aspects of vehicle dynamics, *e.g.*, vibrations and unsteady aerodynamics, and of human behavior may significantly affect results, but can be very challenging to accurately capture in a physics model. For an overview of pop-



(a) ETHz RotorS

(b) Microsoft AirSim

(c) FlightGoggles

Figure 1-2: Environment renders in various simulation software.

ular physics engines, the reader is referred to [7]. In order to generate exteroceptive sensor data, robotics simulators employ a graphics rendering engine in conjunction with the physics engine. A popular example is Gazebo [8], which lets users select various underlying engines. It is often used in combination with the Robot Operating System (ROS) to enable hardware-in-the-loop simulation. However, Gazebo is generally not capable of photorealistic rendering. Specifically, for unmanned aerial vehicles simulation, two popular simulators that are built on Gazebo are the Hector Quadrotor package [9] and RotorS [10]. Both simulators include vehicle dynamics and exteroceptive sensor models, but lack the capability to render photorealistic camera streams. AirSim, on the other hand, is purposely built on the Unreal rendering engine to enable rendering of photorealistic camera streams from autonomous vehicles, but still suffers from the shortcomings of typical physics engines when it comes to vehicle dynamics and inertial measurements [11]. Fig. 1-2 shows a comparison of photorealism in some of the aforementioned simulators. It serves to highlight the shift towards using video game rendering engines to improve realism in robotics simulation.

The rise of data-driven algorithms for autonomous robotics, has bolstered the

need for extensive labeled data sets. Simulation offers an alternative to experimental data gathering. Clearly, there are many advantages to this approach, *e.g.*, cost efficiency, safety, repeatability, and essentially unlimited quantity and diversity. In recent years, several synthetic, or virtual, datasets have appeared in literature. For example, Synthia [12] and Virtual KITTI [13] use Unity to generate photorealistic renders of an urban environment, and ICL-NUIM [14] provides synthetic renderings of an indoor environment based on pre-recorded handheld trajectories. The Blackbird Dataset [15] includes real-world ground truth and inertial measurements of a quadrotor in motion capture, and photorealistic camera imagery rendered in FlightGoggles. The open-source availability of FlightGoggles and its photorealistic assets enables users to straightforwardly generate additional data, including real-time photorealistic renders based on real-world vehicles and actors.

1.2 Publications

While fulfilling the Master of Engineering requirements at MIT, the author of this thesis has co-authored the following papers. Excerpts of the papers below have been included in this thesis with permission from the authors.

1. A. Antonini*, W. Guerra*, V. Murali, T. Sayre-McCord, and S. Karaman, "The Blackbird UAV Dataset," *The International Journal of Robotics Research (IJRR)*, 2019 (Invited Paper. Submitted for review.) [16]

2. W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2019. [17]

3. V. Murali, I. Spasojevic, W. Guerra, and S. Karaman, "Perception-aware trajectory generation for aggressive quadrotor flight using differential flatness," in *American Control Conference (ACC)*, 2019. [18]

*Both authors contributed equally to this work.

4. A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, "The Blackbird dataset: A large-scale dataset for UAV perception in aggressive flight," in *International Symposium on Experimental Robotics (ISER)*, 2018. [15]

5. T. Sayre-McCord, W. Guerra, A. Antonini, J. Arneberg, A. Brown, G. Cavalheiro, Y. Fang, A. Gorodetsky, D. McCoy, S. Quilter, F. Riether, E. Tal, Y. Terzioglu, L. Carlone, and S. Karaman, "Visual-inertial navigation algorithm development using photorealistic camera simulation in the loop," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2566–2573. [19]

1.3 Contributions

1. A novel, open-source simulation framework FlightGoggles for vehicle in-the-loop simulation using virtual reality for development and testing of perception-driven robotics applications. A plug-and-play ROS framework for *in silico* quadrotor simulation is also provided (see Chapter 2).

2. A high-definition, open-source Unity3D *Abandoned Factory* environment captured using photogrammetry for robotics development (see Section 3.1). The included environment contains over 40 million triangles and 1,050 object instances of 84 unique models captured with photogrammetry.

3. A large-scale dataset for UAV perception in aggressive flight containing over 10 hours of flight data across 171 flights at velocities up to $13.8\,\mathrm{m\,s^{-1}}$ in 5 environments with static and dynamic environmental elements (see Section 4.4).

4. An overview of the simulation-phase of the AlphaPilot autonomous drone racing challenge (see Section 4.3). A survey of approaches and results from the top AlphaPilot teams is also provided (see Section 4.3.2).

5. Real-world comparison experiments using FlightGoggles and a UAV with a *Point Grey Flea3 monochrome camera* verify VIO results from the FlightGog-

gles aircraft-in-the-loop simulator was comparable to results using a real camera for visual state estimation (see Section 4.1).

6. Aircraft-in-the-loop experiments using FlightGoggles validate the use of *Perception-aware trajectory generation* methods for improving visual state estimation in environments with non-uniform distribution of visual features (see Section 4.2).

7. A virtual reality framework for FlightGoggles that enables multi-agent or robot-human interaction in a safe manner by superimposing position data from multiple motion capture spaces into a unified virtual reality environment (see Section 4.5).

8. An augmented reality framework for FlightGoggles allowing for seamless transition between simulation and real-world experiments. We introduce an offline proof-of-concept experiment using real-world flight data and a UAV equipped with a real camera (see Section 4.6).

1.4 Organization

This thesis is organized as follows. Chapter 2 provides an overview of the FlightGoggles system architecture, including interfacing with real-world vehicles and actors in motion capture facilities. Chapter 3 outlines the photogrammetry process and the resulting virtual environment. Chapter 3 also details the features of the rendering pipeline used in FlightGoggles and the exteroceptive sensor models available. Chapter 4 describes several applications of FlightGoggles for perception-driven robotics. These applications include high-speed visual state estimation (see Section 4.1), perception-aware planning (see Section 4.2), the AlphaPilot autonomous drone racing challenge (see Section 4.3), large-scale datasets (see Section 4.4, and safe interactions with dynamic environmental elements and actors (see Section 4.5). Chapter 5 summarizes the thesis and the contributions introduced and suggests directions for future exploration using FlightGoggles, particularly by leveraging augmented reality for seamless sim-to-real transfer of robotic algorithms.

Chapter 2

System Architecture

2.1 Overview

FlightGoggles is based on a modular architecture, as shown in Fig. 2-1. This architecture provides the flexibility to tailor functionality for a specific simulation scenario involving real and/or simulated vehicles, and possibly human interactors. As shown in the figure, FlightGoggles' central component is the Unity game engine. It utilizes position and orientation information to simulate camera imagery and exteroceptive sensors, and to detect collisions. Collision checks are performed using polygon colliders, and results are output to be included in the dynamics of simulated vehicles.

Simulation scenarios may also include real-world vehicles through the use of a motion capture system. In this case, Unity simulation of camera figures and exteroceptive sensors, and collision detection are based on the real-world vehicle position and orientation. This type of *vehicle-in-the-loop simulation* can be seen as an extension of customary hardware-in-the-loop configurations. It not only includes the vehicle hardware, but also the actual physics of processes that are challenging to simulate accurately, such as aerodynamics (including effects of turbulent air flows), and inertial measurements subject to vehicle vibrations. FlightGoggles provides the novel combination of real-life vehicle dynamics and proprioceptive measurements, and simulated photorealistic exteroceptive sensor simulation. It allows for real-life physics, flexible exteroceptive sensor configurations, and obstacle-rich environments without

the risk of actual collisions. FlightGoggles also allows scenarios involving both humans and vehicles, colocated in simulation but placed in different motion capture rooms, *e.g.*, for safety.

Dynamics states, control inputs, and sensor outputs of real and simulated vehicles, and human interactors are available to the user through the FlightGoggles API. In order to enable message passing between FlightGoggles nodes and the API, the framework can be used with either ROS [20] or LCM [21]. The FlightGoggles simulator can be run headlessly on an Amazon Web Services (AWS) cloud instance to enable real-time simulation on systems with limited hardware.

Dynamic elements, such as moving obstacles, lights, vehicles, and human actors, can be added and animated in the environment in real-time. Using these added elements, users can change environment lighting or simulate complicated human-vehicle, vehicle-vehicle, and vehicle-object interactions in the virtual environment. In Section 4.5, we describe an use case involving a dynamic human actor. In this scenario, skeleton tracking motion capture data is used to render a 3D model of the human in the virtual FlightGoggles environment. The resulting render is observed in real-time by a virtual camera attached to a quadrotor in real-life flight in a different motion capture room (see Fig. 4-23).

Much of the work presented in this chapter has been published in International Conference on Intelligent Robots and Systems (IROS) in 2019 [17] and in the International Robotics and Automation Conference (ICRA) in 2018 [19].

2.2 Vehicle In-the-loop Simulation

Simulation of images through the FlightGoggles framework is enabled via a ground station computer featuring an NVIDIA RTX8000 GPU for rapid rendering of images based on the motion capture location of the UAV. The simulation of imagery is performed by creating an environment in Unity that contains a virtual world for the UAV, and one or more camera objects which are attached to a ZeroMQ (ZMQ) [22] socket. Over ZMQ, the various parameters of the camera may be set, most impor-

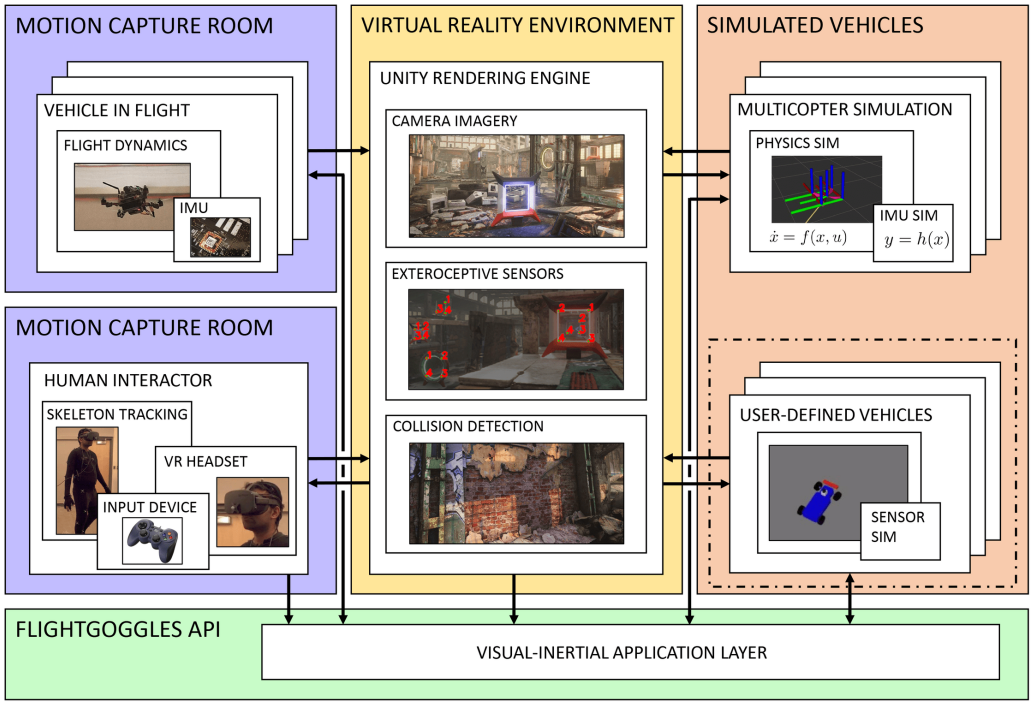


Figure 2-1: Overview of FlightGoggles system architecture. Pose data of real and simulated vehicles, and human interactors is used by the Unity rendering engine. Detected collision can be incorporated in simulated vehicle dynamics, and rendered figures can be displayed in a VR headset. All dynamics states, control inputs, and sensor outputs of real and simulated vehicles, and human interactors are available through the FlightGoggles API.

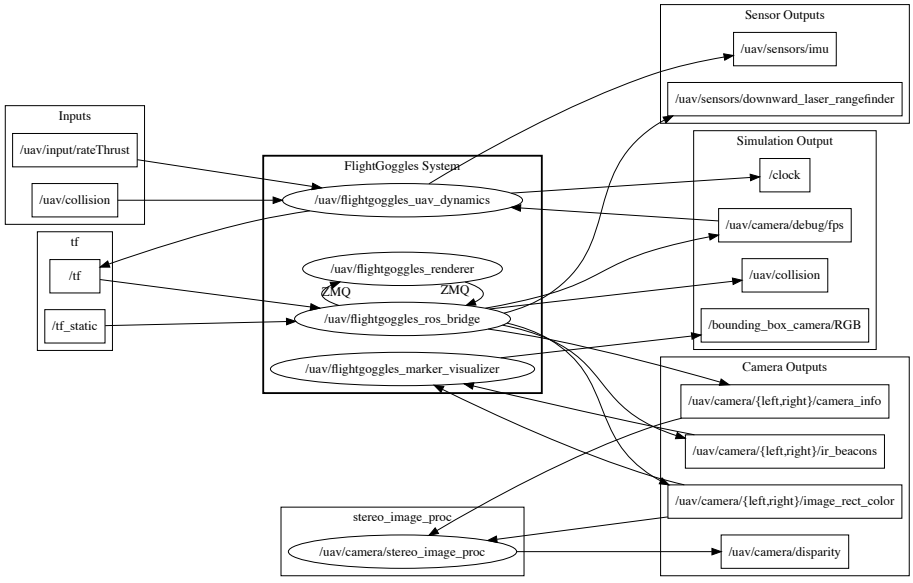


Figure 2-2: Node graph of ROS-version of the FlightGoggles simulation framework.

tantly the camera pose can be set in real time based on the motion capture position of the UAV. For each pose of the camera received, the Unity camera object returns a timestamped image of the virtual reality environment as it would be seen from that pose (see Fig. 2-1). Because of the networking limitations of sending full images wirelessly to the UAV, for our VIO state estimation experiments using simulated imagery, the vision front-end is executed on the ground station computer at a speed that the on-board GPU can execute, and only feature data is sent wirelessly to the UAV for state estimation. The total delay in receiving visual data on the UAV (rendering and wireless transmission) is primarily Gaussian around $37 \pm 8\ ms$ with 1.3% outliers above two standard deviations due to wireless network bottlenecks. For comparison, the time from image acquisition to processed data with our live camera is $15 \pm 5\ ms$.

While our image simulation system may be used for traditional Hardware in the Loop (HiL) simulations (simulated dynamics and inertial measurements with real decision making) or with logged data from a real UAV (real dynamics and inertial measurements, pre-determined decisions), it was implemented with the intention of running in real time while the UAV is in the air for Photorealistic (exteroceptive) sensor simulation in the Loop (real dynamics and inertial measurements, online decision making). By running all systems in real time the vehicle-in-the-loop system comes as close as possible to a real camera running on-board the UAV, allowing visual algorithms to be used in the decision making loop.

2.3 Motion Capture Setup

For vehicle-in-the-loop simulations, an OptiTrack motion capture room is used for estimating the ground truth pose of the UAV in flight. For all works presented in this thesis, OptiTrack Prime 17W cameras were used at either 120 Hz or (more commonly) 360 Hz with millimeter-accurate calibration. On the UAV, four motion capture balls are placed in a square aligned with X, Y axis of the on-board IMU and help automate axis alignment of the motion-capture rigid body with the on-board IMU (see Figure 4-14 for an example implementation).

2.4 Time Synchronization

For most vision-based experiments, accurate and consistent timestamping of camera images is paramount. To ensure accurate timestamping of rendered images, the system clocks of motion capture computer, the ground station computer, and the UAV must be synchronized. We use Chrony[23] to automatically synchronize the system clocks of the UAV to the ground station computer, with an average deviation that is sub-microsecond. Due to operating system restrictions that conflict with Chrony, we perform a different estimation technique to synchronize the timestamps of the motion capture camera exposures with the system clock of the ground station computer which uses leverages latency statistics exposed by the motion capture system.

During vehicle-in-the-loop experiments, ground truth pose data of the UAV in flight is captured by the motion capture cameras at time $t^{(i)}_{mocap}$, where $t^{(i)}_{mocap}$ is the time in nanoseconds of the middle of the motion capture camera exposure at frame i with respect to the system clock of *motion capture computer*. Using latency statistics from the motion capture system, we estimate the constant clock offset between the motion capture computer and the ground station $\Delta t^{mocap}_{groundstation}$. Using this offset, we derive the actual timestamp for each given motion capture frame with respect to the ground station computer $t^{(i)}_{groundstation} = t^{(i)}_{mocap} + \Delta t^{mocap}_{groundstation}$, which is equal to $t^{(i)}_{UAV}$ through clock synchronization using Chrony.

After the motion capture pose enters the FlightGoggles system, FlightGoggles keeps track of the original exposure time $t^{(i)}_{UAV}$ for each render request throughout the sensor simulation pipeline. Thus, each rendered image output from FlightGoggles is timestamped using $t^{(i)}_{UAV}$, the timestamp of the ground truth pose from which this frame was rendered. Using cross-correlation from a 100 Hz IMU, we have noted that this method of timestamping is accurate to within ± 5 ms, which is less than one IMU datapoint and provides validation for the synchonization method used.

2.5 Dynamic Clock Scaling for Offline Simulation

FlightGoggles provides optional dynamic clock scaling to guarantee a nominal camera frame rate in simulation time, even on rendering hardware that is incapable of achieving reliable real-time frame rates. When automatic clock scaling is enabled, FlightGoggles monitors the frame rate of the renderer output and dynamically adjusts the ROS simulation time to achieve the desired nominal frame rate in simulation time. Since the built-in ROS time framework is used, changes in time rate do not affect the relative timing of client nodes, which alleviates non-deterministic timing issues across simulation runs.

Chapter 3

Exteroceptive Sensor Simulation

This section describes the creation of the environment using photogrammetry, lists the features of the render pipeline, and describes each of the exteroceptive sensor models. Much of the work presented in this chapter has been published in International Conference on Intelligent Robots and Systems (IROS) in 2019 [17].

FlightGoggles provides a simulation environment with exceptional visual fidelity. Its high level of photorealism is achieved using 84 unique 3D models captured from real-world objects using photogrammetry, as can be seen in Fig. 3-1. The resulting environment is comprised of over 40 million triangles and 1,050 object instances. To achieve this level of environmental detail, we leverage photogrammetry to model the individual components of the environment, the process of which is explained in detail in the following section.

3.1 Photorealistic Sensor Simulation using Photogrammetry

Photogrammetry is the process in which multiple photographs of a real-world object from different viewpoints are used to efficiently construct a realistic high-resolution 3D model for use in virtual environments. This technique has two major advantages when compared to traditional 3D modeling techniques. Firstly, it requires virtually

(a) Virtual environment in FlightGoggles with barrel (red), rubble (blue), corrugated metal (magenta), and caged tank (green).

(b) Photograph of barrel.

(c) Photograph of rubble.

(d) Photograph of corrugated metal.

(e) Photograph of caged tank.

(f) Rendered image of barrel.

(g) Rendered image of rubble.

(h) Rendered image corrugated metal.

(i) Rendered image of caged tank.

Figure 3-1: Object photographs that were used for photogrammetry, and corresponding rendered assets in FlightGoggles.

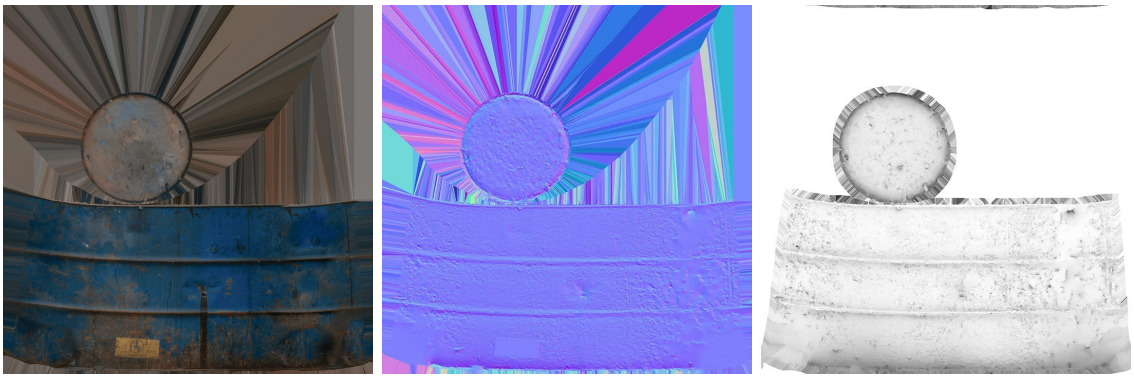
no manual modeling and texturing. The elimination of these time-consuming and artistically demanding processes enables the creation of many high-resolution assets in a relatively short time and at a more moderate cost. Secondly, the resulting renderings are based directly on real-world data, *i.e.*, photographs. Consequently, the simulation includes a photorealistic representation of the real-world object that is being modeled, which may be critical in robotics applications. Due to its advantages over traditional modeling methods, photogrammetry is already widely used in the video game industry; however, its application towards photorealistic robotics simulation, as introduced in FlightGoggles, is novel.

3.1.1 Photogrammetry Asset Capture Pipeline

Photogrammetry was used to create 84 unique open-source 3D assets for the FlightGoggles environment. These assets are based on thousands of high-resolution digital photographs of real-world objects and environmental elements, such as walls and floors. The digital images were first color-balanced, and then combined to reconstruct object meshes using the GPU-based reconstruction software *Reality Capture* [24]. After this step, the raw object meshes were manually cleaned to remove reconstruction artifacts. Mesh baking was performed to generate base color, normal, height and ambient occlusion maps for each object; which are then combined into one high-definition surface material in Unity3D. For a detailed overview of a typical photogrammetry capture workflow, we refer the reader to [25]. Fig. 3-2 shows maps generated using photogrammetry for the scanned barrel object in Fig. 3-1b.

3.1.2 HD Render Pipeline

Fig. 3-1 shows several 3D assets that were generated using the process described above. The figure also shows examples of real-world reference imagery that was used in the photogrammetry process to construct these assets. To achieve photorealistic RGB camera rendering, FlightGoggles uses the Unity Game Engine *High Definition Render Pipeline* (HDRP) [26]. Using HDRP, cameras rendered in FlightGoggles have



(a) Base color map. (b) Normal map. (c) Ambient occlusion map.

Figure 3-2: Texture maps generated using photogrammetry for the scanned barrel asset in Fig. 3-1b.

characteristics similar to those of real-world cameras including motion blur, lens dirt, bloom, real-time reflections, and precomputed ray-traced indirect lighting. Additional camera characteristics such as chromatic aberration, vignetting, lens distortion, and depth of field can be enabled in the simulation environment.

3.1.3 Performance Optimizations and System Requirements

Extensive performance and memory optimizations were performed to ensure that FlightGoggles is able to run on a wide spectrum of GPU rendering hardware with $\geq$ 2GB of video random access memory (VRAM). As can be seen in Table 3.1, FlightGoggles VRAM and GPU computation requirements can be reduced further by user-selectable quality profiles based on three major settings: real-time reflections, maximum object texture resolution, and maximum level of detail (*i.e.* polygon count).

Mesh level of detail

For each object mesh in the environment, three meshes with different levels of detail (LOD), *i.e.*, polygon count and texture resolution, were generated: low, medium, and high. For meshes with lower levels of detail, textures were downsampled using subsampling and subsequent smoothing. During simulation, the real-time render pipeline

	VeryLow2GB	Low2GB	Medium	High (Default)	VeryHigh	Ultra
Mono VRAM Usage	1.45 GB	1.56 GB	1.73 GB	4.28 GB	4.28 GB	4.28 GB
Stereo VRAM Usage	2.00 GB	2.07 GB	2.30 GB	4.97 GB	4.97 GB	4.97 GB
Texture Resolution	1/8	1/4	1/2	1	1	1
Realtime Reflections	-	-	-	✓	✓	✓
Max Level of Detail	Low	Medium	High	High	High	High

Table 3.1: Quality settings for the *Abandoned Factory* environment.

improves render performance by selecting the appropriate level of detail object mesh and texture based on the size of the object mesh in camera image space. Users can also elect to decrease GPU VRAM usage by capping the maximum level of detail to use across all meshes in the environment using the quality settings in Table 3.1.

Pre-baked ray traced lighting

To save run-time computation, all direct and indirect lighting, ambient occlusions, and shadow details from static light sources are pre-baked via NVIDIA RTX raytracing into static lightmaps and are layered onto object meshes in the environment. To precompute the ray traced lighting for each lighting condition in the *Abandoned Warehouse* environment, an NVIDIA Quadro RTX 8000 GPU was used with an average bake time of 45 minutes per lighting arrangement.

Render batching

In order to increase render performance by reducing individual GPU draw calls, FlightGoggles leverages two different methods of render batching according to the capabilities available in the rendering machine. For Windows-based systems supporting DirectX11, FlightGoggles leverages Unity3D's experimental Scriptable Render Pipeline dynamic batcher, which drastically reduces GPU draw calls for all static and dynamic objects in the environment. For Linux and MacOS systems, FlightGoggles statically batches all static meshes in the environment. Static batching drastically in-

Camera Parameter	Default Value
Vertical Field of View (fov)	70°
Image Resolution ($w \times h$)	1024 px × 768 px
Stereo Baseline (t)	32 cm

Table 3.2: Camera sensor parameters enabled by default in FlightGoggles along with their default values.

creases render performance, but also increases VRAM usage in the GPU as all meshes must be combined and preloaded onto the GPU at runtime. To circumvent this issue, FlightGoggles exposes quality settings to the user (see Table 3.1) that can be used to lower VRAM usage for systems with low available VRAM.

3.2 Exteroceptive Sensor Models

FlightGoggles is capable of high-fidelity simulation of various types of exteroceptive sensors, such as RGB-D cameras, time-of-flight distance sensors, and infrared radiation (IR) beacon sensors. Default noise characteristics, and intrinsic and extrinsic parameters are based on real sensor specifications, and can easily be adjusted. Moreover, users can instantiate multiple instances of each sensor type. This capability allows quick prototyping and evaluation of distinct exteroceptive sensor arrangements.

3.2.1 Camera

The default camera model provided by FlightGoggles is a perfect, *i.e.*, distortion-free, camera projection model with optional motion blur, lens dirt, auto-exposure, and bloom. Table 3.2 lists the major camera parameters exposed by default in the FlightGoggles API along with their default values. These parameters can be changed via the FlightGoggles API using ROS param or LCM config. The camera extrinsics T_c^b where b is the vehicle fixed body frame and c is the camera frame can also be changed in real-time.

Figure 3-3: Rendered camera view (faded) with IR marker locations overlaid. The unprocessed measurements and marker IDs from the simulated IR beacon sensor are indicated in red. The measurements are verified by comparison to image-space reprojections of ground-truth IR marker locations, which are indicated in green. Note that IR markers can be arbitrarily placed by the user, including on dynamic objects.

3.2.2 Infrared Beacon Sensor

To facilitate the quick development of guidance, navigation, and control algorithms; an IR beacon sensor model is included. This sensor provides image-space u, v measurements of IR beacons in the camera field of view. The beacons can be placed at static locations in the environment or on moving objects. Using real-time ray-casting from each RGB camera, simulated IR beacon measurements are tested for occlusion before being included in the IR sensor output. Fig. 3-3 shows a visual representation of the sensor output.

3.2.3 Time-of-flight Range Sensor

FlightGoggles is able to simulate (multi-point) time-of-flight range sensors using ray casts in any specified directions. In the default vehicle configuration, a downward-facing single-point range finder for altitude estimation is provided. The noise characteristics of this sensor are similar to the commercially available LightWare SF11/B laser altimeter [27].

Chapter 4

Applications

In this section, we discuss application that FlightGoggles has been used for and potential future applications. Potential applications of FlightGoggles include: human-vehicle interaction, active sensor selection, multi-agent systems, visual inertial navigation research for fast and agile vehicles [15], [18], [19]. The FlightGoggles simulator was used for the simulation part of the AlphaPilot challenge [28].

4.1 Aircraft-in-the-Loop High-Speed Flight using Visual Inertial Odometry

Much of the work presented in this section has been published in The International Robotics and Automation Conference (ICRA) in 2018 [19].

Camera-IMU sensor packages are widely used in both commercial and research applications, because of their relatively low cost and low weight. Particularly in GPS-denied environments, cameras may be essential for effective state estimation. Visual inertial odometry (VIO) algorithms combine camera figures with preintegrated IMU measurements to estimate the vehicle state [19], [29]. While these algorithms are often critical for safe navigation, it is challenging to verify their performance in varying conditions. Environment variables, *e.g.*, lighting and object placement, and camera properties may significantly affect performance, but generally cannot

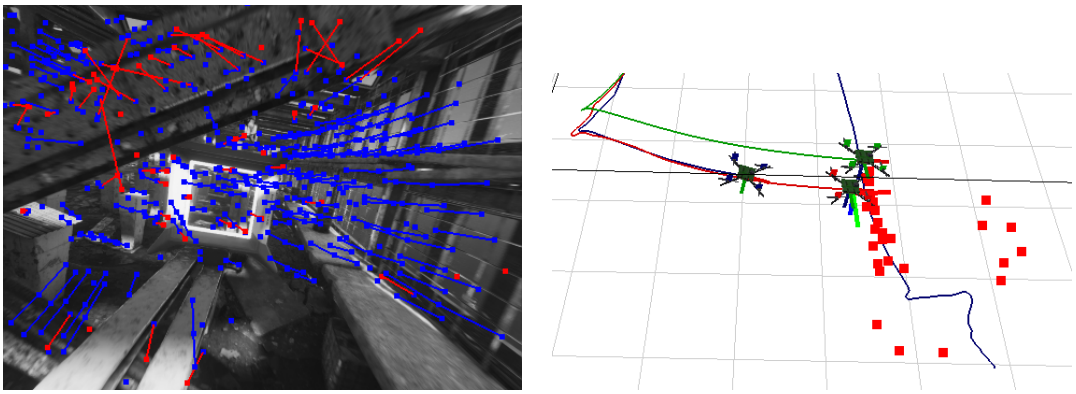


Figure 4-1: The figure on the left shows the visual features tracked using a typical visual inertial odometry pipeline on the simulated camera imagery. On the right, the plot shows three drones, the ground truth trajectory is in green, the high-rate estimate is in red, and the smoothed estimate is in blue. The red squares indicate triangulated features in the environment.

easily be varied in reality. For example, to show robustness in visual simultaneous localization and mapping may require data collected at different times of day or even across seasons [30], [31]. Moreover, obstacle-rich environments may increase the risk of collisions, especially in high-speed flight, further increasing the cost of extensive experiments.

FlightGoggles allows us to change environment and camera parameters and thereby enables us to quickly verify VIO performance over a multitude of scenarios, without the risk of actual collisions. By connecting FlightGoggles to a motion capture room with a real quadrotor in flight, we are able to combine its photo-realistic rendering with true flight dynamics and inertial measurements. This alleviates the necessity of complicated models including unsteady aerodynamics, and the effects of vehicle vibrations on IMU measurements.

Fig. 4-1 gives an overview of a VIO flight in FlightGoggles. The quadrotor uses the trajectory tracking controller described in [32] to track a predefined trajectory that was generated using methods from [33]. State estimation is based entirely on the pose estimate from VIO, which is using the virtual imagery from FlightGoggles and real inertial measurements from the quadrotor. In what follows, we briefly describe a series of experiments using FlightGoggles and an onboard camera which verified that

VIO results from the FlightGoggles aircraft-in-the-loop simulator was comparable to results using a real camera for visual state estimation.

We performed experiments using the FlightGoggles simulator in 2 scenarios to verify the use of a simulator to perform the development of VIO algorithms using real-time exteroceptive camera simulation with aircraft-in-the-loop. For the baseline experiment, we flew the quadrotor through a window without the assistance of a motion capture system first using a on-board camera and then using the simulated imagery from FlightGoggles.

4.1.1 Related Work

Synthetic Environments for Robotics

There has been a variety of work on the use of synthetic data sets and simulation in robotics and more generally computer vision. Synthetically generated data sets, such as those in [12], [34], have become of particular interest as the need for large labeled data sets for deep learning has become prevalent. Of particular note to the work presented here is the method of Richter *et al.* [35] which uses pre-built video games to generate semantically mapped synthetic data sets. Kaneva *et al.* [36] use photorealistic renderings to evaluate the performance of different feature descriptors under a variety camera conditions. Handa *et al.* [14] provide a synthetic data set for the verification of SLAM algorithms against a known 3D model and trajectory. In robotics, Gazebo [8] is the ubiquitous full simulation environment, with specific applications to UAVs in RotorS [10], which is studied in depth in [9]. Of primary relevance to this work is Microsoft Research's recent release of a developing project, AirSim, an Unreal Engine based simulation environment for UAVs [37]. AirSim is a plug-in to Unreal Engine providing a rendered viewpoint of a simulated (or possibly real) UAV location in the Unreal world. Early releases have an eye toward being able to generate large data sets for deep learning based off a simulated UAV model. To the best of our knowledge, the proposed system for UAV development is the first system that allows the UAV computer to use synthetic exteroceptive sensor data along with

real interoceptive sensor data, both streaming in real time, while the UAV is in flight experiencing real physics.

Visual-Inertial Navigation

The literature on visual-inertial navigation is vast, including approaches based on filtering, *e.g.*, [38], [39], fixed-lag smoothing, *e.g.*, [40], [41], and full smoothing [42]–[45]. We refer the reader to the recent survey by Forster *et al.* [44] for a comprehensive review. As the computational power that can be carried on a flying platform has increased, some visual-inertial navigation algorithms have begun to be run in real time on UAVs. Early implementations such as [46]–[48] focused on extending the full SLAM system of Klein and Murray (PTAM) [49] to work on aerial vehicles. Because PTAM was originally designed as a single camera solution for small workspaces, subsequent works primarily focus on application to large workspaces without computational costs growing too high, and using IMU information to correct for the scale drift that is inherent in monocular vision only solutions. More recent approaches have included using a cascading estimate of orientation and position with a low rate stereo camera [50], replacing the PTAM visual SLAM system with the semi-direct approach SVO [51] augmented by an IMU [52], using an off-the-shelf pose estimate from an RGB-D sensor (Google Tango) [53], low energy applications [54], and a factor graph based approach similar to our own [55].

4.1.2 UAV System

A UAV test platform and development environment was built for the testing of on-board estimation and control algorithms while performing agile maneuvers. The UAV is fully controlled by an on-board NVIDIA TX1 module with a modular software framework, enabling rapid testing of new algorithms and sensors.

A single USB 3.0 Point Grey Flea3 monochrome camera with a resolution of 1024x1280 and an external Xsens MTi-3 IMU provide the visual and inertial sensor package for the board. The Point Grey camera uses a Sunex DSL219 fisheye lens; to

avoid the high distortion at the edges of the lens only the center of the image was used for VIO algorithms, leading to an effective resolution of 512x640.

The UAV is controlled through an on-board software setup that provides complete end-to-end operation of the UAV from raw sensor data to the signals sent to each ESC. The system uses Lightweight Communications and Marshaling (LCM) [21] for communication between on-board modules, giving a lightweight and flexible framework. All processing occurs on-board the CPU and GPU of the TX1.

4.1.3 Experiments

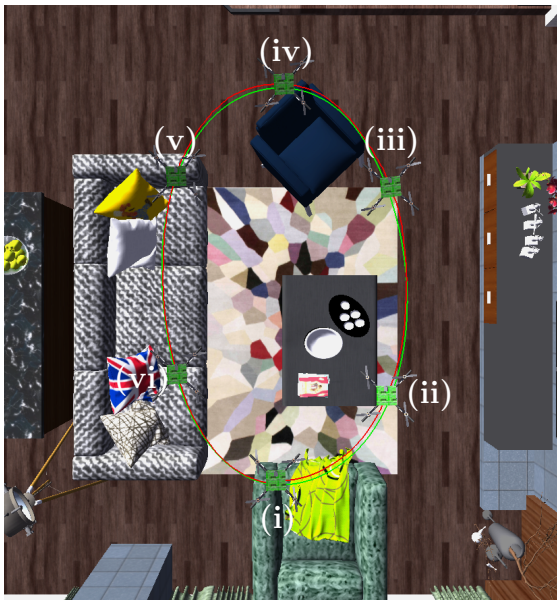
Experimental Setup. Experiments were performed in an approximately rectangular 6 m x 4 m environment. A set of 6 OptiTrack Prime 17W cameras provide a ground truth pose estimate in the enclosed area, running at 120Hz, which is used for photorealistic camera image generation. Three sets of experiments were performed:

1. Visual state estimation and control in a baseline scenario involving an indoor environment

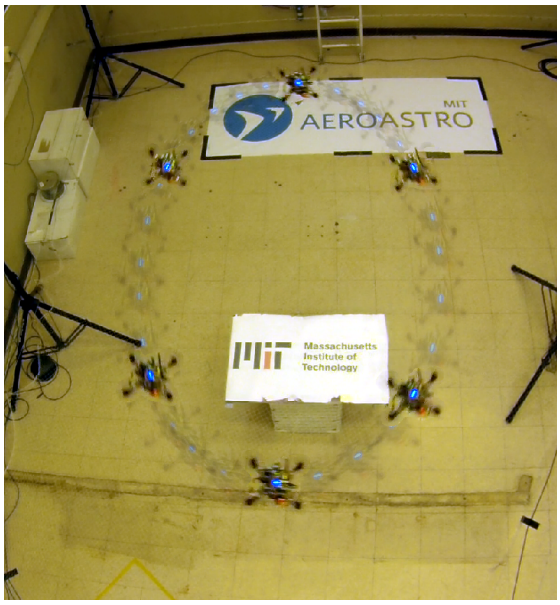
2. Visual state estimation and control in a challenging scenario involving flying through a window

3. Camera parameter sweep to investigate estimation accuracy for various camera parameters

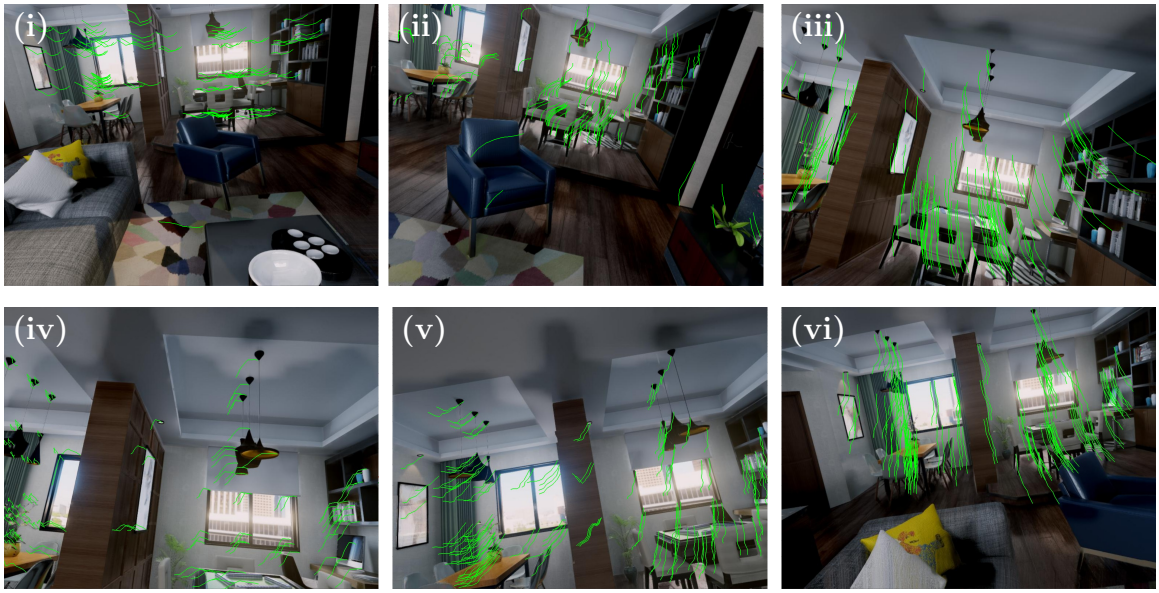
The first two experiments were conducted both in simulated environments (using FlightGoggles) and in real environments, whereas the last experiment was performed in simulated environments. Each experiment has two phases; first, a take-off phase where the UAV flies under motion capture using position references provided by the operator, and second; the experimental phase where the UAV flies with the VIO state estimate in the loop and executes a predefined periodic pattern until a low battery warning occurs. The take-off period serves to both stabilize the VIO state estimate with visual features (a natural restriction of a monocular method) and to initialize the integrator for the controller. Due to the small available flight space and long flight



(a) Trajectory from a single oval of VIO-in-the-loop flight overlaid on the environment 3D model. The green line shows the position of the drone as recorded from motion capture while the red line shows the VIO state estimate of position used for control.



(b) Top view of the drone flying in the lab during the experiment, the pose of the drone for the six images images displayed below is emphasized.



(c) Six photorealistic images generated and streamed to the drone in real time for VIO state estimation during a VIO experiment around an oval trajectory. Green lines show 0.3 seconds of history for visual features detected and tracked by the vision front end.

Figure 4-2: Visualization of a VIO experiment using simulated imagery showing the (a) true and estimated trajectories of the drone, (b) top view of the drone's flight, and (c) six images generated during flight with the tracked features shown.

times the UAV with eventually drift into a wall if given a fixed reference trajectory in the global coordinate frame. To keep the UAV within the flight cage drift is corrected by shifting the global desired trajectory to match the visual-inertial odometry (VIO) local frame after each loop. This mimics the behavior that would occur if the UAV were generating trajectories based on its available local map to navigate a room.

Visual Navigation in Open Space. In total 42 experiments were performed in open space, 21 using the on-board camera on the UAV and 21 using our photorealistic image generation system to simulate a camera in real time. In all 21 experiments using the on-board camera and in 19 out of 21 experiments using the simulated camera the UAV traced out the desired trajectory with the VIO state estimate in the control loop for the full life of the battery (2-3 minutes). In the two simulated experiments that had to be ended early, Wifi network dropouts caused visual data to not reach the UAV, and the experiment was ended for safety. The reference trajectory flown for these experiments is an oval of length 2.8 m and width 1.6 m, with a period of $3.5 - 3.8$ sec, for an average speed of $\sim 2\,\mathrm{m\,s^{-1}}$ and a maximum speed of $\sim 3\,\mathrm{m\,s^{-1}}$ on the long sides of the oval.

The estimation error as a function of distance traveled for all 42 experiments is shown in Fig. 4-3. Since this system has no loop closures the initial estimation error during take-off cannot be recovered, resulting in the higher error percentages at the beginning of the flight when little distance has been traveled. Once the UAV starts flying its trajectory the estimation error remains below 1% (1 cm error for every 1 m flown) in all experiments. Note that the tracking of features is intentionally limited to 3 seconds, both to maintain low computation costs and to better mimic flying through an ever changing environment where no features can be seen continuously. The VIO state estimate was continuously in the control loop without assistance from motion capture for all 42 flights, demonstrating a stable and accurate state estimate.

Visual Navigation through a Window. Our second set of experiments involves flying through a window (0.90 m x 0.60 m, approximately twice the size of the UAV) with the VIO state estimate in the loop. Flying through windows presents a challenging problem for monocular VIO systems with forward facing cameras, as

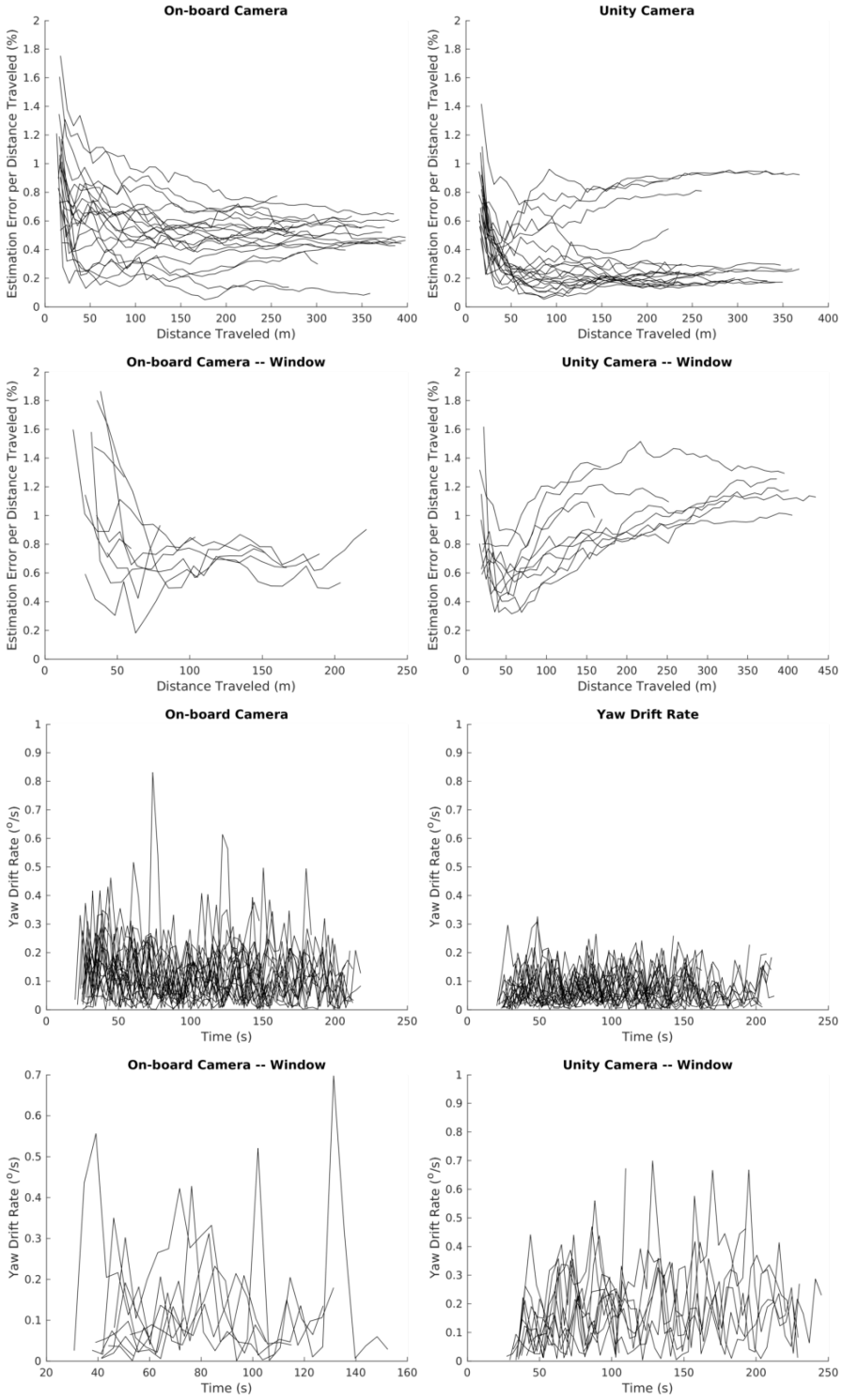


Figure 4-3: Error in VIO state estimate of the UAV's position as a percentage of the distance traveled by the UAV, and rate of error in the VIO estimate of yaw. Flights flown with the on-board camera are shown on the left while flights flown using camera images rendered in Unity are shown on the right; a total of 21 flights were flown with each style of camera without a window, and 8 and 10 flights with the real and simulated cameras through a window.

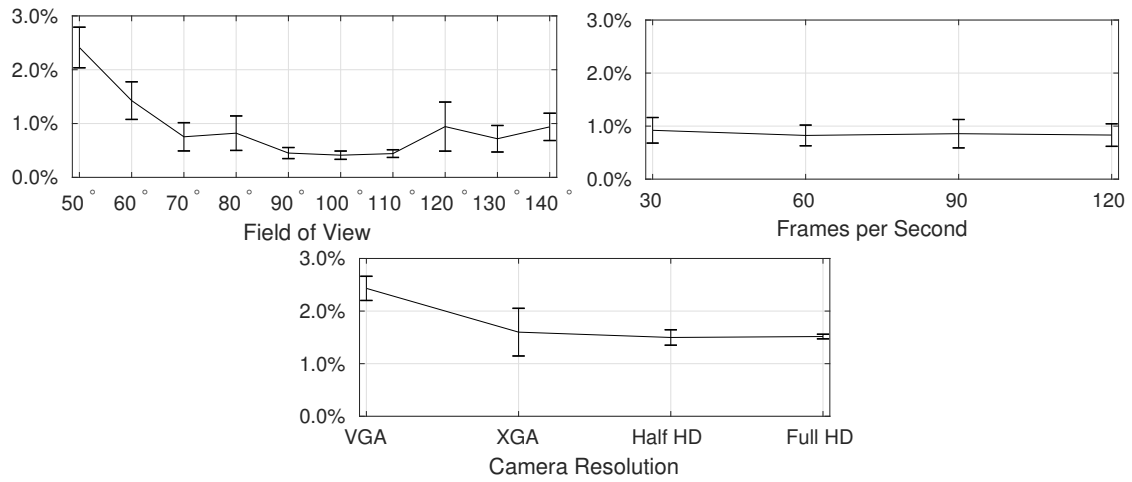
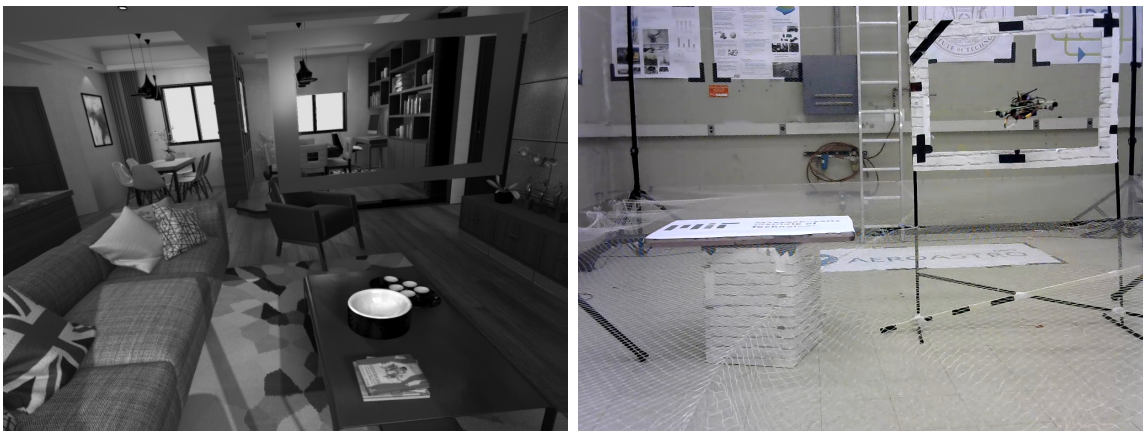


Figure 4-4: VIO estimation error per distance traveled (%) using FlightGoggles with prerecorded IMU data across multiple camera parameters. Eight trials were performed for each sensor type. All trials were run in real-time using our simulation pipeline and an attached Jetson TX1. FOV trials were conducted with XGA (1024x768) resolution at 60 FPS. FPS trials were run at VGA (640x480) resolution and 80° field of view. Camera resolution trials were conducted at 50 FPS and 60° FOV.



(a) Unity generated image with window to fly through in the upper right corner.

(b) Image of UAV flying through a physical window using VIO in the loop based on its on-board camera and IMU.

Figure 4-5: Images from VIO experiments, showing an image from a virtual on-board camera (top) and of our UAV flying through a window gap under VIO control (bottom)

the visual element of the VIO system relies on motion to triangulate features. When flying through a window the only visual data linking state estimates on one side of the window to the other are those seen through the window, for which there is little tangential motion making triangulation inaccurate.

To the best of our knowledge the only two demonstrations of on-board, vision-based navigation through window openings come from Loianno *et al.* [56] and Falanga *et al.* [57]. In both cases the focus was on trajectory generation and control under uncertainty, and state estimation was only maintained for a single traversal of the window before landing. A key concern is the ability to continue flight after passing through a window; therefore we sought to repeat the baseline experiment with a window in the path of the oval trajectory, although at a slower speed (average speeds of $\sim 1.7\,\mathrm{m\,s^{-1}}$ and maximum speed of $\sim 2.3\,\mathrm{m\,s^{-1}}$). At each loop a simulated window detection occurred to set a new flight trajectory through the window, however this window detection was not used for state estimation.

Our photorealistic sensor simulation system provides the platform to develop our algorithms for window navigation. Developing algorithms with a physical window hazards numerous crashes as the system is developed, and the development of the system without a window or without visual navigation algorithms in the loop does not provide an accurate description of the performance of the UAV powered by visual navigation algorithms. Instead, our development environment allows for the visual effect of flying through a window, real dynamics, and real inertial measurements, without crashing on failure.

A total of 10 flights were performed with a simulated camera, constituting 361 traversals of the window, with 3 traversals resulting in a "crash" with the virtual window (crashes detected from the motion capture position of the UAV). The estimation error across the flights is shown in cyan in Fig. 4-3.

Through experimentation with simulated imagery we found that a high keyframe rate with less feature data is necessary to both consistently bridge the gap created by flying through the window, and to quickly re-establish an accurate state estimate on the other side of the window. Based on these lessons, we performed the same

experiment with a real window and the on-board camera. A total of 8 flights were performed, constituting 119 traversals of the window with 6 crashes/pilot take overs due to estimation divergence. Based on the analyzed data the lower success rate when using a real camera was due to a combination of noisier visual data than provided by the simulation system, and the additional computational load on the on-board computer of image acquisition slowing down the optimization rate. The noisier visual data can come from a combination of lower quality features in the world, motion blur of the live camera, and imperfections in the estimated camera model.

Camera Parameter Tests. In addition to allowing for testing in a variety of visual environments, our development environment also allows for rapidly evaluating sensor properties and configurations. For instance, we took a 70 second pre-recorded flight of our UAV flying an oval trajectory under motion capture and tested the VIO's performance in real-time using real-world IMU measurements against a set of camera parameters spanning Field of View, Camera Resolution, and Frame Rate. See Fig. 4-4 for a selection of results. These measurements are not meant as a declaration of the best camera to use for visual-inertial navigation, but rather to show the capabilities of the system for rapid system prototyping to fit new challenges. While we have focused on a few parameters of the camera sensor itself, a wide range of other effects such as camera blur, scene lighting, feature richness, and accuracy of the camera model can easily be investigated.

4.2 Perception-aware Planning

Much of the work presented in this section has been published in The American Control Conference (ACC) in 2019 [18].

During the performance of agile maneuvers by a quadrotor, visually salient features in the environment are often lost due to the limited field of view of the on-board camera. This can significantly degrade the estimation accuracy. To address this issue, we present an approach to incorporate the perception objective of keeping salient features in view in the quadrotor trajectory planning problem. The FlightGoggles

simulation environment was used to perform experiments. It allowed rapid experimentation with feature-rich objects in varying amounts and locations. This enabled straightforward verification of the performance of the algorithm when most of the salient features are clustered in small regions of the environment. The experiments show that significant gains in estimation performance can be achieved by using the proposed vision aware planning algorithm as the speed of the quadrotor is increased.

A robust state estimator is often critical in performing various agile maneuvers and loss of tracked features can result in collisions. Longer feature tracks often result in more robust visual inertial state estimation performance. Motion planning algorithms agnostic to the location of the visual features in the environment potentially result in trajectories for which the vehicle may face featureless areas of the environment (*e.g.*, empty walls). This leads to diminished state estimation accuracy, which degrades the performance of trajectory tracking, potentially resulting in catastrophic failure, such as a collision. *Perception-aware motion planning algorithms* consider trajectories that ensure better observation of visual features, improving state estimation accuracy and enabling faster navigation in complex environments (as illustrated in Figure 4-7).

This experiment focuses on perception-aware trajectories for quadrotor aircraft which pass through a finite sequence of predetermined *waypoints*. The perception objective consists of observing a sparse set of triangulated landmarks from multiple keyframes in order to improve both the estimate of the position of the landmarks and the state of the aircraft itself. The perceptual sensor yields information about the relative transformation between consecutive keyframes, which motivates the *co-visibility* constraint added to the optimal control problem. Nevertheless, this allows for significant freedom in specifying both the path and the orientation of the aircraft between the waypoints in order to optimize an objective of interest.

4.2.1 Related Work

Traditional methods for generating quadrotor trajectories exploit the differential flatness properties [58] of the dynamics of the quadrotor. The flat outputs, the trajectories of the x, y, z, and ψ components of the state of the quadrotor, can be specified

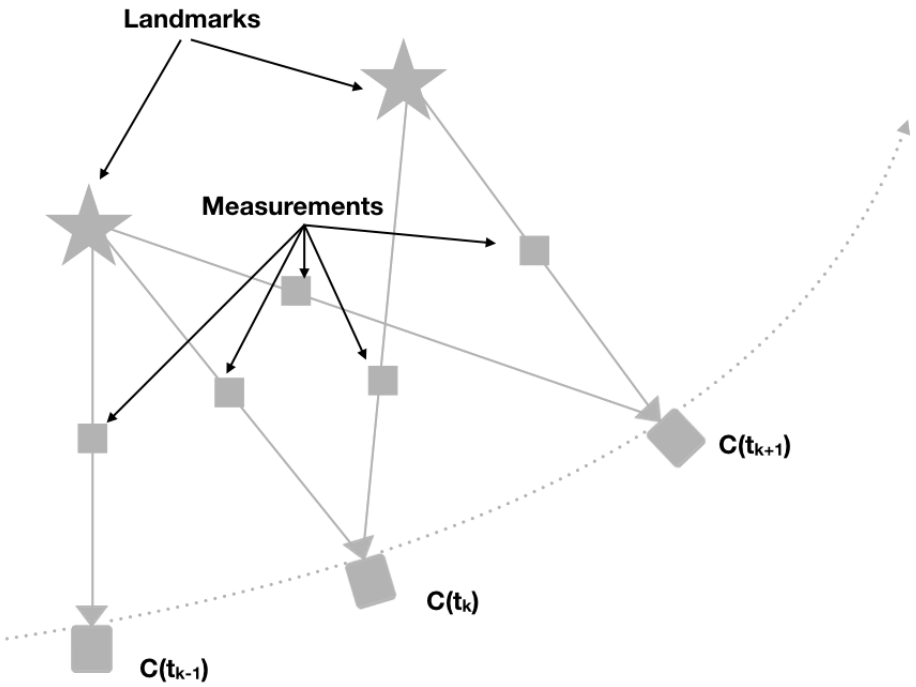


Figure 4-6: Aspects of visual-inertial navigation. Observations of landmarks are acquired at discrete time intervals over a continuous trajectory.

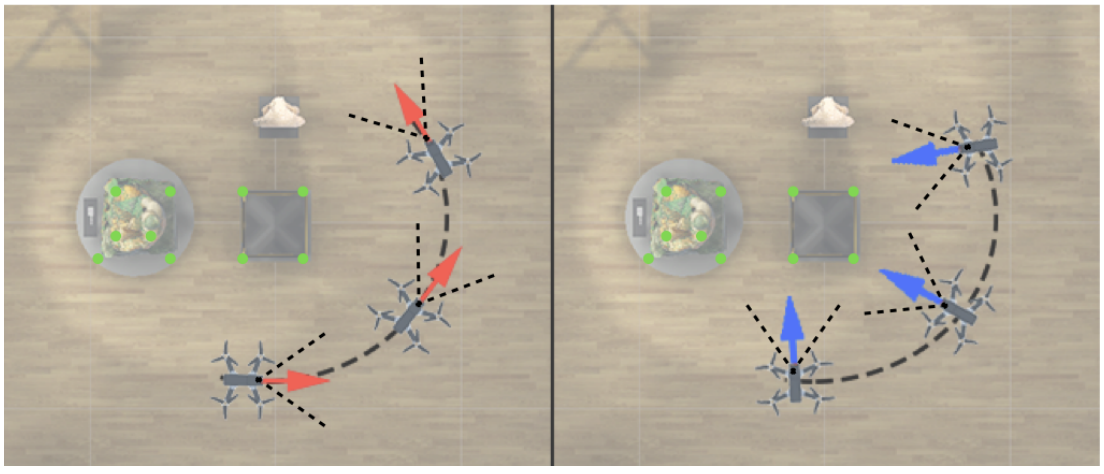


Figure 4-7: An environment where most of the visual features are situated in the middle. Traditional planning approaches could consider facing forward as the vehicle navigates through a tight turn. Perception-aware planning algorithms, on the other hand, consider facing towards the visual features in the middle of the room while executing the same maneuver. This results in improved state-estimation performance, especially at high speeds.

independently, and furthermore, they uniquely determine the trajectories of the remaining components of the state as well as the required control inputs. Mellinger and Kumar [58] first proposed the idea of parametrizing flat outputs as polynomials in time, selecting coefficients that minimize the snap of the quadrotor. Richter *et al.* [33] proposed an extension to this work that bounds the actuator effort and adds a time penalty to the cost function to encourage fast flight. Other approaches, *e.g.*, proposed by Mueller *et al.* [59], make the assumption that the yaw of the quadrotor is constant during the execution of the trajectory and formulate the polynomial optimization problem in terms of minimum jerk.

In addition to planning dynamically feasible trajectories, autonomy entails accurate on-board state estimation. The difficulty of the latter task may vary significantly based on the choice of the trajectory. Prior work has considered planning to minimize various measures of state-estimation uncertainty with the goal of satisfying a chance constraint. The problem under consideration is to plan a path in the belief space that ensures the probability of collision with the environment is lower than a set threshold. Bry and Roy [60] propose a sampling-based motion planning algorithm that generates a rapidly exploring random tree where the associated cost function accounts for both the cost of executing the path and the uncertainty associated with the action. The algorithm proposed by van den Van Den Berg *et al.* [61] accounts for uncertainty during the planning phase through forward propagation of the covariance matrix over paths planned using a RRT based planner. They use this to enumerate the paths and select the one with the lowest cost.

Bry and Roy [60] and Van Den Berg *et al.* [61] solve problems involving chance constraints and require full knowledge of obstacle locations in the environment in addition to the locations of informative landmarks: beacons, visual landmarks, etc. The chance constraint can impose conditions on the full state of the robot under consideration, and these methods plan trajectories in the full state space of the robot. For instance as mentioned previously, quadrotor trajectory planning often takes place in the differentially flat space. However, to satisfy the chance constraint, these approaches would require either mapping the obstacles and landmarks into this space

or a projection of the differentially flat space back into the full state space to check the chance constraint during planning.

Approaches that specifically target visual-inertial navigation have been proposed. In [62], the authors propose a trajectory optimization method that first solves for a goal location given a target image and then plans the trajectory for which the reprojection error of the desired features is minimized during execution of the trajectory. In [63], the authors use a B-spline polynomial parameterization for the trajectory, and solve a minimum-time trajectory optimization problem with the objective of always keeping a certain set of features in the field of view. These methods are motivated by keeping a specified set of features in view. As such, they do not provide a flexible framework for allowing the algorithm to keep a smaller set of features in view to balance the perceptual constraints and the desired average speed while ensuring the actuator constraints are not violated. The set of possible trajectories that can be achieved using this constraint is often restrictive and not suitable to applications where aggression of the quadrotor is preferred over target tracking.

Falanga *et al.* [64] present an approach that adds a perception objective into their model predictive control framework. They model the cost as minimizing the velocity of a point of interest in the camera frame. They consider keeping a cluster of features within the field of view. While this holds when the visible features in the environment are clustered together at every position along the trajectory, there is room for improvement when the features are more dispersed. We address two main remaining challenges, (1) Enforcing the constraint that a significant number of features are visible in consecutive keyframes instead of enforcing that a certain set of features stay in view, (2) A computationally-efficient formulation to solve this problem while generating the reference trajectory to follow.

Passive methods for keeping features in view have also been studied. Attention and anticipation [65] has been studied for visual feature selection where the goal is to select features that will remain in the field of view without adding any actuator effort to the quadrotor. The authors of [65] use an information gain metric for a greedy algorithm to select features that will add the most information to the state

estimate by forward simulating the dynamics of the quadrotor. However, this method separates perception from the planning and does not actuate the quadrotor to account for observing features.

4.2.2 Experiments

We fly generated reference trajectories using a custom built quadrotor platform in an 11 m x 11 m x 5.5 m motion capture room, with 24 OptiTrack Prime 17W cameras providing the ground truth position of the quadrotor. For the purpose of state estimation, there is an Xsens MTi-3 IMU onboard the quadrotor. Visual data is generated in real-time using the FlightGoggles [19] photo-realistic image generation system. The visual inertial odometry system fuses the information from the IMU and camera to generate an estimate of the state of the quadrotor.

We design a trajectory to fly in the environment shown in Figure 4-8. The room has been emptied to demonstrate the effectiveness of the approach. The walls are textureless and thus devoid of visual landmarks but the center of the room has pillars and statues that are landmark-rich.

The reference trajectory generated by our proposed method is shown in Figure 4-9.

In this section, we assume locations of the landmarks in the environment are known *a priori*. This is typical in some autonomous quadrotor aircraft scenarios such as drone racing, industrial warehouses, and more general scenarios where the environment has been mapped beforehand and the topological visual map of the environment is known. In practice, the landmark locations used could be replaced by accumulating triangulated landmarks from visual inertial odometry over a short window. For the purpose of experimentation, we relax this constraint.

Since we would like to optimize the polynomials over segments defined by keyframes, we want the optimization to be performed at keyframe rate at the very least. In practice, we use a keyframe rate of 10 Hz and indeed our implementation computes the polynomials in less than 0.1 sec. Our timing experiments were run on a 10 core Intel® i9-7900X CPU with 32GB of RAM. As mentioned previously, it is possible to write

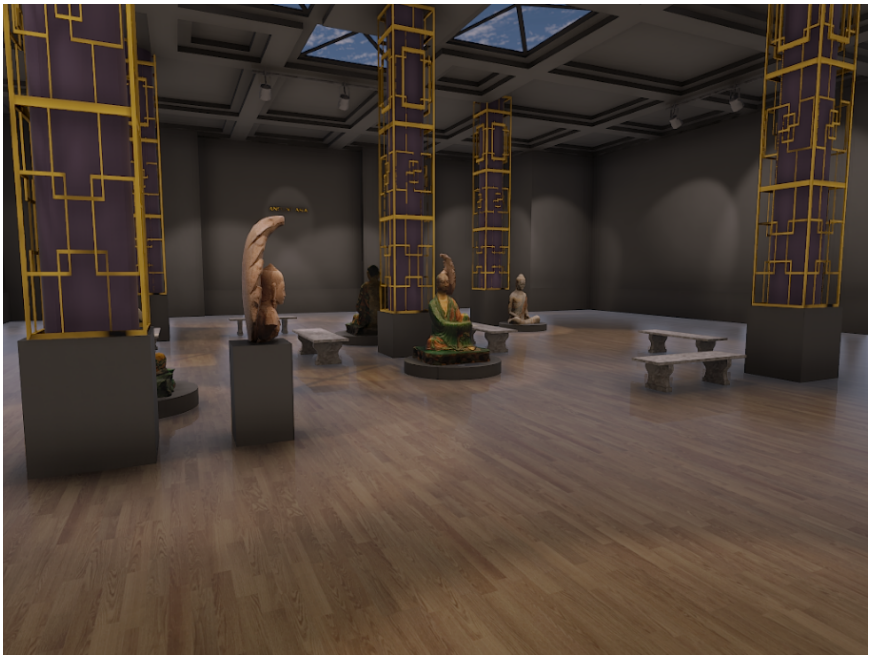


Figure 4-8: The figure shows the environment that is used for the experiments. This environment has no landmarks on the walls and all the objects are in the center of the room.

an efficient implementation that uses a GPU which would fit on the Tegra TX2 which is currently onboard our quadrotor aircraft platform.

In the rest of this section, we present two experiments. In the first experiment, the state estimate for closing the control loop is provided by the motion capture system, whereas in the second experiment, the state estimation is provided by visual inertial odometry. We compare the effect on the performance of the visual front-end and the tracking error of the controller with respect to the reference trajectory and our optimized trajectory. The reference trajectory follows a forward facing objective for the yaw. The parameters for the system are kept constant across both of the experiments.

Experiment 1 (Mocap In-the-loop)

For the first experiment the quadrotor is commanded to perform the trajectory at various top speeds ($1.7\,\mathrm{m\,s^{-1}}, 2.7\,\mathrm{m\,s^{-1}}, 3.4\,\mathrm{m\,s^{-1}}$) with the state estimation from the motion capture system being used to close the control loop. The reference trajectory

	Forward Facing	Optimized Yaw
$1.7\,\mathrm{m\,s^{-1}}$	134.1836	180.0496
$2.7\,\mathrm{m\,s^{-1}}$	126.7671	179.1885
$3.4\,\mathrm{m\,s^{-1}}$	122.4460	179.1367

Table 4.1: The average number of tracked features between keyframes in Experiment 1 is shown above.

that is generated by applying our approach is shown in Figure 4-9.

This trajectory is controlled using a nonlinear dynamics inversion based controller as described in [66]. In the figure, the arrows represent the heading angle of the quadrotor. As can be seen in the figure, the quadrotor tries to keep the yaw angle pointed towards the feature rich part of the environment which is the expected behavior.

In this experiment, the average number of tracked landmarks between consecutive keyframes is measured and compared between a forward facing trajectory and the optimized yaw trajectory. The parameters for both the state estimation and the controller are kept the same across all speeds. The results of this experiment is shown in Table 4.1. As can be seen in the table, a significantly larger number of landmarks is tracked across different speeds, which shows that the perception objectives are maximized by our proposed method. Since the number of landmarks tracked across keyframes is significantly larger, this directly validates that our approach is able to generate trajectories that can maintain co-visibility of a large number of landmarks between keyframes.

Experiment 2 (VIO In-the-loop)

In this experiment, the quadrotor is commanded to fly the same trajectory three times with top speeds of $1.7\,\mathrm{m\,s^{-1}}$ and $2.7\,\mathrm{m\,s^{-1}}$ with the state estimate to close the control loop provided by visual inertial odometry for both the forward facing yaw and the optimized yaw. The parameters are again kept the same for every trial. The qualitative comparison is shown in Figure 4-10. As can be seen from the figure, the quadrotor fails to follow the trajectory while attempting to face forward, but is

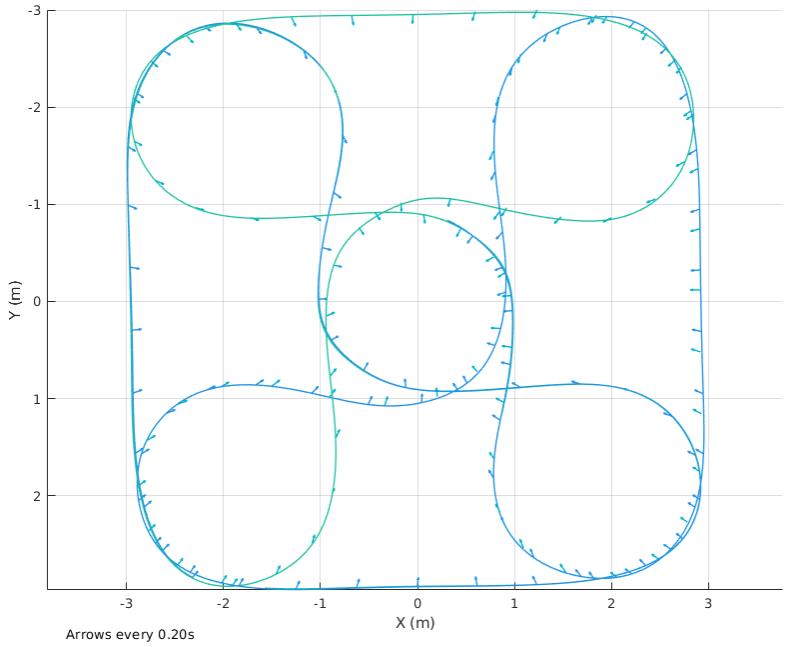


Figure 4-9: The figure shows the optimized trajectory generated by the proposed algorithm after taking the perceptual constraints into account. As can be seen in the figure, the quadrotor prefers facing towards the objects in the environment and avoids facing the empty walls.

successful when the optimized reference trajectory is used. It is important to note here that the state estimation error drifts because the walls are featureless and the poor state estimation results in unsuccessful tracking.

The quantitative results for this evaluation are shown in Table 4.2. The metric we use to measure the tracking performance of the controller with the reference trajectories is the absolute trajectory error statistics. These are the average root-mean-square error (RMSE), mean, and median of the trajectory errors over the three separate trials for each instance of the forward facing and the optimized yaw trajectory. As can be seen in the table, there is a significant improvement in the absolute trajectory error statistics for the optimized yaw trajectory over the forward facing behavior. As noted in the qualitative comparison, this can be attributed to having a better state estimate due to a larger number of constraints between keyframes.

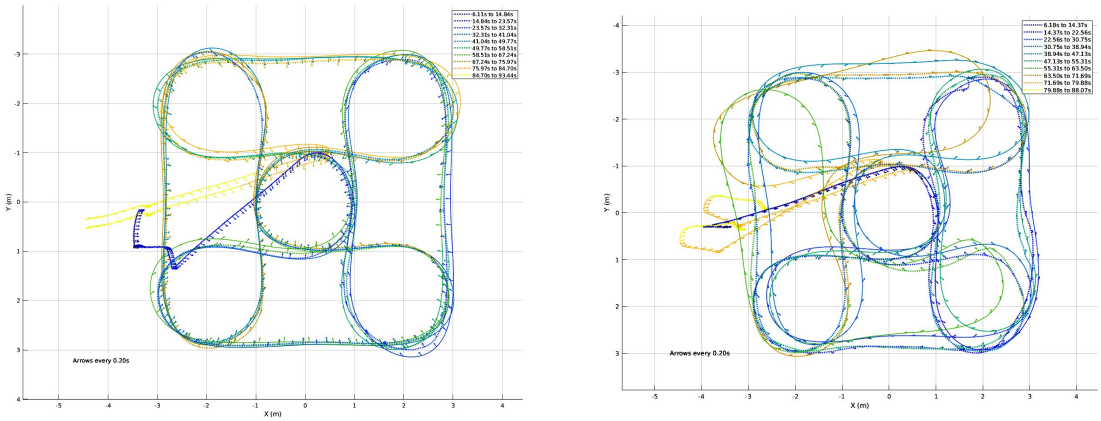


Figure 4-10: The Figure above shows the qualitative comparison between the quadrotor following the reference trajectory at $2.7\,\mathrm{m\,s^{-1}}$ closed by visual inertial odometry. On the left, the trajectory flown with optimized yaw is shown. On the right, the trajectory flown with a forward facing yaw is shown.

		Forward Facing	Optimized Yaw
$1.7\,\mathrm{m\,s^{-1}}$	Mean	$0.3675\,\mathrm{m}$	$0.1854\,\mathrm{m}$
	Median	$0.3234\,\mathrm{m}$	$0.1987\,\mathrm{m}$
	RMSE	$0.4838\,\mathrm{m}$	$0.2246\,\mathrm{m}$
$2.7\,\mathrm{m\,s^{-1}}$	Mean	$0.5152\,\mathrm{m}$	$0.1549\,\mathrm{m}$
	Median	$0.5185\,\mathrm{m}$	$0.1625\,\mathrm{m}$
	RMSE	$0.5958\,\mathrm{m}$	$0.1802\,\mathrm{m}$

Table 4.2: The average absolute trajectory error statistics over 3 trials of the trajectory are shown in the above table.

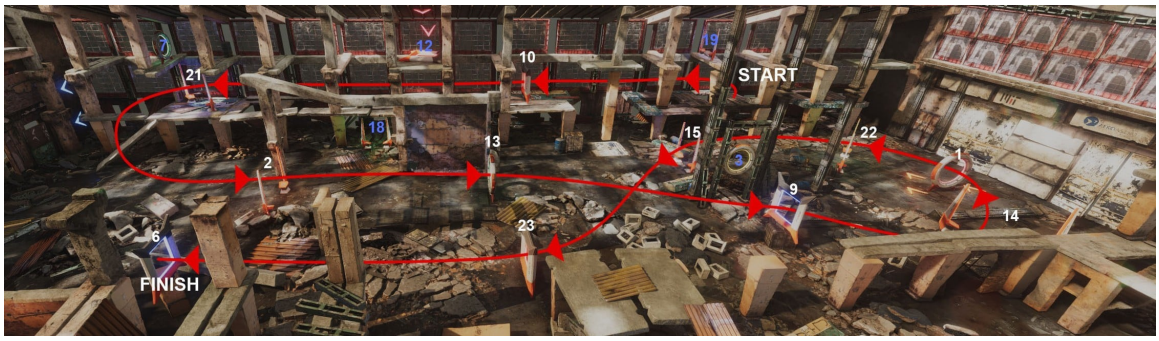


Figure 4-11: Racecourse layout for the AlphaPilot simulation challenge. Gates along the racecourse have unique IDs labeled in white. Gate IDs in blue are static and not part of the race. The racecourse has 11 gates, with a total length of ∼240m.

4.3 AlphaPilot Challenge

Much of the work presented in this section has been published in The International Conference on Intelligent Robots and Systems (IROS) in 2019 [17].

The AlphaPilot challenge [28] is an autonomous drone racing challenge organized by Lockheed Martin, NVIDIA, and the Drone Racing League (DRL). The challenge is split into two stages. A simulation phase open to the general public, and a real-world phase in which teams compete against each other by programming fully-autonomous racing drones built by DRL. During the simulation phase, the FlightGoggles simulation framework was used as the main qualifying test for selecting nine teams that would progress to the next stage of the AlphaPilot challenge. To complete the test, contestants had to submit code to autonomously race a simulated quadrotor with simulated sensors through the 11-gate race track shown in Fig. 4-12. Test details were revealed to all contestants on February 14th, 2019 and final submissions were due on March 20th, 2019. This section describes the AlphaPilot qualifying test and provides an analysis of anonymized submissions.

4.3.1 Challenge Outline

The purpose of the AlphaPilot simulation challenge was for teams to demonstrate their autonomous guidance, navigation, and control capability in a realistic simulation environment. The participants' aim was to complete the track as fast as possible

using a simulated quadrotor based on the FlightGoggles multicopter dynamics model. To accomplish this, measurements from four simulated sensors were provided: (stereo) cameras, IMU, downward-facing time-of-flight range sensor, and infrared gate beacons. Through the FlightGoggles ROS API, autonomous systems could obtain sensor measurements and provide collective thrust and attitude rate inputs to the quadrotor low-level acro/rate mode controller.

The race track was located in the FlightGoggles *Abandoned Factory* environment and consisted of 11 gates. To successfully complete the entire track, the quadrotor had to pass through all the gates in order. The final score was calculated as $score = 10 \cdot gates - time$ where $gates$ is the number of gates passed in order and $time$ is the time taken in seconds to reach the final gate. If the final gate was not reached within the race time limit or the quadrotor collided with an environment object, a score of zero was recorded. To discourage memorization of the course, the exact gate locations were subject to random unknown perturbations. These perturbations were large enough to require adapting the vehicle trajectory, but did not change the track layout in a fundamental way. The final score for each team was the average of their five highest scores over an evaluation set of 25 perturbed courses that was kept unknown to the teams. For development and verification of their algorithms, participants were provided with the nominal gate locations, as well as another set of 25 perturbed courses with identically distributed gate locations.

4.3.2 Survey of AlphaPilot Simulation Challenge Results

FlightGoggles Sensor Usage

Table 4.3 shows the usage of provided sensors, the algorithm choices, and final and five highest scores for the 20 top teams (sorted by final score). All of these 20 teams used both the simulated IMU sensor and the infrared beacon sensors. Several teams chose to also incorporate the camera and the time-of-flight range sensor. A more detailed overview of the sensor combinations used by the teams is shown in Table 4.4. This table shows the number of teams that employed a particular combination of

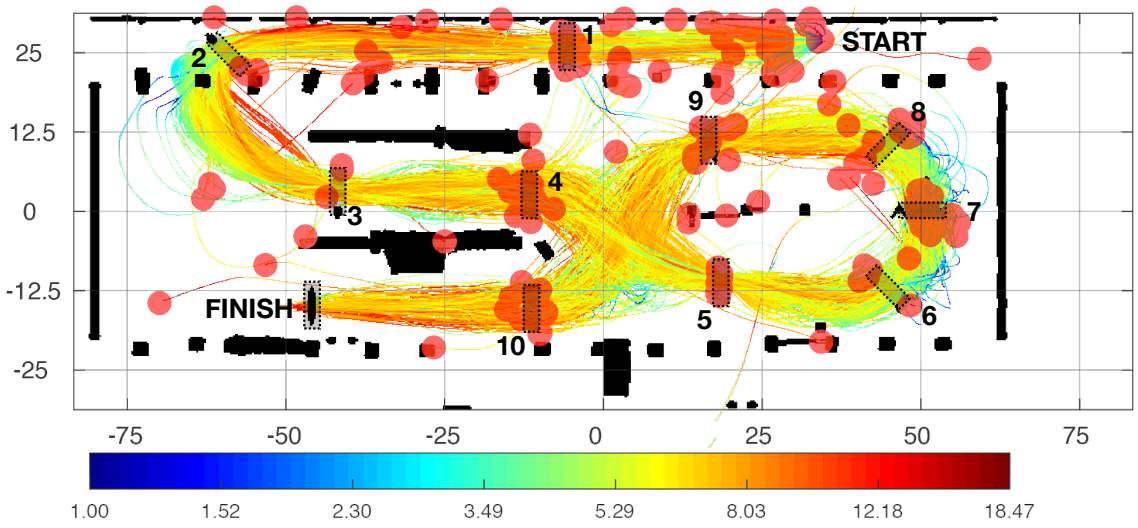


Figure 4-12: Overhead visualization of speed profiles (in ms^{-1}) and crash locations for top 20 AlphaPilot teams across all 25 runs. Nominal gate locations are numbered in track order and marked with boxes. Note that most crashes occur near gates, obstacles, or immediately after takeoff.

sensors, the percentage of runs completed, and the mean and standard deviation of the scores across all 25 attempts.

Algorithm Choices

The contestants were tasked with developing guidance, navigation, and control algorithms. Table 4.3 tabulates the general estimation, planning, and control approaches used for each team alongside the sensor choices and their scores.

Of the top 20 teams, only one used an end-to-end learning-based method. The other 19 teams relied on more traditional pipelines (estimation, planning, and control) to complete the challenge. One of those teams used learning to determine the pose of the camera from the image.

Estimation: For state estimation, all but one team used a filtering algorithm such as the extended Kalman filter [67], unscented Kalman filter [68], particle filter [69], or the Madgwick filter [70] with the other team using a smoothing based technique [71]. The teams that chose to use a visual inertial odometry algorithm opted to use off-the-shelf solutions such as ROVIO [72], [73] or VINS-Mono [74] for state estimation.

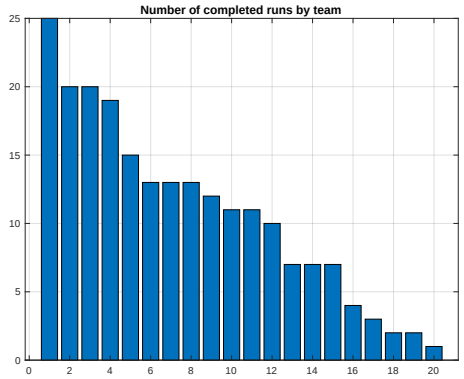
Planning: The most common methods used for planning involved visual servo

Camera	IMU	Ranger	Infrared	Learning	VIO	Filter	Smoother	Polynomial	Visual Servo	Other	Linear	MPC	Other	Final Score	Score 1	Score 2	Score 3	Score 4	Score 5
	✓		✓			✓				✓		✓		91.39	91.52	91.50	91.38	91.32	91.24
	✓		✓				✓	✓			✓		✓	84.52	85.35	84.62	84.33	84.19	84.10
	✓		✓			✓		✓				✓		81.04	81.47	81.05	80.94	80.92	80.85
	✓		✓			✓			✓		✓			80.56	80.99	80.86	80.40	80.29	80.26
✓	✓		✓		✓	✓				✓		✓		78.61	78.78	78.65	78.56	78.55	78.53
	✓		✓			✓		✓					✓	78.55	78.69	78.59	78.50	78.50	78.48
✓	✓		✓		✓	✓						✓		76.08	76.60	76.17	75.95	75.90	75.77
	✓		✓			✓		✓			✓			74.23	74.51	74.17	74.17	74.14	74.13
✓	✓		✓		✓	✓			✓		✓			71.44	71.50	71.47	71.45	71.44	71.36
	✓		✓			✓		✓					✓	71.10	71.28	71.10	71.07	71.02	71.00
	✓	✓	✓			✓			✓		✓			70.87	73.58	72.80	72.78	72.70	62.50
✓	✓	✓	✓	✓		✓				✓	✓			70.46	71.03	70.79	70.22	70.21	70.03
	✓		✓			✓				✓	✓			69.91	71.42	70.70	69.29	69.17	68.96
	✓	✓	✓	✓										57.26	76.96	76.35	66.54	66.48	0.00
	✓		✓			✓		✓			✓			56.28	56.48	56.36	56.21	56.17	56.16
	✓	✓	✓			✓			✓		✓			55.88	57.50	56.15	55.71	55.29	54.73
✓	✓		✓		✓	✓		✓			✓			29.84	74.76	74.46	0.00	0.00	0.00
	✓		✓			✓			✓		✓			12.99	25.19	19.89	19.85	0.00	0.00
	✓		✓			✓			✓		✓			12.58	41.05	21.84	0.00	0.00	0.00
	✓		✓			✓			✓		✓			11.81	59.07	0.00	0.00	0.00	0.00

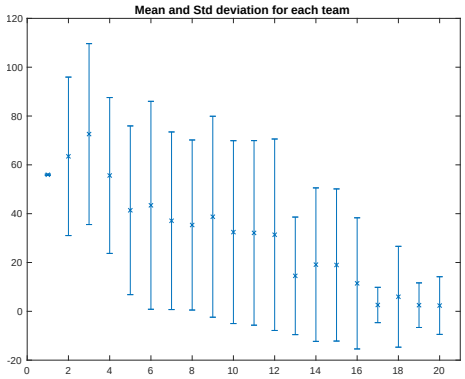
Table 4.3: Sensor usage, algorithm choices, and final and five highest scores in AlphaPilot simulation challenge.

Sensor Package Selection	Number of Teams	Completed Runs (%)	Mean Score	Std. Dev.
IMU + IR	12	48.67	35.32	37.72
IMU + IR + Camera	4	36	26.72	35.87
IMU + IR + Ranger	3	24	15.04	27.43
IMU + IR + Ranger + Camera	1	60	41.39	34.55

Table 4.4: Sensor combinations used by the top AlphaPilot teams, percentage of completed runs, mean score and standard deviation across all 25 evaluation courses.



(a) The number of completed runs for the top 20 teams.



(b) The mean and standard deviation of scores.

Figure 4-13: The figures above show the number of completed runs by each of the top 20 AlphaPilot teams along with the mean and standard deviation of their scores for all 25 runs.

using infrared beacons or polynomial trajectory planning such as [33], [58]. Other methods used for planning either used manually-defined waypoints or used sampling-based techniques for building trajectory libraries. 5 of the 19 teams to use model based techniques also incorporated some form of perception awareness to their planning algorithms.

Control: The predominant methods for control were linear control techniques and model predictive control [75]. The other algorithms that were used were geometric and backstepping control methods [76]. Five of the 19 teams that used model-based techniques also incorporated some form of perception awareness in their planning algorithms.

Analysis of Trajectories

To visualize the speed along the trajectories, we discretized the horizontal plane and colored each grid cell on a logarithmic scale according to the average of the local speeds. From the figure, we can observe that most teams chose to slow down for the sharp turns at gates 2 and 7. We can also observe that in general the average speed around gates is lower than at other portions of the environment, which can be attributed to the need to 'search' for the next gate. Fig. 4-12 also shows the crash

locations of all the failed attempts. We observe that many of the crash locations are in the vicinity of the gates, which may be caused by widespread use of visual-servo-based techniques combined with the fact the infrared gate beacons are more likely to leave the camera field of view at close range.

Individual Performance of top Teams

For individual performance, we analyze the number of completed runs for each of the top teams and the mean and standard deviations of the scores. This is shown in Fig. 4-13. Given that the final scoring function for the competition only included the five best scoring runs, teams were encouraged to take significant risk to improve their top scores. Consequently, 75% of the contestants failed to complete the course in at least half of their 25 runs. Only one team completed the entire course in all of their 25 runs. Notably, this team also achieved very consistent scores across all runs. While their average score across all runs ranks among the highest of all teams; their final score based on the five best runs is ranked significantly lower, showing that risk-taking strategies are indeed rewarded.

4.4 The Blackbird Dataset: A Large-Scale Dataset for UAV Perception in Aggressive Flight

Much of the work presented in this section has been published in The International Symposium on Experimental Robotics (ISER) in 2018 [15] and an extended version has been submitted to The International Journal of Robotics Research in 2019 [16].

As hardware and control capabilities of aerial platforms have increased dramatically in the last decade, autonomy algorithms in general, and perception algorithms in particular, have struggled to keep up. New videos of remotely piloted vehicles racing through gates at high speeds have drawn a sharp contrast with current autonomy capabilities and has inspired many researchers to challenge autonomy algorithms to keep up with the hardware capabilities.

Table 4.5: UAV Visual Inertial Datasets Comparison

	EuRoC MAV [80]	UPenn Fast Flight [81]	Zurich Urban MAV [82]	UZH-FPV Drone Racing [83][a]	**Ours**
Environments	2	1	3	2	**5**[b]
Sequences	11	4	1	27	**186**
Camera	20 Hz	40 Hz	20 Hz	30/50 Hz	**120 Hz**
IMU	200 Hz	200 Hz	10 Hz	**500/1000 Hz**	100 Hz
Motor Tachometers	n/a	n/a	n/a	n/a	**∼190 Hz**
Depth Cameras	n/a	n/a	n/a	n/a	**60 Hz**
Segmentation	n/a	n/a	n/a	n/a	**60 Hz**
Max Distance	130.9 m	700 m	**2 km**	340.1/923.5 m	860.8 m
Top Speed	2.3 m s^{-1}	17.5 m s^{-1}	3.9 m s^{-1}[c]	12.8/**23.4 m s^{-1}**	**13.8 m s^{-1}**[d]
mm Ground Truth	100 Hz [e]	n/a	n/a	20 Hz[f]	**360 Hz**

[a] Values are separated as (value indoors)/(value outoors)
[b] Additional environments may be rendered using FlightGoggles [c] Instantaneous velocity from GPS
[d] Fastest among indoor trajectories [e] Accuracy only guaranteed during low agility maneuvers
[f] Drops out frequently at high speeds or during aggressive maneuvers

One of the key challenges that must be solved in high speed autonomy is the ability to localize in an unknown environment without the aid of external systems such as GPS or motion capture. Due to their light weight and low power consumption, a combination of inertial measurement units (IMUs) and cameras have been the most popular sensor configuration for small UAVs. This sensor set has led to an increased focus on visual inertial simultaneous localization and mapping (VI-SLAM) for autonomous unmanned aerial vehicle (UAV) flight in recent years. With improved hardware and algorithms recent works such as [19], [77], [78] have demonstrated the possibility of high speed flight using on board state estimation, however, the performance gap with human operators remains large.

With the increased focus of high performance perception algorithms, two recently released challenges have pushed forward the race to develop autonomy algorithms at near human abilities. The [79] challenge organized by DARPA involves traversing a variety of environments including unknown and unstructured scenarios such as those typical of caves. This has challenged the community to develop robust autonomy algorithms capable of navigating in unstructured environments for large durations of time with a high degree of situational awareness.

One of the primary challenges in the development of new high performance perception algorithms is the difficulty in creating a safe development platform. Developing high performance UAVs with a full sensor suite and safely operating them requires a long and costly process. In addition, the desire to have a ground truth motion capture solution and a safe operating environment typically limits experiments to small motion capture rooms with no variation in the surrounding environment. The Blackbird UAV Dataset seeks to remove the heavy development burden on researchers by providing a large and highly varied dataset that pushes past the current capabilities of on board algorithms, allowing development to be focused on the perception algorithms alone.

A secondary driver for the Blackbird UAV Dataset has been new extensions of traditional visual inertial odometry algorithms that use novel sensors such as event based camera systems [84], [85], or motor tachometers [86], [87]. To aid extensions such as the algorithms above, the dataset includes motor tachometer data, a wide range of visual sensors beyond the traditional stereo pair, and the ability for a user to generate new types of exteroceptive data in the future using the FlightGoggles simulation system in conjunction with the existing proprioceptive and ground truth data in the dataset.

The key features of the Blackbird UAV Dataset can be found in Table 4.5. The core of the dataset, including the majority of the flights with proprioceptive measurements (IMU and motor tachometers) and grayscale imagery was initially published in [15]. The full dataset provided here provides an extension of that work adding *a*) new, higher speed flights, *b*) a dynamic visual environment, *c*) RGB streams with motion blur for all flights, *d*) depth streams for all flights, *e*) ground truth semantic labels for all images, and *f*) benchmarking of state-of-the-art visual odometry algorithms.

The benchmarking of two state-of-the-art visual and visual-inertial odometry algorithms is performed to provide a point of reference for users of this new dataset. Namely, we benchmark VINS-Fusion, from [88], and ORB-SLAM, from [89], which are currently the top two open-source, vision-based odometry estimation algorithms in the KITTI odometry benchmark, [90]. We report this results in Section 4.4.6. In

addition, we provide tools to run the same evaluation on other algorithms.

4.4.1 Related Work

As can be seen from Table 4.5, we seek to combine and add to the best features of existing UAV datasets, with the high accuracy ground truth found in indoor datasets, *e.g.*[80], the high speed and agility found in outdoor datasets, *e.g.*[81], [82], [91], and additionally providing an unprecedented variety and volume of sensor modalities. [80] present the widely used EuRoC MAV datasets, a collection of 11 trajectories with an average speed of $1\,\mathrm{m\,s^{-1}}$ and highly accurate ground truth on half of the sequences. While the dataset was a large step forward, it contains relatively low speeds and low rate cameras by today's standards. [81] present a fast outdoor flight dataset with the same trajectory at 4 different speeds and GPS ground truth. Although this does allow the evaluation of online algorithms in outdoor settings with long distances, it does not provide high quality ground truth or varied visual environments. The Zurich Urban MAV Dataset, [82], contains 2 km of visual and inertial data recorded from a tethered UAV flying in an urban setting, but it lacks high-precision ground truth pose. [91] present TorontoCity, a very large UAV dataset with data from multiple perspectives of the city of Toronto captured from different cameras and a LiDAR. TorontoCity focuses on tasks such as segmentation and classification of the environment. It, however does not contain inertial information and cannot be used in the context of visual inertial navigation. More recently, [83] released the UZH-FPV Drone Racing dataset, which does include a set of aggressive outdoor trajectories, and, like Blackbird, provides visual-inertial data and highly accurate ground truth, though not through highly dynamic maneuvers. While UZH-FPV improves over previous datasets by bringing highly accurate ground truth to outdoor, aggressive flights, the Blackbird UAV Datasets stands out due to the following features: *a*) the volume of unique flights, *b*) the variety of its visual data, *c*) the high rate and millimeter precision of its ground truth data persists throughout every flight and maneuver, *d*) high rate measurement of the four rotor speeds, *e*) exact depth data for the left and down cameras, and *f*) ground truth segmentation for both the left and down cameras.

The Blackbird dataset also differs from existing UAV datasets due to the use of synthetic imagery, which allows for the generation of high rate, visually varied imagery in cluttered environment while maintaining the real data from on board proprioceptive sensors. The use of synthetic imagery for algorithm development has seen a rapid increase in robotics driven by the volume of data required for learning algorithms and the capability of modern systems to generate photorealistic synthetic imagery. Synthetic datasets have been made for verification of SLAM algorithms in a known 3D world such as [12]–[14], and for generating large labeled datasets for deep learning [12], [34]. In addition to the use of synthetic imagery in datasets, simulation based evaluation frameworks have become prevalent. Starting from the ubiquitous robotics simulator Gazebo presented by [8], newer simulators such as AirSim by [37] and NVIDIA Issac provide similar capabilities to the FlightGoggles system used for this dataset in rendering photorealistic imagery. By combining true (hard to simulate) proprioceptive data from our UAV with (easy to simulate) exteroceptive data from FlightGoggles, we get a high rate and high accuracy dataset of challenging flight scenarios.

This contribution is organized as follows. In Section 4.4.2 we describe the quadrotor platform and system that was used for collecting the data presented in this paper. Section 4.4.5 describes the data provided in the dataset. In Section 4.4.3 and Section 4.4.4 we describe the format of the available data and the validation of the data. Section 4.4.6 provides benchmarks of visual and visual-inertial navigation algorithms. Finally, Section 4.4.7 describes the known issues within the dataset.

4.4.2 Data Collection Setup

UAV Platform

Data was collected using a custom built quadrotor UAV designed for agile autonomous flight, which we call Blackbird (Figure 4-14). The UAV caries an Xsens MTi-3 IMU, custom made optical motor encoders for high rate motor speed measurements, a DJI Snail propulsion system, and a NVIDIA Jetson TX2 on a custom carrier board. The

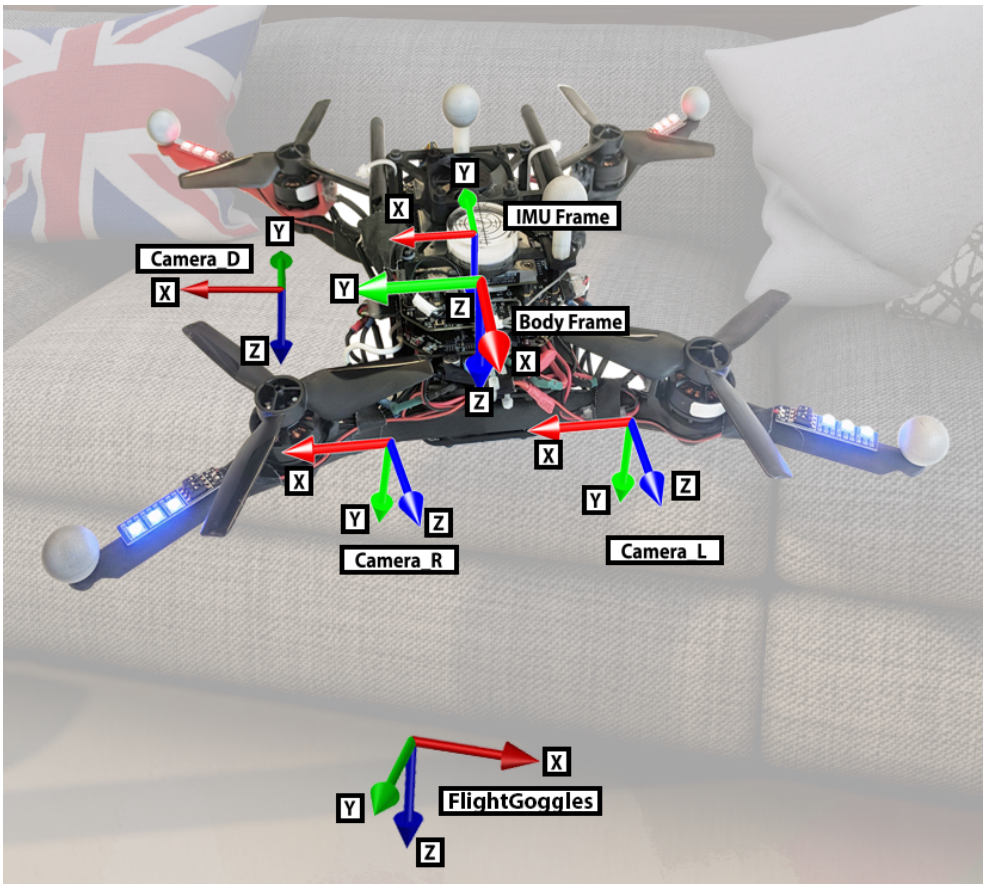


Figure 4-14: Coordinate frames in use for this dataset. Note that `Camera_D` and `Body_Frame` are coincident, but are translated in the figure for visualization.

Table 4.6: Quadrotor characteristics

Property	Value	Description
Mass	0.915 kg	Mass with battery
I_{xx}	4.9×10^{-2} kg m^{-2}	X moment of inertia
I_{yy}	4.9×10^{-2} kg m^{-2}	Y moment of inertia
I_{zz}	6.9×10^{-2} kg m^{-2}	Z moment of inertia
Arm Length	0.13 m	Center to end of arm
f_x, f_y	665.108 px	Cameras' focal length
FOV	60.0°	Cameras' vertical FOV
σ^{gyro}	1.2×10^{-4} rad s$^{-1}\sqrt{\text{Hz}}$	Gyroscope noise density
σ_b^{gyro}	4.7×10^{-6} rad s$^{-2}\sqrt{\text{Hz}}$	Gyroscope random walk bias
σ^{accel}	2.0×10^{-3} m s$^{-2}\sqrt{\text{Hz}}$	Accelerometer noise density
σ_b^{accel}	4.4×10^{-5} m s$^{-3}\sqrt{\text{Hz}}$	Accelerometer random walk bias
C_T	2.27×10^{-8} N/rpm^2	RPM to thrust coefficient
Image Size	1024 px × 768 px	Image width and height
z_{near}	0.01 m	Near render plane distance for depth camera.
z_{far}	100.0 m	Far render plane distance for depth camera.
Stereo baseline	10 cm	Distance between stereo pair

body of the vehicle is constructed from 3D printed MarkForged Onyx continuous carbon fiber composite. Rubber dampeners are used to mechanically isolate vibrations from the propulsion system from flight sensors. The physical properties of the quadrotor as well as sensor statistics are shown in Table 4.6.

Experimental Setup

Flights were performed in an $11\,\text{m} \times 11\,\text{m} \times 5.5\,\text{m}$ motion capture room, with 24 OptiTrack Prime 17W cameras[1] providing the 6D pose of the drone at 360 Hz. Each flight in the dataset is between 3-4 minutes long as the drone traces out a pre-defined periodic trajectory using a non-linear dynamic inversion controller from [92]. The drone is controlled and data is recorded by a custom software framework, [19], using the Lightweight Communications and Marshaling (LCM) protocol, [21].

Visual Data Generation

Visual data was generated in post process using the FlightGoggles photo-realistic image generation system from [17], [19]. FlightGoggles uses the ground truth 6D pose of the drone from motion capture to generate images from the viewpoint of each camera on the drone in a virtual environment. The system allows for complete control over the visual appearance of the environment, the rate of camera images (up to the 360 Hz motion capture rate), the number of cameras, and each camera's location and intrinsic and extrinsic properties. The visual data generated by FlightGoggles has been previously validated for use in visual inertial state estimation in [19].

As part of the rendering process, a number of transforms are used to transform NED ground truth data from world frame into FlightGoggles' environment frame. To ensure that all recorded flights in each trajectory takeoff from a common altitude and overlap in the XY plane, $\mathbf{T}_{\text{norm}}^{\text{mocap}}$ is introduced as a *per-flight* translational offset applied to the ground truth data to correct for offsets introduced during dataset collection and generate a set of normalized trajectories. $\mathbf{T}_{\text{FG}_{\text{env}}}^{\text{norm}}$ is a *per-trajectory*

[1] For the `Egg` trajectory, the room was extended to twice the length and 24 motion capture cameras were added to provide full coverage.

common transform applied to the normalized trajectory that positions flights into the FlightGoggles environment in a collision-free manner. The full transform chain from ground-truth coordinates to render coordinates is shown in Equation (4.1).

$$\mathbf{T}_{\mathrm{FG_{env}}}(t) = \mathbf{T}^{\mathrm{norm}}_{\mathrm{FG_{env}}} \cdot \mathbf{T}^{\mathrm{mocap}}_{\mathrm{norm}} \cdot \mathbf{T}_{\mathrm{mocap}}(t) \tag{4.1}$$

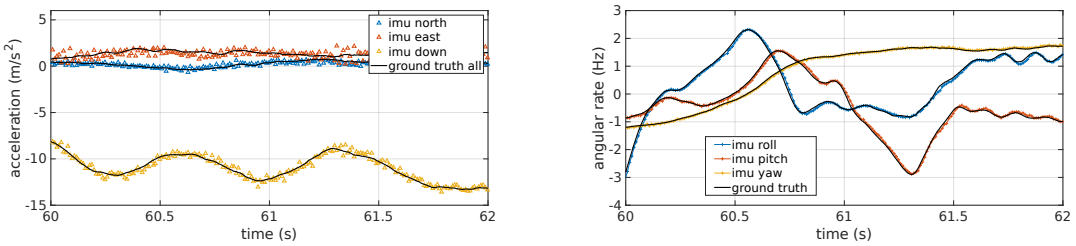
Where $\mathbf{T}_{\mathrm{FG_{env}}}(t)$ is the render pose in FlightGoggles' virtual environment at timestamp t. These transforms can be used to align the generated visual map between different trajectories by the composition of these transforms for the evaluation of algorithms that rely on re-localization.

Sensor Calibration and Temporal Synchronization

The Kalibr package, [93], was used to find the noise characteristics of the IMU and the IMU-to-camera transform. A 3 second period at rest is included in every flight to allow for initialization of the time varying IMU bias. Force and torque coefficients of the drone were found experimentally through measurements in a wind tunnel to obtain the relationship between motor speeds and vehicle dynamics. Clock synchronization between motion capture data and on-board sensors was performed using a combination of clock estimation over gigabit ethernet and chrony, from [23], over the wireless network, with an upper bounded offset of ±5ms.

4.4.3 Dataset Format

Each flight within the dataset contains timestamped values for the following: ground truth 6D pose of the UAV at 360 Hz, IMU measurements at 100 Hz, RPM measurements for each motor at ∼190 Hz, three camera streams of grayscale images (forward facing stereo pair and downward facing) at 120 Hz, three streams of RGB images with simulated motion blur at 60 Hz from the same cameras, and two streams of depth images (forward-facing left and down-facing cameras) at 120 Hz. The data is provided as grayscale, RGB, depth and segmentation video feeds, LCM logs, ROS bags and CSV files (one file per ROS topic) for easy use in typical pipelines. Scripts and binaries



(a) IMU accelerometer vs ground truth.

(b) IMU gyroscope vs ground truth.

Figure 4-15: Derivative of position and rotational ground truth data compared with accelerometer and gyroscope data for a flight at $4\,\mathrm{m\,s^{-1}}$.

necessary to re-render images using FlightGoggles at other rates (up to 360 Hz) or camera parameters and configurations are available at `http://blackbird-dataset.mit.edu/`.

In addition to the raw data streams, the full calibration information of the UAV system is included in the dataset i.e, IMU noise characteristics, IMU-camera transform, camera intrinsic and extrinsic parameters (as currently rendered), and torque and thrust coefficients. Some of these are included in Table 4.6. The dataset file structure is specified in Figure B-1.

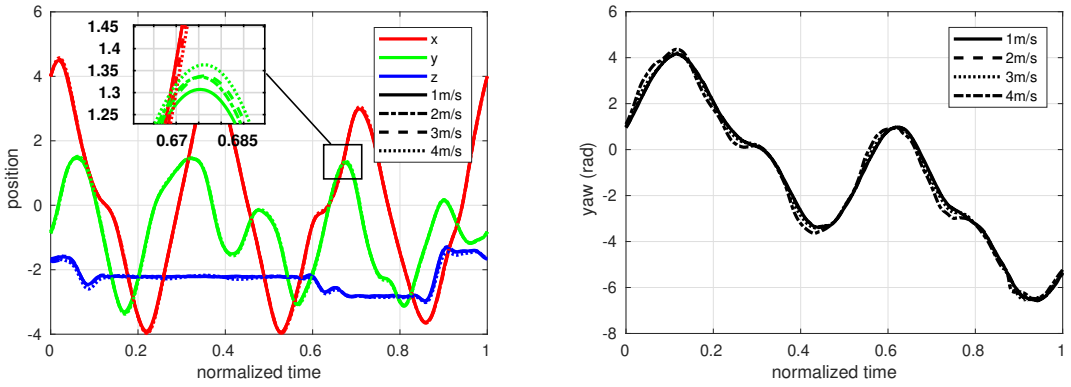
Lossless Temporal Image Compression

In an effort to reduce the download size of the pre-rendered dataset, all image feeds are temporally compressed using lossless High Efficiency Video Coding (HEVC) and are stored in platform-agnostic video containers. OpenCV-based helper utilities for republishing the provided video feeds into ROS are provided in the dataset.

Depth Encoding and Resolution

The depth images provided in this dataset are distributed as single-channel 8-bit resolution video feeds compressed using a non-linear bitwise mapping that increases depth resolution near the camera. The mapping from compressed depth to true depth in meters is given by

$$z_{\text{meters}}(z_{\text{compressed}}) = z_{\text{near}} + \frac{z_{\text{compressed}}^4 \cdot z_{\text{far}}^2}{255^4 \cdot (z_{\text{far}} + z_{\text{near}})} \tag{4.2}$$



(a) Position tracking

(b) Yaw tracking

Figure 4-16: Tracking precision while flying the same trajectory at speeds of 1 to 4m s^{-1}

where z_{near} and z_{far} are the depth values for the depth camera's near and far rendering planes and are defined in Table 4.6.

4.4.4 Data Validation

Validation of ground truth data and inertial measurements in both quality and temporal synchronization was performed by comparing raw inertial measurements with derivatives of the ground truth pose using a Savitzky-Golay filter, [94]. Figure 4-15 shows a comparison of the IMU angular rate and acceleration with respect to ground truth. The accuracy of the drone's motor speed sensors were verified through the use of an external tachometer.

Trajectory Tracking

A feature of the provided dataset is the ability to repeatably run perception algorithms on a nominal trajectory pattern while flying at different speeds with new inertial, dynamical, and visual data. A comparison of the ground truth pose of the same trajectory (Sphinx, see Figure 4-17) flown four times at speeds between 1 m s^{-1} and 4 m s^{-1} is shown in Figure 4-16, with sufficient tracking accuracy for a user to isolate the speed of flight from other parameters that may affect VI-SLAM algorithms.

Table 4.7: Blackbird Dataset Flights

Trajectory	Constant Yaw									Forward-Facing									
Top speed ($m\,s^{-1}$)	0.5	1.0	2.0	3.0	4.0	5.0	6.0	7.0	13.8	0.5	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
3D Figure 8	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
Ampersand	-	M	M	H	-	-	-	-	-	-	H	H	-	-	-	-	-	-	-
Bent Dice	-	E	E	M	M	-	-	-	-	E	E	E	E	-	-	-	-	-	-
Clover	-	H	H	H	H	H	H	-	-	H	H	H	H	H	H	-	-	-	-
Dice	-	-	E	E	M	-	-	-	-	-	E	E	E	-	-	-	-	-	-
Egg	-	-	-	-	-	-	-	-	H	-	-	-	M	M	H	H	H	H	H
Flat Figure 8	✓	✓	✓	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
Half-Moon	-	E	E	E	M	-	-	-	-	-	M	M	M	M	-	-	-	-	-
Mouse	-	M	M	M	M	M	M	M	-	M	M	M	M	M	M	M	M	-	-
Oval	-	-	M	M	H	-	-	-	-	-	M	M	H	H	-	-	-	-	-
Patrick	-	E	E	E	E	M	-	-	-	E	E	E	E	E	-	-	-	-	-
Picasso	M	M	M	M	M	M	M	-	-	M	M	-	H	H	H	-	-	-	-
Sid	-	E	E	E	E	E	E	E	-	M	M	M	M	M	M	-	-	-	-
Sphinx	-	H	H	H	H	-	-	-	-	-	M	M	M	M	-	-	-	-	-
Star	-	M	M	M	H	H	-	-	-	M	M	M	M	M	H	-	-	-	-
Thrice	-	E	E	E	E	E	M	M	-	E	E	E	E	E	E	M	-	-	-
Tilted Thrice	-	E	E	E	E	E	M	M	-	E	E	E	E	E	E	E	-	-	-
Winter	-	M	M	M	M	M	-	-	-	M	-	H	H	H	-	-	-	-	-

Click flight for grayscale video preview of flight in all rendered environments.

Collision Checking

To verify that the trajectories being rendered do not collide with any obstacles in the virtual environment, *bullet3* based simulation, from [95], was used to verify that the trajectories were collision free in the virtual environment prior to rendering the camera streams.

4.4.5 Dataset Generation Methodology

The trajectories in this dataset were designed to encompass scenarios with low agility such as a simple oval trajectory to trajectories that have rapid changes in the viewpoint or rotational rate of the drone. In this section, we first describe the trajectory generation and choices made for generating the trajectories available in this dataset. Second, we classify the trajectories into categories for evaluation and ease of use. Finally, we describe the available exteroceptive data and the process for generating the data.

Trajectory Generation

To allow methodical evaluation of algorithms across varying degrees of agility, we choose to keep the translational trajectory of the quadrotor the same across variations and independently vary the following flight characteristics: *speed and yaw* for each trajectory type. This variations can easily be achieved by exploiting the differential flatness properties of the quadrotor. Further, we include *18* distinct types of trajectories with varying agility. The trajectories included in this dataset are shown in Fig. 4-17. As can be seen in the figure, the variation in the translational trajectories is high ranging from simple smooth trajectories (e.g oval, egg, half moon) to trajectories that exhibit rapid changes in the direction of motion (e.g ampersand).

The included trajectories are classified by difficulty ([E]asy, [M]edium, [H]ard) (shown in Table 4.7). These categories were empirically determined according to the mean number of features tracked when running the visual inertial state estimation pipeline from [19] on the visual environment each flight was originally flown in. These trajectories range in complexity from an oval with constant yaw and altitude to trajectories with varying speed, altitude, and yaw as they weave through visual obstacles (*e.g.*the Sphinx) as shown in Figure 4-17. For flights that are rendered in multiple environments (see Figure 4-18), some environments are harder for state estimation than others due to the size and visual complexity of the environment.

To generate smoothly trackable trajectories, minimum snap optimization was performed over a set of requested waypoint positions using the non-linear optimization technique described in [33], [96], with boundary conditions to make the trajectory periodic. Table 4.7 shows all the sequences included. There are 171 unique flights of approximately three minutes each for a total of over 10 h and 60 km of ground truth pose, inertial measurements, motor speeds, and rendered imagery. Trajectories were designed for specific flight environments (*e.g.*Sphinx for Ancient Egypt Museum Room) and are therefore particularly well suited to those environments, however, where it does not result in virtual collisions with objects, the same trajectory can be re-rendered in multiple environments using FlightGoggles. We have done this, for ex-

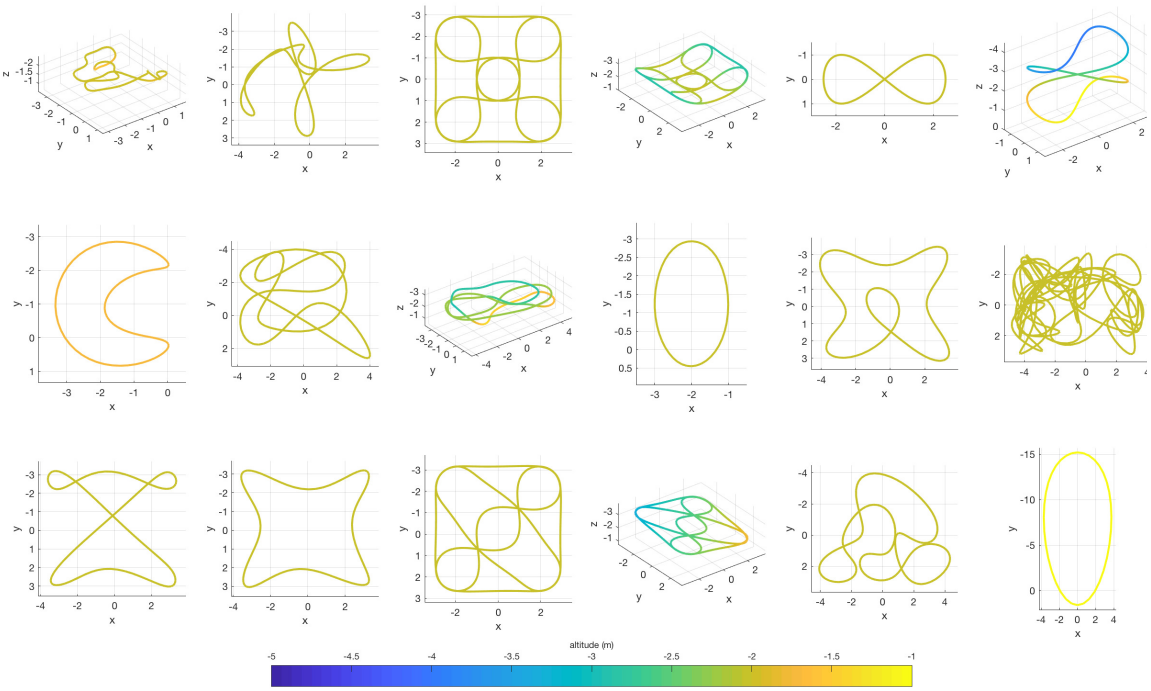
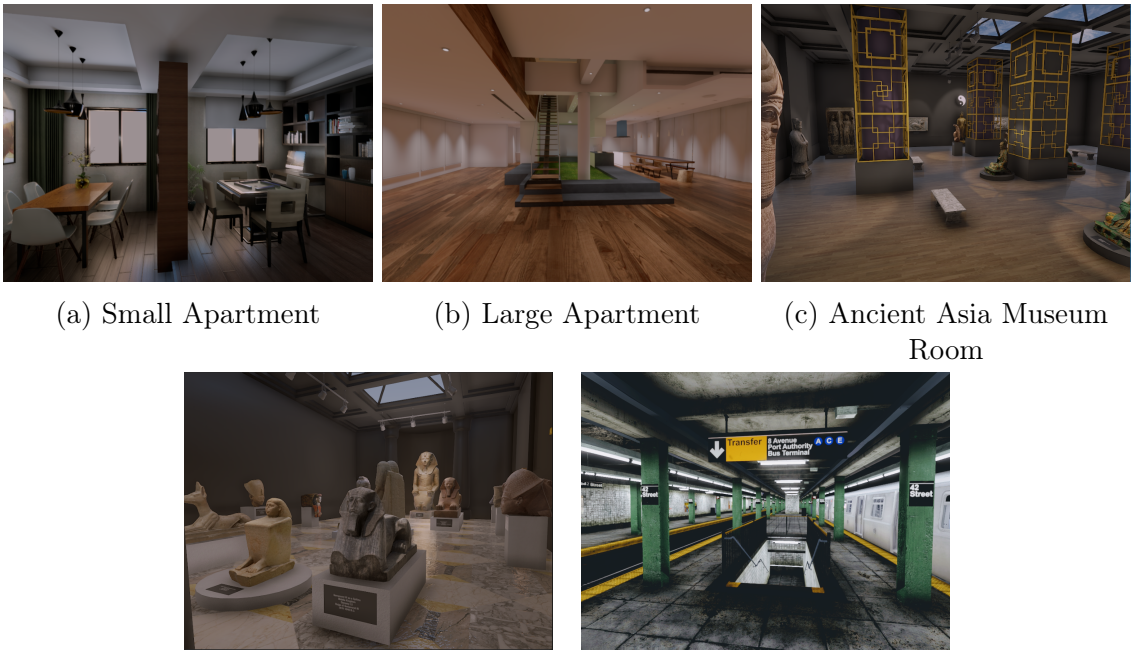


Figure 4-17: Diagrams of trajectory paths included in this dataset.



(a) Small Apartment

(b) Large Apartment

(c) Ancient Asia Museum Room

(d) Ancient Egypt Museum Room

(e) NYC Subway Station

Figure 4-18: Five rendering environments used in the dataset to generate visual data.

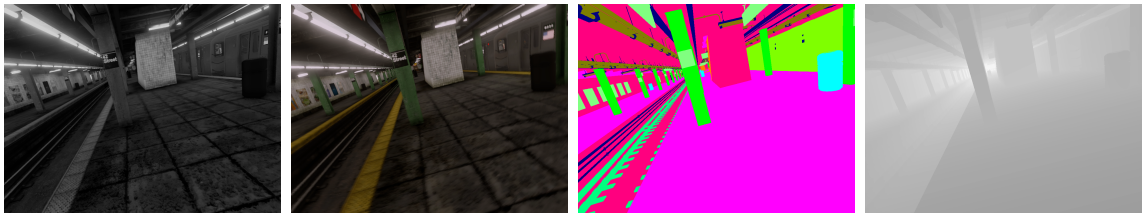
ample, on the benchmarks described in Section 4.4.6. Flights marked in Table 4.7 by ✓ do not have visual data associated with them. These 'figure 8' flights were designed to avoid dynamic biases by having the quadrotor fly roughly equal amounts of time in all directions and at different speeds. They are therefore intended for calibration of dynamics parameters in algorithms such as the ones in [86] and [97] that make use of dynamics information.

Exteroceptive Sensor Generation

As an extra challenge for vision-based perception algorithms, this dataset extends the work presented at ISER by [15] by providing 60 Hz RGB image feeds with added motion blur on the left stereo and downward-facing camera. Motion blur for the RGB video feeds was generated at render time using dense optical flow, depth information, and a rotary shutter model. The simulated exposure length for the motion blurred cameras was set to 100% of the time between frames. Camera shader code from the AirSim simulator by [37] was modified and integrated into FlightGoggles and used to render semantic segmentation and depth images. Depth image streams are provided for the left stereo and downward-facing camera at 60 Hz. Semantically labeled image streams were also generated for the left stereo and downward-facing camera at 60 Hz using 22 curated semantic labels. These semantic labels were manually generated and added to all object mesh renderers viewable in each environment from the pre-rendered trajectories. A list of all semantic labels along with their color mappings is tabulated in Table 4.8. Additionally, Figure 4-19 contains a side-by-side comparison of the various exteroceptive sensors types provided in this dataset.

Environments with Dynamic Obstacles

We provide a sequence with dynamic obstacles based on the previously existing `NYC Subway Station` environment. In this added sequence, the subway train in the environment travels in and out of the station on a loop, with a mean velocity of $10.0\,\mathrm{m\,s^{-1}}$.

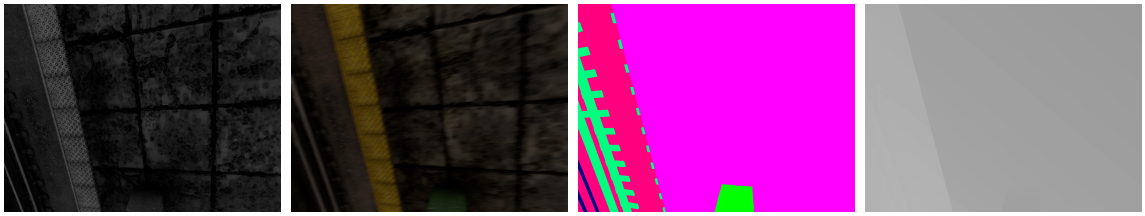


(a) Left stereo grayscale (120 Hz, no motion blur)

(b) Left stereo RGB (60 Hz, with motion blur)

(c) Left stereo semantic labels (60 Hz)

(d) Left stereo depth (60 Hz)



(e) Downward-facing grayscale camera (120 Hz, no motion blur)

(f) Downward-facing RGB camera (60 Hz, with motion blur)

(g) Downward-facing camera semantic labels (60 Hz)

(h) Downward-facing depth camera (60 Hz)

Figure 4-19: Camera image feeds provided in the pre-rendered dataset. Note that a grayscale feed is provided for the right stereo camera at 120 Hz, but is omitted from this figure.

Sofa	Cushion	Book	Cabinet	Table	Wall	Floor	Chair
Shelf	Curtain	Lights	Window	Screen	Stairs	Sky	Statue
Column	Rails	Trashcan	Train	Sign	Misc		

Table 4.8: The color scheme used in that dataset for the 22 provided semantic labels.

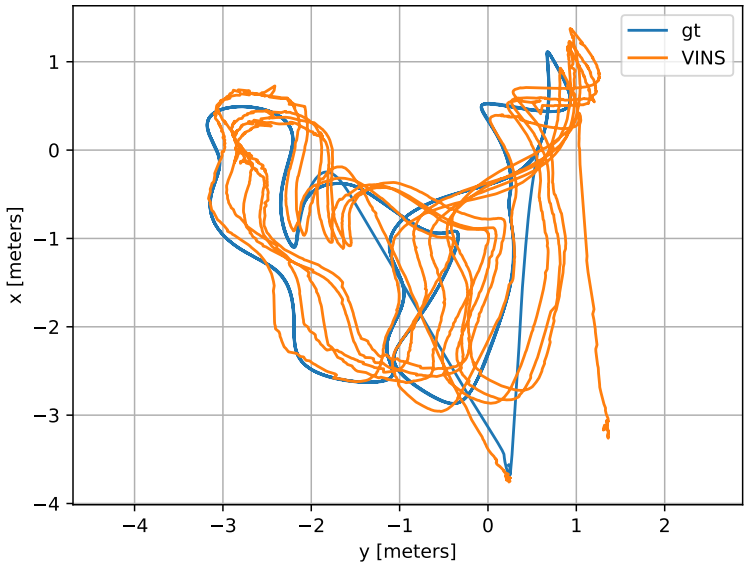
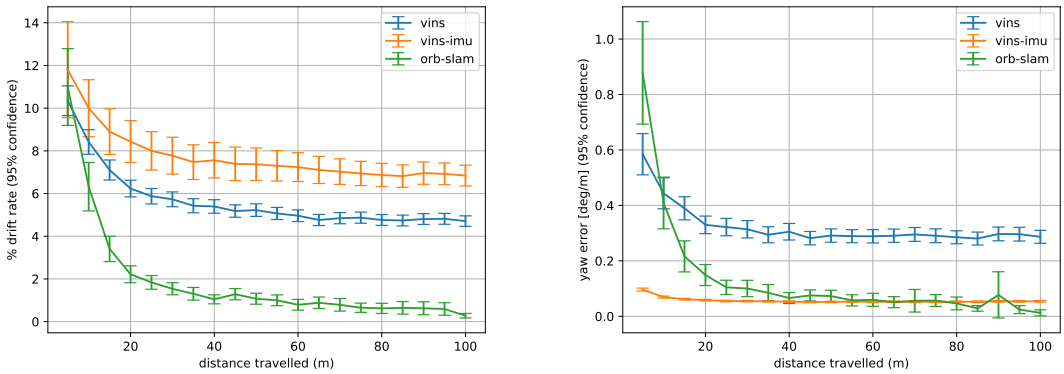


Figure 4-20: VINS-Fusion against the ground truth for the 'Ampersand' trajectory flown at $1\,\mathrm{m\,s^{-1}}$ and constant yaw in the environment 'Small Apartment'.

4.4.6 Benchmarks

We evaluate the following state-of-the-art visual and visual-inertial estimation algorithms on a select subset (see Appendix A.1) of the data: VINS-Fusion ([88]), with and without IMU measurements, and ORB-SLAM2 ([89]). We abbreviate these as VINS-IMU, VINS, and ORB-SLAM2, respectively. These are, at the time of this writing, the top two open-source vision-based odometry estimation algorithms in KITTI's odometry challenge leaderboard (`http://www.cvlibs.net/datasets/kitti/eval_odometry.php`). For the purpose of this evaluation, the loop closure features of both these algorithms is disabled. As an example, we show in Figure 4-20 the top view of the estimated trajectory from VINS against the ground truth for the 'Ampersand' trajectory flown at $1\,\mathrm{m\,s^{-1}}$ and constant yaw in the environment 'Small Apartment'.

Following [90], we measure the translation error, as percent of distance travelled, and the yaw error, in degrees per meter travelled. We summarize these key performance indicators (KPI's) by taking the average of the errors evaluated at intervals of a minimum of $5\,\mathrm{m}$. In Figures 4-21 and 4-22 we report the performance of the three algorithms in terms of said KPI's. The per-flight averages for each algorithm



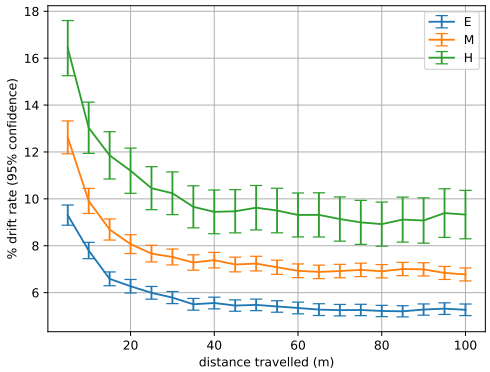
(a) Translation error

(b) Yaw error

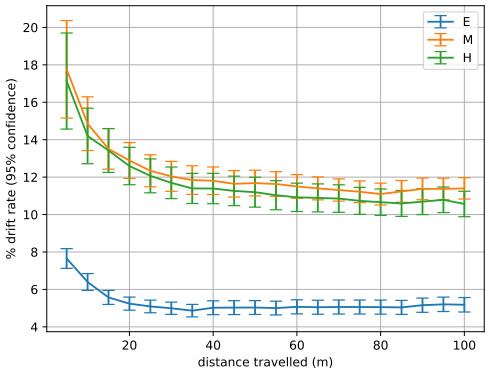
Figure 4-21: Performance of state-of-the-art visual odometry algorithms, VINS-Fusion (stereo and mono+IMU) and ORB-SLAM on 33 Blackbird flights. Note: we only average the errors for 33 flights here for fair comparison, because these are the flights for which all three estimators successfully tracked enough of the trajectory to allow for the computation of these averages.

are available in the dataset, as are scripts to evaluate any pose estimation algorithm by the same metrics.

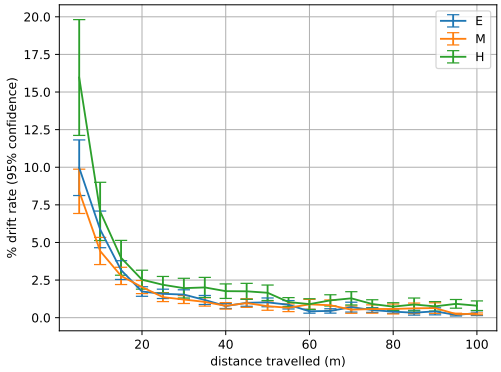
Given the challenging nature of many of the flights in the dataset, however, the estimators often fail to track some of those flights and diverge suddenly from the true trajectory. Note here that for both VINS-Fusion and ORB-SLAM2 the parameters used are the default parameters for the EuRoC dataset with appropriate modifications for parameters such as image size and camera calibration. Tuning these parameters could lead to better tracking performance on this dataset. We estimate that the algorithm has 'failed' to track when the translation error is above 100% (of distance travelled), and exclude those flights from the averages shown in Figure 4-22. For transparency, in Appendix A.1, Tables A.1 to A.3 we report for each algorithm the flights in which it 'failed' to track. We view the lost tracks as the challenge for state estimation algorithms to provide robust tracking performance in the highly dynamic scenarios that today's robotics hardware is now capable of.



(a) VINS



(b) VINS-IMU



(c) ORB-SLAM

Figure 4-22: Average performance of VINS, VINS-IMU, and ORB-SLAM on the flights they successfully tracked, separated by difficulty level. Note that in this figure the sets of plots are not the same for every algorithm (because they fail in different flights), hence they are not plotted against each other here.

4.4.7 Known Issues

In this dataset, we synchronized motion capture ground truth data with onboard IMU measurements using camera exposure timestamps provided by OptiTrack, IMU measurement timestamps provided on arrival by our UAV's TX2, and clock sync and drift correction provided by Chrony, [23]. However, due to the complexity and stochastic nature of the systems involved, we are only able to guarantee IMU and ground truth temporal alignment to within ± 5 ms across all flights in this dataset. This upper bound was verified in post process by cross correlation of IMU and ground truth measurements for each flight.

Over the course of the various data recording sessions required to create this dataset, at times we observed an increase in the mean tracking error of the motion capture ground-truth setup with respect to the freshly calibrated system. This is due to thermal expansion and contraction of the motion capture support beams. Therefore, we recalibrated often to ensure that the mean tracking error was under 1.5 mm for all flights.

4.5 Collision Avoidance of Dynamic Actors Using Online Perception-aware Planning

This work proposes a preliminary online perception-aware quadrotor trajectory generation algorithm for quadrotors operating in close proximity to dynamic obstacles. Perception-aware algorithms benefit from the exploitation of past measurements to perform either opportunistic loop closure or increase the length of the feature tracks to improve the quality of the state estimation. In this scenario, a human pointcloud is rendered in real time based on skeleton tracking motion capture data, while a quadrotor is simultaneously flying in a separate motion capture room (see Fig. 4-24 and Fig. 4-25). While both dynamic actors (*i.e.* human and quadrotor) are physically in separate spaces, they are both in the same virtual environment (see Figure 4-23). For the experiment, the drone is tasked with navigating between arbitrary, dynamic

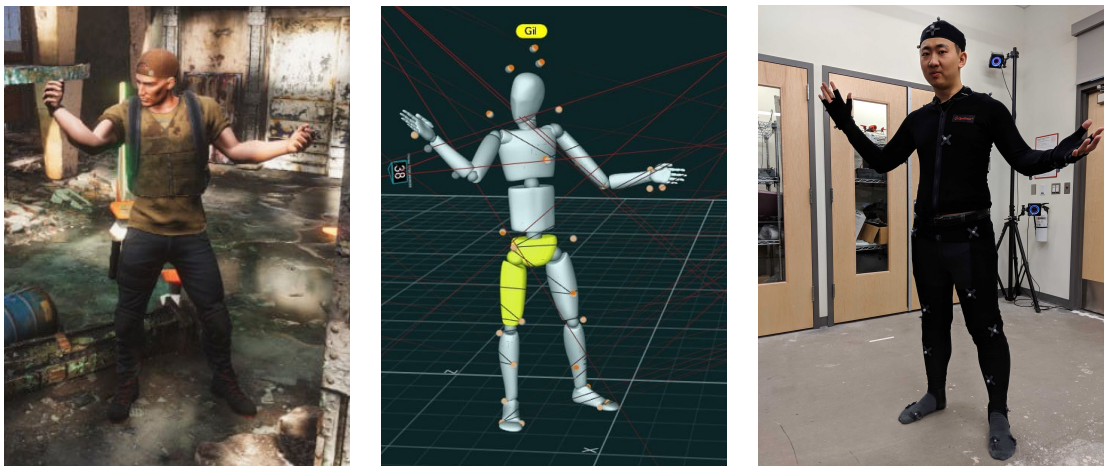


Figure 4-23: A dynamic human actor in the FlightGoggles virtual environment is rendered in real-time, based on skeleton tracking data of a human in a motion capture suit with markers.

waypoints while optimizing for weighted visual feature co-visibility and obeying obstacle constraints. All online trajectory replanning takes place on an in-flight UAV carrying an NVIDIA Jetson TX2 computer.

4.5.1 Related Work

Differential flatness is often exploited for trajectory generation for quadrotors such as in [33], [98], [99]. Quadrotor trajectory generation is a well studied problem. One of the first approaches to generating trajectories was proposed by Mellinger and Kumar [58]. They proposed to model the trajectory using polynomials between one keyframe to the next. The optimization function that they used was minimizing the snap with the intuition being that the snap was the first derivative which contained all of the inputs to the quadrotor and minimizing this enforces smoothness in the trajectory. Richter *et al.* [33] proposed extension to this work that bounds the actuator and add a time penalty to the cost function with the intention of flying faster. Other methods have been proposed that minimize the jerk [99] of the quadrotor which makes the assumption that the yaw of the quadrotor is fixed and does not change during the execution of the trajectory. Richter *et al.* [33] use a corridor constraint to deal with collisions for polynomial trajectory planning for quadrotors. More recently, Liu *et al.*

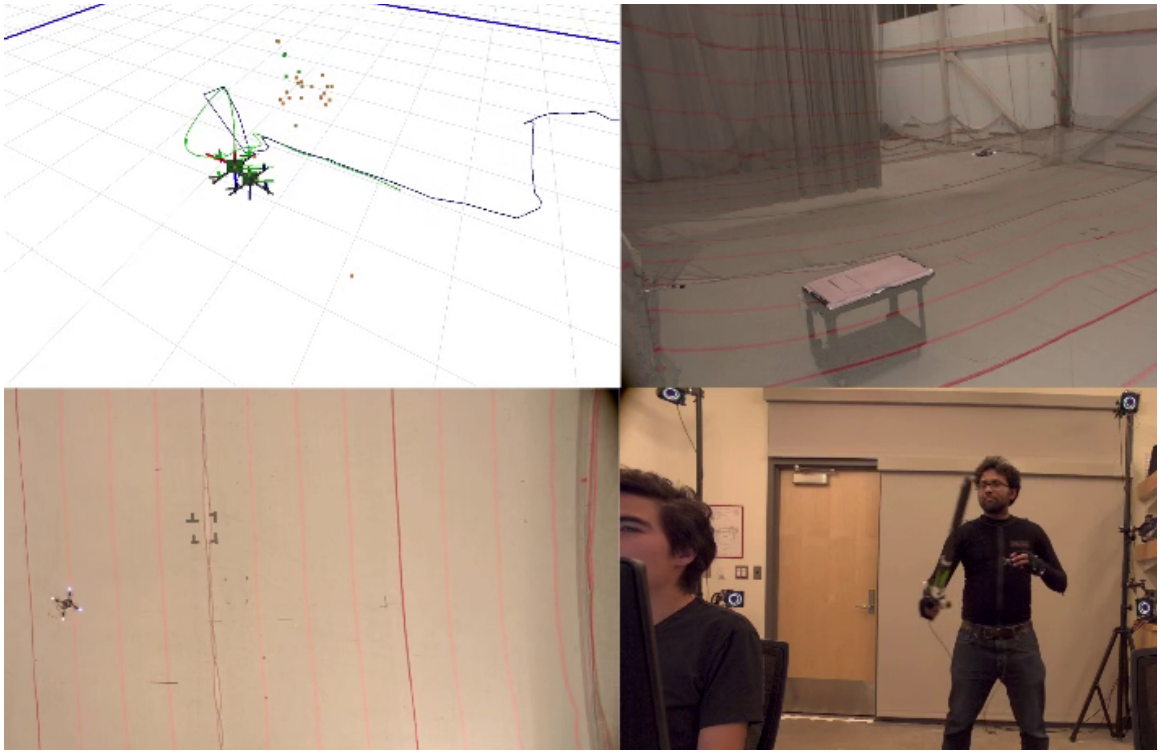


Figure 4-24: Experiment of avoidance of a dynamic actor holding a virtual prop using perception-aware planning. The drone in flight is running an online variant of the perception-aware planning algorithm from [18] and used in Section 4.2. The drone is tasked with navigating between dynamic waypoints while optimizing perception and dynamic obstacle constraints. Motion capture ground truth is in green, VIO estimate in blue, human pointcloud in orange, and the dynamic prop pointcloud is in green.

Figure 4-25: Dynamic actor using VR to interact with a drone in flight using a virtual prop held by the actor. The drone in flight is circled in red, the dynamic prop held by the actor is circled in white.

[100] propose a method that uses search trees to find feasible quadrotor trajectories from pre-generated motion primitives. However, these methods still do not consider dynamic obstacles and plan safe corridors based on pre-existing knowledge of the world.

4.5.2 Proposed Algorithm

In this work, we use a two stage approach to trajectory generation. This is different from the one stage approach used by [64] which uses a one stage approach to trajectory generation. We choose to use a two stage process for ease of implementation and faster computation. The key differences between a single stage trajectory generation approach and a two stage approach are:

- The single stage can enforce visibility by planning over the full attitude of the quadrotor (ϕ, θ, ψ) whereas the two stage approach can only effectively plan for the ψ of the quadrotor. This can be mitigated in part by also enforcing a target thrust vector.

- The single stage approach couples the controller and the planning in a single model predictive control framework. The two stage approach decouples these into planning and control.

- The single stage approach is not trivially parallelizable since the full dynamics model of the quadrotor is used, whereas the two stage approach is trivially parallelizable in the flat outputs of the quadrotor.

Algorithm 1 describes the proposed algorithm. The waypoints for the trajectory are defined initially by the user and later refined in the case of collision. The yaw angle is free, and we enforce the end points of the yaw trajectories to point towards the weighted centroid of the important points in the environment. $\{w_0, w_1, ...w_k, ..., w_K\}$ is the set of waypoints defined in x, y, z for the quadrotor to follow and p_x, p_y, p_z, p_ψ are the polynomials in x, y, z, ψ respectively. For both steps, we choose to use the constrained objective function defined by [33].

Data: w_k

Result: $\left[[p_x, p_y, p_z, p_\psi]_0^1 ... [p_x, p_y, p_z, p_\psi]_{k-1}^k\right]$

Step 1: **for** k *in* K **do**

find $p_x(t), p_y(t), p_z(t)\ \forall\ t \in (t_{k-1}, t_k)$ that solves

$$\begin{aligned} \underset{p_x, p_y, p_z, \alpha}{\text{minimize}} \quad & \int_{\alpha t_{k-1}}^{t_k} \sum_{i:=[x,y,z]} \left(\overset{(4)}{p_i}(\alpha t)\right)^2 dt \\ \text{subject to} \quad & p_x(t_k) = w_k[x], \\ & p_y(t_k) = w_k[y], \\ & p_z(t_k) = w_k[z], \\ & p_x(t_{k-1}) = w_{k-1}[x], \\ & p_y(t_{k-1}) = w_{k-1}[y], \\ & p_z(t_{k-1}) = w_{k-1}[z] \end{aligned} \tag{4.3}$$

end

Step 2: **for** k *in* K **do**

find $p_\psi(t)\ \forall\ t\epsilon(t_{k-1}, t_k)$ that solves

$$\underset{p_\psi}{\text{minimize}} \quad \int_{t_{k-1}}^{t_k} \left(\ddot{p}_\psi(t)\right)^2 dt \tag{4.4}$$

end

Algorithm 1: Proposed two step algorithm for optimizing feature co-visibility while also obeying the dynamic collision constraint.

4.5.3 Collision Constraint

We represent the collision constraint similar to Richter et al [33]. We however make the distinction between dynamic objects in the scene and static objects in the scene. For the planned polynomial we check that the trajectory does not intersect with a linearly propagated dynamics of the obstacle o for every obstacle. In practice, if the collision constraint determines that the planned path will be in collision we add another waypoint away from the velocity of the point in collision and try to find a new polynomial and check the constraint again. A visual representation of this is shown in Figure 4-26.

4.5.4 Implementation

The proposed methods for both trajectory generation and collision checking are implemented in C++ for use on a Tegra TX2. The trajectory generation uses NLOPT [101] and has a cut-off on the maximum runtime of the program to ensure that we can get a usable trajectory in a reasonable time. The implementation interfaces with the lightweight communications and marshaling framework used for interprocess communication both on board the drone and the remote computer. The generated trajectories are followed using the INDI controller from [92].

4.5.5 Experiments

Metrics for Perception

We use three metrics for perception:

- Mean features tracked between frames
- Mean total number of features (tracked + new)
- New features added to the tracker by loss of tracking or “aging"

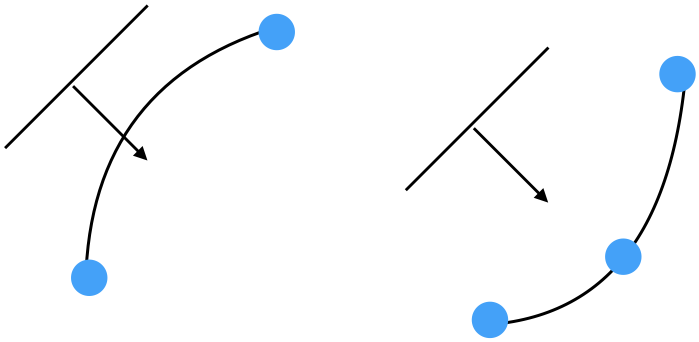


Figure 4-26: Diagram of collision constraint with dynamic obstacle. On the left, the dynamic obstacle is modeled using a line and it's velocity is shown by the arrow. The fitted polynomial intersects with the potential motion of the line. On the right, adding another waypoint effectively draws the trajectory away from the potential path of the projected obstacle.

Active Perception with VIO Pointcloud

For the first experiment we run the visual inertial odometry system presented by Sayre-McCord *et al.* [19] with the features being republished to the active perception system. The active perception system then generates paths to maximize the *co-visibility* objective. In this experiment there are no dynamic objectives, and we are only trying to measure the effect on the feature tracking process with active perception. To measure the performance of this system, we plot the total number of features used by the feature tracker and the total number of new features that are added because of loss of tracking into the system. This can be seen in Figure 4-27. As can be seen in the Figure, the total number of features and the tracked features are very similar and shows that the active perception helps preserve longer feature tracks. The number of new features added due to loss of tracking is very low and sporadic over the course of the flight. The spikes in the new features being added can also be attributed to "aging" where the feature tracker discards features after a certain track length so that the optimization window can be smaller. The mean number of new features added over the course of the entire flight is 0.9773.

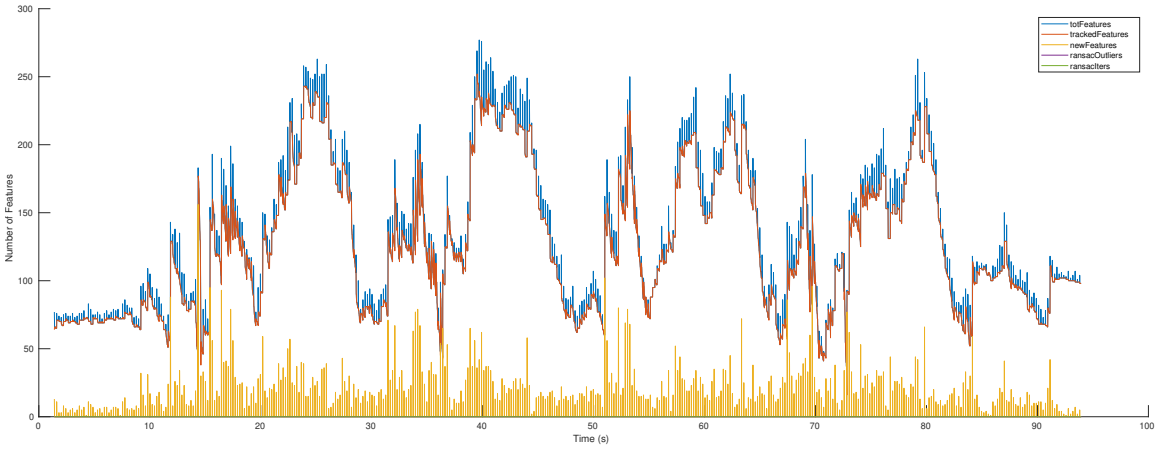


Figure 4-27: VIO tracked feature statistics from a flight using online perception-aware planning using only the VIO pointcloud. The plot shows the number of tracked features, the total number of features, the inlier and outliers from RANSAC and the number of features added into the feature tracker at every time step.

Active Perception with VIO & Dynamic Pointcloud

For the second experiment, we run the same experiment with the addition of dynamic sword prop in the environment. The dynamic prop is projected into the scene using 4 active motion capture markers. The yaw objective for the environment has two objectives now, with different weights. The centroid for the dynamic prop (*i.e.*dynamic obstacle) is set to 2 and the weight for the environment is set to 1. The results show that this still does not significantly affect the results. The mean tracked features for this experiment is 112.18, the mean total features is 113.27 and the mean new features added are 1.0919.

Active Perception and Collision Avoidance with VIO & Dynamic Pointcloud

For the third set of experiments, we enabled the collision tracking objective which takes quick reaction maneuvers to avoid being in collision. This has a more significant result on the results for perception and often results in less features being tracked by the feature tracker. We recorded three flights in this set of experiments. The mean tracked features are $89.69, 98.26, 109.4935$, the mean total number of features are

$91.20, 100.06, 111.9149$ and the mean new features are $1.5114, 1.801, 2.4215$.

Collision Avoidance Ability

In this section, we measure the number of potential collisions identified by the system during each of the flights of the quadrotor with dynamic actors in the scene. The flights get progressively harder and the number of potential collisions identified by the system is $31, 57$ and 285. The quadrotor manages to successfully avoid the collisions for around $45s$ in the last 2 flights before crashing into the net to avoid aggressive motion towards the quadrotor.

4.6 Augmented Reality for Aircraft-in-the-loop Experiments

We extend the work presented by Guerra *et al.* [17] by integrating a physical camera with the FlightGoggles simulation framework to create an augmented reality simulation. By integrating a physical camera with the FlightGoggles simulation framework, one could selectively add virtual objects, actors, or environmental elements to a real world camera stream. This extension could facilitate sim-to-real transfer by allowing for researchers to overlay virtual elements in to a real camera stream, and hence "adjust" the realism of the camera stream. For example, a researcher could overlay a virtual human actor such as that in Figure 4-23 over a real camera stream in real-time to facilitate safe testing of human-robot interaction in a real-world environment.

To test this system, we present an application where a virtual 3D "gate" from FlightGoggles is overlayed onto a camera stream from a camera on-board a UAV. Additionally, we outline a latency compensation method for dealing with delays in the AR rendering pipeline and present some preliminary experimental results.

4.6.1 Related Work

In recent years, robotics research using augmented reality technologies falls into a few categories. Many researchers are looking into using augmented reality in human-robot joint tasks to help convey robot intention and perception of the world back to their human collaborators. For example, Reardon *et al.* [102] use AR head-mounted-displays to display robot-generated trajectory paths meant for the human to follow, Walker *et al.* [103] use AR to signal robot motion intent and attention to human teammates, Some robotics researchers are using AR to assist robot teleoperation, by providing a more intuitive user interface for consuming data and inputting commands. In this vein, Erat *et al.* [104] use AR to provide the human controller of the robot with an exocentric viewpoint in order to assist complex exploration tasks and Hashimoto *et al.* [105] overlay "virtual handles" around controllable portions of a robot displayed from an exocentric view onto a 2D touch-screen in order to aid the human operator of the robot. Finally, some researchers are using non-vision augmented reality to simulate sensors for resource-constrained swarm platforms. In particular, Reina *et al.* [106] and Antoun *et al.* [107] simulate virtual sensors such as gradient, heading, and position sensors for inexpensive, small swarm robots and send measurements to individual robots in real time.

To our knowledge, research has yet to appear that uses real-world camera-based augmented reality for robot-in-the-loop experiments, which makes the augmented reality extension to FlightGoggles, novel.

4.6.2 Latency Compensation using Homographies

Due to delays incurred during processing of motion capture, rendering of the augmented image, and image transmission, the timestamp of the augmented reality (AR) overlay differs from that of the most recent frame from the on-board camera.

In short, the augmented reality overlay image stream is delayed behind the current camera image. This issue of timestamp delay can be resolved via buffering the on-board camera stream or by reprojecting the latest augmented reality overlay into

the current image. To provide maximal performance, we opted to reproject the camera overlay into the latest on-board camera image using homographies derived using motion capture pose data.

First, we derive the relative transformation ΔT_{NED} in camera pose traversed between the last AR overlay timestamp t_{AR} and the timestamp of the most recent on-board camera frame t_{Camera}. By design, the pose of the virtual camera and the on-board camera are identical and constant with respect to the UAV's IMU frame. Thus, $T_{mocap}(t_i)$ denotes the pose of the virtual & on-board camera in the motion capture room at time t_i, which allows us to calculate the relative transform between the received AR camera frame and the current on-board camera frame.

$$\Delta T_{NED} = T_{mocap}(t_{Camera}) - T_{mocap}(t_{AR}) \tag{4.5}$$

Recall that homographies are usually computed using relative transforms in camera frame (EDN).

$$\Delta T_{EDN} = R_{EDN}^{NED} \cdot T \cdot (R_{EDN}^{NED})^T \tag{4.6}$$

We then compute the homography between the received AR overlay frame and the current image captured by the on-board camera using Equation 5.23 from Ma *et al.* [108].

$$H = K \cdot ((\Delta R_{EDN})^T + \frac{1}{d} \Delta t_{EDN} \cdot N^T) \cdot K^{-1} \tag{4.7}$$

In Equation (4.7), K is the camera calibration matrix of the on-board camera derived using Kalibr[93] and ΔR_{EDN} and Δt_{EDN} are the rotation and translation components of ΔT_{EDN}. d is the distance between the optical center of the AR camera and the plane created by the gate's corners in FlightGoggles. N is the unit normal vector in AR camera frame of the plane created by the gate's corners. Note that the K matrix of the FlightGoggles camera was changed to match that of the onboard camera.

Using this homography, we project all points X_1 in the AR overlay image into

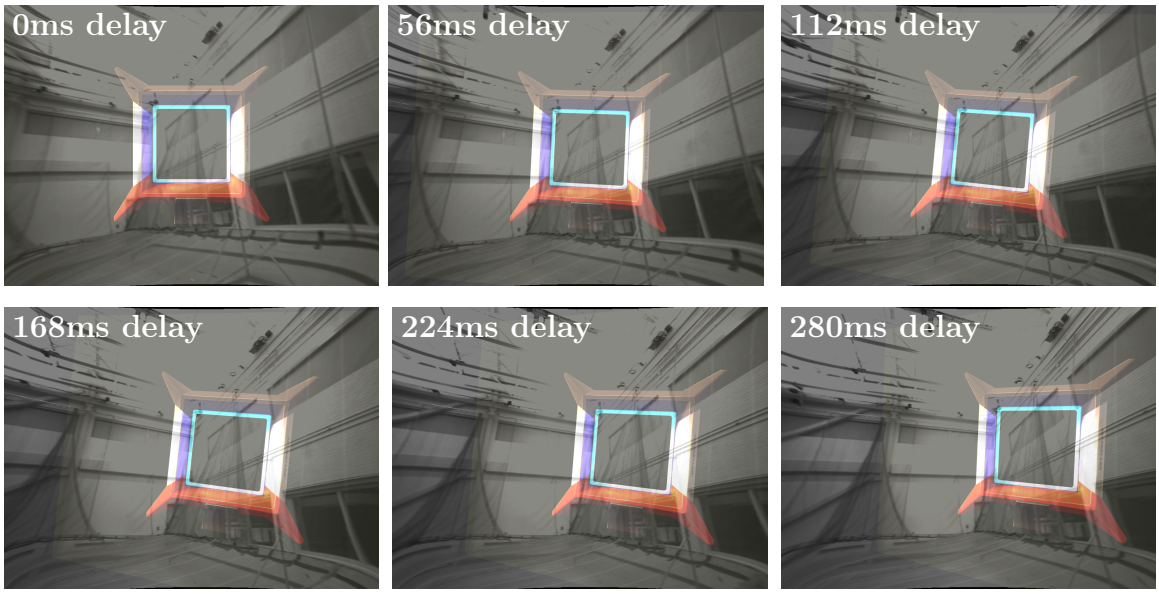


Figure 4-28: Six images captured with UAV on-board camera with augmented reality overlay from FlightGoggles displaying effect of latency compensation. Note that the movement of the AR gate is consistent with the movement of the camera, even as the delay for the AR overlay is increased beyond the expected 50 ms nominal delay.

image points X_2 compatible with the current pose of the on-board camera [108].

$$X_2 = H \cdot X_1 \tag{4.8}$$

We then composite X_2 with the current image from the on-board camera.

4.6.3 Experiments

To test the augmented reality capabilities of FlightGoggles, we collected data from a UAV with an on-board camera while flying under motion capture. We then modified the normal FlightGoggles camera using render layers to render an augmented reality overlay image of a 3D "gate" object suitable for compositing on top of the images acquired from the on-board camera. In an offline manner, we applied delays between 0 ms - 280 ms to the AR overlay and applied the latency correction method outlined in Section 4.6.2 and compared the registration quality of the resulting composited image. Figure 4-28 shows the results of the latency compensation technique. This offline proof-of-concept experiment suggests that AR technology could be used with

FlightGoggles in the future to perform mixed reality experiments and simulations.

Chapter 5

Future Work & Conclusions

5.1 Dynamic Obstacle and Actor Avoidance using Perception-aware Planning

As can be seen in Section 4.5.5, we present a system that actively exploits past perception for perception-aware trajectory generation. We further add dynamic collisions by virtually placing a human in the shared workspace via telepresence. The collision avoidance module does predict potential collisions successfully and is able to avoid collisions with the human and sword effectively. However, several future works still remain in this problem. From a system point of view, we need to add the torque and thrust constraints into the trajectory optimization problem to ensure that the system is generating dynamically feasible trajectories. In future, we will also add the visibility constraint in the form of a differentiable soft indicator function into the optimization problem.

Important research challenges still remain in this domain. The control and vision objectives are still coupled in a minimal time formulation and are often competing objectives. For quadrotors with more limited torque authority as compared to thrust authority, yaw rates are often limited and algorithms exist to prioritize the saturation such as [109] but visibility objectives are not considered in the optimization and can often be a competing objective between reaching a desired target location and target

viewpoint.

5.2 Sim-to-real Transfer using Augmented Reality

In Section 4.6, we presented a proof of concept experiment for using FlightGoggles to create a real-time, low-latency augmented reality overlay using latency compensation for a camera on-board a UAV in flight. Using augmented reality for vehicle-in-the-loop experiments could facilitate faster development of safety-critical applications by allowing researchers to add and remove virtual objects in the on-board camera feed. Future work could explore the possibility of using augmented reality to facilitate sim-to-real transfer of data-driven models, perhaps by allowing researchers to fine tune their models on half-real, half-simulated visual feeds, where some elements of the environment (such as cars and humans) are simulated for safety.

5.3 Conclusions

In this work, we present FlightGoggles, a new modular framework for realistic simulation to aid robotics testing and development. We also present some applications of FlightGoggles and some recent work enabled by the FlightGoggles framework. FlightGoggles is enabled by photogrammetry and virtual reality technologies. Heavy utilization of photogrammetry helps provide realistic simulation of camera sensors. Utilization of virtual reality allows direct integration of real vehicle motion and human behavior acquired in motion capture facilities directly into the simulation system. FlightGoggles is being actively utilized by a community of robotics researchers. In particular, FlightGoggles has served as the main test for selecting the contestants for the AlphaPilot autonomous drone racing challenge and has been used to develop large-scale datasets and perception-aware planning algorithms.

Appendix A

Tables

A.1 Blackbird Dataset Benchmarking Flights

Much of the work presented in this section has been submitted to The International Journal of Robotics Research in 2019 [16].

Across all flights benchmarked we report the average translational estimation error for VINS, VINS-IMU, and ORB-SLAM over 50 m traveled with the 95% confidence interval. A track is reported as "stopped" if the estimator stops tracking before 70 m, which is the distance required to compute the 50 m average with a minimum of 5 samples at a minimum spacing of 5 m. A track is reported as "failed" if it fails to track at all.

Table A.1: Blackbird Dataset flights of category 'Easy' used for benchmarking.

Count	Trajectory	Environment	Max speed	Yaw Type	Category	VINS	VINS-IMU	ORB-SLAM
1	BentDice	Ancient Asia Museum Room	0.5	Forward	E	failed	failed	failed
2	BentDice	Ancient Asia Museum Room	1.0	Constant	E	2.5±0.5	stopped	failed
3	BentDice	Ancient Asia Museum Room	1.0	Forward	E	1.2±0.1	1.3±0.2	failed
4	BentDice	Ancient Asia Museum Room	2.0	Constant	E	2.7±0.3	2.8±0.2	stopped
5	BentDice	Ancient Asia Museum Room	2.0	Forward	E	failed	failed	1.4±0.9
6	BentDice	Ancient Asia Museum Room	3.0	Forward	E	10.6±0.5	13.3±0.4	1.7±1.4
7	Dice	Ancient Asia Museum Room	1.0	Forward	E	2.5±0.3	failed	0.5±0.5
8	Dice	Ancient Asia Museum Room	2.0	Constant	E	3.5±0.5	failed	stopped
9	Dice	Ancient Asia Museum Room	2.0	Forward	E	failed	failed	failed
10	Dice	Ancient Asia Museum Room	3.0	Constant	E	5.1±0.7	5.0±0.5	1.6±2.1
11	Dice	Ancient Asia Museum Room	3.0	Forward	E	failed	failed	failed
12	HalfMoon	Large Apartment Night Near Column	1.0	Constant	E	1.1±0.2	1.5±0.8	stopped
13	HalfMoon	Large Apartment Night Near Column	2.0	Constant	E	failed	failed	stopped
14	HalfMoon	Large Apartment Night Near Column	3.0	Constant	E	5.8±0.2	failed	failed
15	HalfMoon	Small Apartment	1.0	Constant	E	0.9±0.1	failed	0.1±0.0
16	HalfMoon	Small Apartment	2.0	Constant	E	1.8±0.3	failed	1.1±0.6
17	HalfMoon	Small Apartment	3.0	Constant	E	failed	failed	failed
18	HalfMoon	Small Apartment	4.0	Constant	E	6.6±0.3	failed	failed
19	Patrick	Ancient Asia Museum Room	0.5	Forward	E	stopped	stopped	failed
20	Patrick	Ancient Asia Museum Room	1.0	Constant	E	0.9±0.2	1.3±0.1	failed
21	Patrick	Ancient Asia Museum Room	1.0	Forward	E	failed	1.9±0.6	0.4±0.1
22	Patrick	Ancient Asia Museum Room	2.0	Constant	E	2.1±0.4	2.3±0.3	failed
23	Patrick	Ancient Asia Museum Room	2.0	Forward	E	failed	4.9±0.9	stopped
24	Patrick	Ancient Asia Museum Room	3.0	Constant	E	3.0±0.5	3.2±0.2	stopped
25	Patrick	Ancient Asia Museum Room	3.0	Forward	E	failed	failed	failed
26	Patrick	Ancient Asia Museum Room	4.0	Constant	E	6.8±1.0	failed	failed
27	Patrick	Ancient Asia Museum Room	4.0	Forward	E	18.3±1.1	failed	failed
28	Sid	NYC Subway Station	1.0	Constant	E	2.1±0.9	1.8±1.3	failed
29	Sid	NYC Subway Station	2.0	Constant	E	3.8±0.9	failed	failed
30	Sid	NYC Subway Station	3.0	Constant	E	9.6±1.1	1.8±0.1	1.2±1.2
31	Sid	NYC Subway Station	4.0	Constant	E	4.9±0.9	failed	stopped
32	Sid	NYC Subway Station	5.0	Constant	E	4.9±0.6	6.4±0.3	2.0±1.2
33	Sid	NYC Subway Station	6.0	Constant	E	6.3±0.9	failed	failed
34	Sid	NYC Subway Station	7.0	Constant	E	failed	failed	failed
35	Thrice	Ancient Asia Museum Room	0.5	Forward	E	failed	failed	stopped
36	Thrice	Ancient Asia Museum Room	1.0	Constant	E	1.3±0.5	1.1±0.2	stopped
37	Thrice	Ancient Asia Museum Room	1.0	Forward	E	0.6±0.1	failed	stopped
38	Thrice	Ancient Asia Museum Room	2.0	Constant	E	1.5±0.2	2.2±0.1	failed
39	Thrice	Ancient Asia Museum Room	2.0	Forward	E	failed	failed	stopped
40	Thrice	Ancient Asia Museum Room	3.0	Constant	E	2.6±0.3	3.4±0.2	failed
41	Thrice	Ancient Asia Museum Room	3.0	Forward	E	3.8±0.2	4.8±1.5	failed
42	Thrice	Ancient Asia Museum Room	4.0	Constant	E	2.5±0.3	failed	failed
43	Thrice	Ancient Asia Museum Room	4.0	Forward	E	failed	failed	stopped
44	TiltedThrice	Ancient Asia Museum Room	0.5	Forward	E	1.1±0.3	1.2±0.4	0.4±0.3
45	TiltedThrice	Ancient Asia Museum Room	1.0	Constant	E	failed	failed	failed
46	TiltedThrice	Ancient Asia Museum Room	1.0	Forward	E	2.3±1.3	1.7±0.3	stopped
47	TiltedThrice	Ancient Asia Museum Room	2.0	Constant	E	2.8±0.5	2.3±0.3	1.0±0.6
48	TiltedThrice	Ancient Asia Museum Room	2.0	Forward	E	1.4±0.3	2.3±0.2	failed
49	TiltedThrice	Ancient Asia Museum Room	3.0	Constant	E	3.9±0.4	failed	failed
50	TiltedThrice	Ancient Asia Museum Room	3.0	Forward	E	2.3±0.3	4.9±0.3	1.1±0.9
51	TiltedThrice	Ancient Asia Museum Room	4.0	Constant	E	7.0±0.7	failed	stopped
52	TiltedThrice	Ancient Asia Museum Room	4.0	Forward	E	failed	failed	failed
53	TiltedThrice	Ancient Asia Museum Room	5.0	Constant	E	9.0±0.9	2.5±0.2	stopped
54	TiltedThrice	Ancient Asia Museum Room	5.0	Forward	E	10.6±0.4	failed	failed
55	TiltedThrice	Ancient Asia Museum Room	6.0	Forward	E	failed	17.7±1.0	failed

Table A.2: Blackbird Dataset flights of category 'Medium' used for benchmarking.

Count	Trajectory	Environment	Max speed	Yaw Type	Category	VINS	VINS-IMU	ORB-SLAM
1	Ampersand	Large Apartment Night Near Couches	1.0	Constant	M	2.2±0.2	failed	stopped
2	Ampersand	Small Apartment	1.0	Constant	M	1.0±0.2	1.9±0.1	stopped
3	Ampersand	Small Apartment	2.0	Constant	M	4.5±0.7	3.7±0.7	failed
4	BentDice	Ancient Asia Museum Room	3.0	Constant	M	4.3±0.5	failed	failed
5	BentDice	Ancient Asia Museum Room	4.0	Constant	M	failed	failed	failed
6	Dice	Ancient Asia Museum Room	4.0	Constant	M	11.4±0.9	failed	failed
7	HalfMoon	Large Apartment Night Near Column	1.0	Forward	M	failed	failed	stopped
8	HalfMoon	Large Apartment Night Near Column	1.5	Forward	M	1.7±0.1	failed	failed
9	HalfMoon	Large Apartment Night Near Column	2.0	Forward	M	3.7±0.2	failed	failed
10	HalfMoon	Large Apartment Night Near Column	3.0	Forward	M	7.6±0.4	20.9±0.7	failed
11	HalfMoon	Large Apartment Night Near Column	4.0	Constant	M	12.7±0.6	failed	failed
12	HalfMoon	Large Apartment Night Near Column	4.0	Forward	M	6.0±0.5	21.7±1.1	failed
13	HalfMoon	Small Apartment	1.0	Forward	M	failed	failed	failed
14	HalfMoon	Small Apartment	1.5	Forward	M	1.0±0.2	failed	0.8±0.2
15	HalfMoon	Small Apartment	2.0	Forward	M	2.4±0.3	failed	failed
16	HalfMoon	Small Apartment	3.0	Forward	M	failed	failed	failed
17	HalfMoon	Small Apartment	4.0	Forward	M	6.0±0.7	17.1±0.8	failed
18	Mouse	NYC Subway Station	0.5	Forward	M	stopped	0.7±0.4	0.3±0.1
19	Mouse	NYC Subway Station	1.0	Constant	M	2.6±2.5	0.8±0.1	failed
20	Mouse	NYC Subway Station	1.0	Forward	M	stopped	failed	stopped
21	Mouse	NYC Subway Station	2.0	Constant	M	3.7±2.2	1.8±0.2	0.3±0.1
22	Mouse	NYC Subway Station	2.0	Forward	M	15.8±1.6	3.0±0.5	stopped
23	Mouse	NYC Subway Station	3.0	Constant	M	1.1±0.1	4.2±0.6	0.8±0.7
24	Mouse	NYC Subway Station	3.0	Forward	M	11.5±4.6	failed	failed
25	Mouse	NYC Subway Station	4.0	Constant	M	3.3±0.2	5.9±0.3	failed
26	Mouse	NYC Subway Station	4.0	Forward	M	failed	9.7±5.4	1.7±2.1
27	Mouse	NYC Subway Station	5.0	Constant	M	5.5±0.3	failed	failed
28	Mouse	NYC Subway Station	5.0	Forward	M	9.4±1.4	15.7±0.4	failed
29	Mouse	NYC Subway Station	6.0	Constant	M	7.3±0.4	failed	stopped
30	Mouse	NYC Subway Station	6.0	Forward	M	17.4±5.9	failed	failed
31	Mouse	NYC Subway Station	7.0	Constant	M	10.7±0.5	7.2±0.9	failed
32	Mouse	NYC Subway Station	7.0	Forward	M	failed	failed	failed
33	Oval	Large Apartment Day Near Kitchen	1.0	Forward	M	1.6±0.1	failed	failed
34	Oval	Large Apartment Day Near Kitchen	2.0	Constant	M	2.6±0.2	failed	0.3±0.4
35	Oval	Large Apartment Day Near Kitchen	2.0	Forward	M	1.8±0.1	6.2±2.9	failed
36	Oval	Large Apartment Day Near Kitchen	3.0	Constant	M	failed	failed	failed
37	Oval	Outdoor Patio Night	1.0	Forward	M	failed	failed	failed
38	Oval	Outdoor Patio Night	2.0	Constant	M	2.0±0.2	failed	failed
39	Oval	Outdoor Patio Night	2.0	Forward	M	23.4±2.0	5.7±3.3	failed
40	Oval	Outdoor Patio Night	3.0	Constant	M	16.1±0.3	38.7±1.1	failed
41	Oval	Small Apartment	1.0	Forward	M	failed	failed	failed
42	Oval	Small Apartment	2.0	Constant	M	failed	3.1±0.2	stopped
43	Oval	Small Apartment	2.0	Forward	M	4.9±0.2	10.4±9.5	failed
44	Oval	Small Apartment	3.0	Constant	M	4.2±0.5	6.8±0.2	failed
45	Picasso	NYC Subway Station	0.5	Constant	M	stopped	failed	stopped
46	Picasso	NYC Subway Station	0.5	Forward	M	stopped	failed	stopped
47	Picasso	NYC Subway Station	1.0	Constant	M	stopped	0.7±0.2	0.2±0.2
48	Picasso	NYC Subway Station	1.0	Forward	M	failed	failed	0.4±0.1
49	Picasso	NYC Subway Station	2.0	Constant	M	0.7±0.1	1.9±0.5	failed
50	Picasso	NYC Subway Station	3.0	Constant	M	2.3±0.3	3.1±0.7	0.8±0.3
51	Picasso	NYC Subway Station	4.0	Constant	M	3.4±0.6	24.5±10.8	2.5±1.5
52	Picasso	NYC Subway Station	5.0	Constant	M	6.5±1.8	17.8±2.2	stopped
53	Picasso	NYC Subway Station	6.0	Constant	M	12.5±1.3	21.5±6.1	stopped
54	Sid	NYC Subway Station	0.5	Forward	M	failed	stopped	stopped
55	Sid	NYC Subway Station	1.0	Forward	M	0.5±0.2	1.3±0.2	failed
56	Sid	NYC Subway Station	2.0	Forward	M	failed	9.3±11.6	failed
57	Sid	NYC Subway Station	3.0	Forward	M	2.6±0.3	5.7±0.4	stopped
58	Sid	NYC Subway Station	4.0	Forward	M	3.9±0.4	9.5±0.3	stopped
59	Sid	NYC Subway Station	5.0	Forward	M	failed	failed	failed
60	Sphinx	Ancient Egypt Museum Room	1.0	Forward	M	1.0±0.2	1.8±0.2	stopped
61	Sphinx	Ancient Egypt Museum Room	2.0	Forward	M	2.2±0.3	failed	failed
62	Sphinx	Ancient Egypt Museum Room	3.0	Forward	M	4.8±0.5	10.0±0.3	stopped
63	Sphinx	Ancient Egypt Museum Room	4.0	Forward	M	failed	failed	failed
64	Star	NYC Subway Station	0.5	Forward	M	failed	stopped	stopped
65	Star	NYC Subway Station	1.0	Constant	M	0.6±0.1	0.8±0.1	stopped
66	Star	NYC Subway Station	1.0	Forward	M	1.8±0.6	2.2±0.2	failed
67	Star	NYC Subway Station	2.0	Constant	M	2.2±0.2	3.3±0.7	stopped
68	Star	NYC Subway Station	2.0	Forward	M	failed	failed	stopped
69	Star	NYC Subway Station	3.0	Constant	M	3.8±0.3	6.4±0.4	failed
70	Star	NYC Subway Station	3.0	Forward	M	13.8±2.6	9.9±0.8	failed
71	Star	NYC Subway Station	4.0	Forward	M	18.7±1.2	failed	failed
72	Thrice	Ancient Asia Museum Room	6.0	Constant	M	3.8±0.8	6.0±0.9	failed
73	Thrice	Ancient Asia Museum Room	6.0	Forward	M	17.8±0.8	20.0±1.0	failed
74	Thrice	Ancient Asia Museum Room	7.0	Constant	M	10.2±1.3	5.8±0.8	failed
75	TiltedThrice	Ancient Asia Museum Room	6.0	Constant	M	9.5±1.0	failed	failed
76	TiltedThrice	Ancient Asia Museum Room	7.0	Constant	M	12.4±1.7	6.7±0.8	stopped
77	Winter	NYC Subway Station	1.0	Constant	M	0.5±0.1	failed	failed
78	Winter	NYC Subway Station	2.0	Constant	M	0.5±0.1	2.0±0.2	failed
79	Winter	NYC Subway Station	3.0	Constant	M	1.8±0.2	failed	failed
80	Winter	NYC Subway Station	4.0	Constant	M	6.1±0.3	16.4±2.3	failed
81	Winter	NYC Subway Station	5.0	Constant	M	8.4±0.6	29.4±3.8	failed

Table A.3: Blackbird Dataset flights of category 'Hard' used for benchmarking.

Count	Trajectory	Environment	Max speed	Yaw Type	Category	VINS	VINS-IMU	ORB-SLAM
1	Ampersand	Large Apartment Night Near Couches	1.0	Forward	H	2.8±0.5	5.8±1.2	failed
2	Ampersand	Small Apartment	1.0	Forward	H	failed	3.1±0.7	failed
3	Ampersand	Small Apartment	2.0	Forward	H	4.7±0.4	11.8±0.8	failed
4	Ampersand	Small Apartment	3.0	Constant	H	9.8±1.8	8.6±1.8	failed
5	Clover	Large Apartment Night Near Couches	0.5	Forward	H	stopped	failed	failed
6	Clover	Large Apartment Night Near Couches	1.0	Constant	H	stopped	1.1±0.2	0.1±0.1
7	Clover	Large Apartment Night Near Couches	1.0	Forward	H	stopped	failed	failed
8	Clover	Large Apartment Night Near Couches	2.0	Constant	H	1.3±0.4	3.5±0.4	0.4±0.3
9	Clover	Large Apartment Night Near Couches	2.0	Forward	H	1.3±0.2	6.2±0.7	failed
10	Clover	Large Apartment Night Near Couches	3.0	Constant	H	2.4±0.2	5.2±0.5	1.7±0.7
11	Clover	Large Apartment Night Near Couches	3.0	Forward	H	failed	10.2±0.6	failed
12	Clover	Large Apartment Night Near Couches	4.0	Constant	H	4.4±0.4	failed	2.6±1.0
13	Clover	Large Apartment Night Near Couches	4.0	Forward	H	failed	failed	failed
14	Clover	Large Apartment Night Near Couches	5.0	Constant	H	8.1±0.4	failed	failed
15	Clover	Large Apartment Night Near Couches	5.0	Forward	H	7.2±1.0	failed	failed
16	Clover	Large Apartment Night Near Couches	6.0	Constant	H	failed	8.5±1.0	failed
17	Oval	Large Apartment Day Near Kitchen	3.0	Forward	H	4.5±0.4	18.8±3.7	failed
18	Oval	Large Apartment Day Near Kitchen	4.0	Constant	H	failed	19.3±4.6	failed
19	Oval	Large Apartment Day Near Kitchen	4.0	Forward	H	9.8±0.4	failed	failed
20	Oval	Outdoor Patio Night	3.0	Forward	H	61.4±3.6	failed	failed
21	Oval	Outdoor Patio Night	4.0	Constant	H	failed	failed	failed
22	Oval	Outdoor Patio Night	4.0	Forward	H	failed	failed	failed
23	Oval	Small Apartment	3.0	Forward	H	4.6±0.5	failed	failed
24	Oval	Small Apartment	4.0	Constant	H	9.4±1.1	14.3±3.4	failed
25	Oval	Small Apartment	4.0	Forward	H	4.8±0.4	20.0±4.2	failed
26	Patrick	Ancient Asia Museum Room	5.0	Constant	H	10.9±1.3	7.9±1.3	failed
27	Picasso	NYC Subway Station	3.0	Forward	H	2.3±0.3	failed	failed
28	Picasso	NYC Subway Station	4.0	Forward	H	failed	19.9±1.4	failed
29	Picasso	NYC Subway Station	5.0	Forward	H	9.0±0.8	failed	failed
30	Sphinx	Ancient Egypt Museum Room	1.0	Constant	H	stopped	1.3±0.3	failed
31	Sphinx	Ancient Egypt Museum Room	2.0	Constant	H	1.3±0.2	2.4±0.2	failed
32	Sphinx	Ancient Egypt Museum Room	3.0	Constant	H	3.2±0.3	failed	stopped
33	Sphinx	Ancient Egypt Museum Room	4.0	Constant	H	6.7±0.6	failed	failed
34	Star	NYC Subway Station	4.0	Constant	H	failed	8.3±0.6	failed
35	Star	NYC Subway Station	5.0	Constant	H	8.6±1.3	failed	failed
36	Star	NYC Subway Station	5.0	Forward	H	failed	16.3±1.6	failed

Appendix B

Figures

The work presented in this section has been submitted to The International Journal of Robotics Research in 2019 [16].

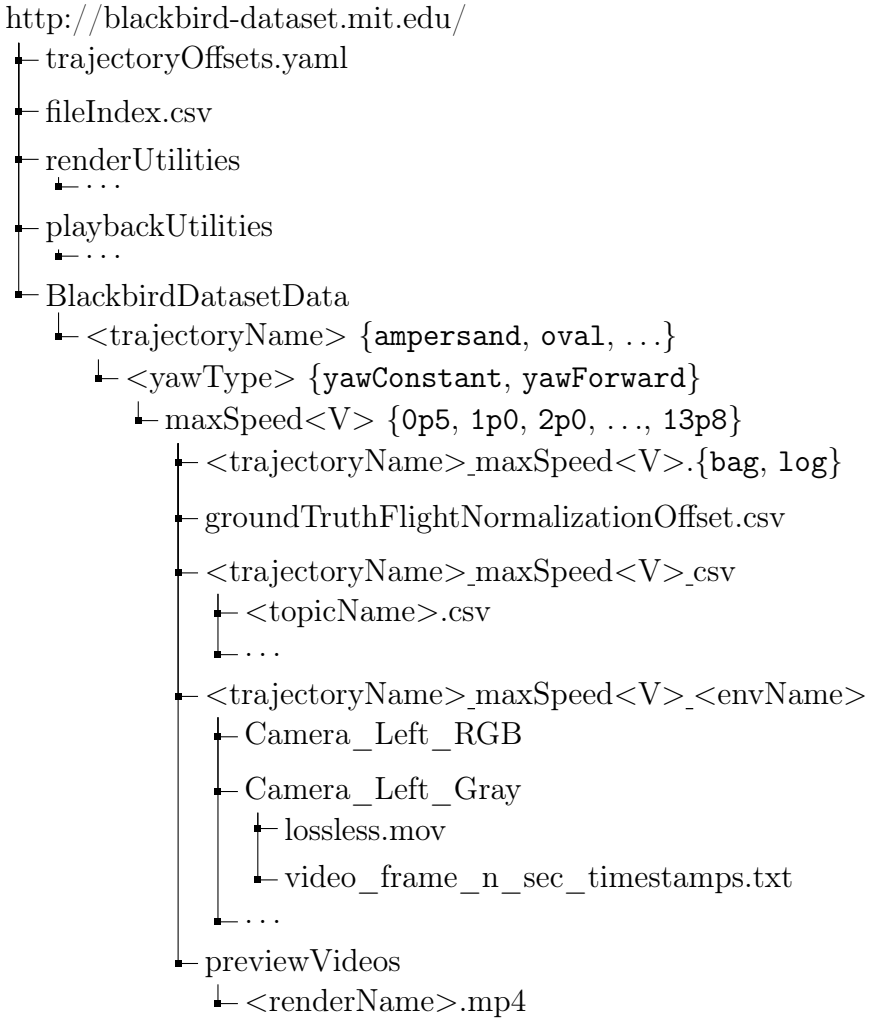


Figure B-1: Blackbird dataset file hierarchy.

Bibliography

[1] R. A. Brooks and M. J. Mataric, "Real robots, real learning problems," in *Robot learning*, Springer, 1993, pp. 193–213.

[2] H. Chiu, V. Murali, R. Villamil, G. D. Kessler, S. Samarasekera, and R. Kumar, "Augmented reality driving using semantic geo-registration," in *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2018, pp. 423–430. DOI: 10.1109/VR.2018.8447560.

[3] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[4] *Unreal Engine*, https://www.unrealengine.com/, [Online; accessed 28-February-2019], 2019.

[5] *Unity3d Game Engine*, https://unity3d.com/, [Online; accessed 28-February-2019], 2019.

[6] "Nvidia Turing GPU architecture," Nvidia Corporation, Tech. Rep. 09183, 2018.

[7] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and Physx," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4397–4404.

[8] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 2149–2154.

[9] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. Von Stryk, "Comprehensive simulation of quadrotor UAVs using ROS and Gazebo," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, Springer, 2012, pp. 400–411.

[10] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS: A modular Gazebo MAV simulator framework," in *Robot Operating System (ROS)*, Springer, 2016, pp. 595–625.

[11] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, Springer, 2018, pp. 621–635.

[12] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The Synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3234–3243.

[13] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual worlds as proxy for multi-object tracking analysis," in *CVPR*, 2016.

[14] A. Handa, T. Whelan, J. McDonald, and A. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Hong Kong, China, May 2014.

[15] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, "The Blackbird dataset: A large-scale dataset for UAV perception in aggressive flight," in *International Symposium on Experimental Robotics (ISER)*, 2018.

[16] A. Antonini*, W. Guerra*, V. Murali, T. Sayre-McCord, and S. Karaman, "The Blackbird UAV Dataset," *The International Journal of Robotics Research (IJRR)*, 2019 (Invited Paper. Submitted for review.)

[17] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[18] V. Murali, I. Spasojevic, W. Guerra, and S. Karaman, "Perception-aware trajectory generation for aggressive quadrotor flight using differential flatness," in *American Control Conference (ACC)*, 2019.

[19] T. Sayre-McCord, W. Guerra, A. Antonini, J. Arneberg, A. Brown, G. Cavalheiro, Y. Fang, A. Gorodetsky, D. McCoy, S. Quilter, F. Riether, E. Tal, Y. Terzioglu, L. Carlone, and S. Karaman, "Visual-inertial navigation algorithm development using photorealistic camera simulation in the loop," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2566–2573.

[20] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[21] A. S. Huang, E. Olson, and D. C. Moore, "LCM: Lightweight communications and marshalling," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 4057–4062.

[22] F. Akgul, *ZeroMQ*. Packt Publishing Ltd, 2013.

[23] M. Lichvar, *Chrony*, `https://chrony.tuxfamily.org/`, 2017. (visited on 04/29/2018).

[24] *Reality Capture*, `https://www.capturingreality.com/Product`, [Online; accessed 28-February-2019], 2019.

[25] S. Lachambre, S. Lagarde, and C. Jover, "Unity photogrammetry workflow," *Unity Developer—Rendering Research. Retrieved from https://unity3d.com/files/solutions/photogrammetry/Unity-Photogrammetry-Workflow_2017-07_v2.pdf*, 2017.

[26] *High Definition Render Pipeline overview*, https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@6.9/manual/index.html, [Online; accessed 30-July-2019], 2019.

[27] *LightWare SF11/B Laser Range Finder*, http://documents.lightware.co.za/SF11%20-%20Laser%20Altimeter%20Manual%20-%20Rev%208.pdf, [Online; accessed 28-February-2019], 2019.

[28] *AlphaPilot – Lockheed Martin AI Drone Racing Innovation Challenge*, https://www.herox.com/alphapilot, [Online; accessed 28-February-2019], 2019.

[29] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration theory for fast and accurate visual-inertial navigation," *IEEE Transactions on Robotics*, pp. 1–18, 2015.

[30] C. Beall and F. Dellaert, "Appearance-based localization across seasons in a metric map," *6th PPNIV, Chicago, USA*, 2014.

[31] P. F. Alcantarilla, S. Stent, G. Ros, R. Arroyo, and R. Gherardi, "Street-view change detection with deconvolutional networks," *Autonomous Robots*, vol. 42, no. 7, pp. 1301–1322, 2018.

[32] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," in *IEEE Conference on Decision and Control (CDC)*, 2018, pp. 4282–4288.

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

[34] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual worlds as proxy for multi-object tracking analysis," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4340–4349.

[35] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Springer International Publishing, 2016, pp. 102–118, ISBN: 978-3-319-46475-6.

[36] B. Kaneva, A. Torralba, and W. T. Freeman, "Evaluation of image features using a photorealistic virtual world," in *Intl. Conf. on Computer Vision (ICCV)*, IEEE, 2011, pp. 2282–2289.

[37] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. eprint: arXiv:1705.05065.

[38] D. G. Kottas, J. A. Hesch, S. L. Bowman, and S. I. Roumeliotis, "On the consistency of vision-aided inertial navigation," in *Intl. Sym. on Experimental Robotics (ISER)*, 2012.

[39] J. Hesch, D. Kottas, S. Bowman, and S. Roumeliotis, "Camera-IMU-based localization: Observability analysis and consistency improvement," *Intl. J. of Robotics Research*, vol. 33, no. 1, pp. 182–201, 2014.

[40] A. Mourikis and S. Roumeliotis, "A dual-layer estimator architecture for long-term localization," in *Proc. of the Workshop on Visual Localization for Mobile Platforms at CVPR*, Jun. 2008.

[41] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial SLAM using nonlinear optimization," *Intl. J. of Robotics Research*, 2015.

[42] M. Bryson, M. Johnson-Roberson, and S. Sukkarieh, "Airborne smoothing and mapping using vision and inertial sensors," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Kobe, Japan, 2009, pp. 3143–3148.

[43] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *Robotics: Science and Systems (RSS)*, 2015.

[44] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry," *IEEE Transactions on Robotics*, pp. 1–20, 2016, ISSN: 15523098. DOI: 10.1109/TRO.2016.2597321.

[45] R. Mur-Artal, J. Montiel, and J. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[46] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, vol. 320, p. 240, 2012.

[47] M. Bloesch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based MAV navigation in unknown and unstructured environments," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 21–28.

[48] S. Weiss, M. W. Achtelik, S. Lynen, M. C. Achtelik, L. Kneip, M. Chli, and R. Siegwart, "Monocular vision for long-term micro aerial vehicle state estimation: A compendium," *J. of Field Robotics*, vol. 30, no. 5, pp. 803–831, 2013.

[49] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, Nara, Japan, Nov. 2007, pp. 225–234.

[50] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor," in *Robotics: Science and Systems (RSS)*, 2013.

[51] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-Direct Monocular Visual Odometry," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014. DOI: 10.1109/ICRA.2014.6906584.

[52] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3D mapping with a quadrotor micro aerial vehicle.," *J. of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[53] G. Loianno and V. Kumar, "Vision-based fast navigation of micro aerial vehicles," in *Proc. SPIE, Micro- and Nanotechnology Sensors, Systems, and Applications*, 2016.

[54] Z. Zhang, A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach," in *Robotics Science and Systems*, 2017.

[55] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *J. of Field Robotics*, vol. 00, pp. 1–29, 2017.

[56] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2017, ISSN: 2377-3766. DOI: 10.1109/LRA.2016.2633290.

[57] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2017.

[58] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 2520–2525.

[59] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[60] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 723–730.

[61] J. Van Den Berg, P. Abbeel, and K. Goldberg, "Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011.

[62] M. Sheckells, G. Garimella, and M. Kobilarov, "Optimal visual servoing for differentially flat underactuated systems," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, IEEE, 2016, pp. 5541–5548.

[63] B. Penin, R. Spica, P. R. Giordano, and F. Chaumette, "Vision-based minimum-time trajectory generation for a quadrotor uav," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 6199–6206.

[64] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-aware model predictive control for quadrotors," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

[65] L. Carlone and S. Karaman, "Attention and anticipation in fast visual-inertial navigation," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 3886–3893.

[66] E. Tal and S. Karaman, "Precision tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," in *Decision and Control (CDC), 2018 IEEE 57th Conference on*, IEEE, 2018.

[67] G. L. Smith, S. F. Schmidt, and L. A. McGee, "Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle," 1962.

[68] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*, International Society for Optics and Photonics, vol. 3068, 1997, pp. 182–194.

[69] J. S. Liu and R. Chen, "Sequential monte carlo methods for dynamic systems," *Journal of the American Statistical Association*, vol. 93, no. 443, pp. 1032–1044, 1998. DOI: 10.1080/01621459.1998.10473765.

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

[71] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "Isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2012.

[72] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2015, pp. 298–304.

[73] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053–1072, 2017.

[74] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[75] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," in *Robot Operating System (ROS)*, Springer, 2017, pp. 3–39.

[76] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *Proceedings of the 2005 IEEE international conference on robotics and automation*, IEEE, 2005, pp. 2247–2252.

[77] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5774–5781.

[78] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, *et al.*, "Fast, autonomous flight in gps-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[79] SubT, *DARPA Subterranean Challenge*, `https://www.darpa.mil/program/darpa-subterranean-challenge`, [Online; accessed 6-July-2019], 2019.

[80] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[81] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[82] A. L. Majdik, C. Till, and D. Scaramuzza, "The zurich urban micro aerial vehicle dataset," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 269–273, 2017. DOI: `10.1177/0278364917702237`. eprint: `https://doi.org/10.1177/0278364917702237`. [Online]. Available: `https://doi.org/10.1177/0278364917702237`.

[83] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are we ready for autonomous drone racing? the uzhfpv drone racing dataset," in *IEEE Int. Conf. Robot. Autom.(ICRA)*, 2019.

[84] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 994–1001, 2018.

[85] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza, "Continuous-time visual-inertial odometry for event cameras," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1425–1440, 2018.

[86] A. Antonini, J. Leonard, and S. Karaman, "Pre-Integrated Dynamics Factors and a Dynamical Agile Visual-Inertial Dataset for UAV Perception," Master's thesis, Massachusetts Institute of Technology, 2018. [Online]. Available: `http://hdl.handle.net/1721.1/118667`.

[87] B. Nisar, P. Foehn, D. Falanga, and D. Scaramuzza, "Vimo: Simultaneous visual inertial model-based odometry and force estimation," in *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, Jun. 2019. DOI: 10.15607/RSS.2019.XV.082.

[88] T. Qin, J. Pan, S. Cao, and S. Shen, *A general optimization-based framework for local odometry estimation with multiple sensors*, 2019. eprint: arXiv:1901.03638.

[89] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. DOI: 10.1109/TRO.2015.2463671.

[90] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[91] S. Wang, M. Bai, G. Mattyus, H. Chu, W. Luo, B. Yang, J. Liang, J. Cheverie, S. Fidler, and R. Urtasun, "Torontocity: Seeing the world with a million eyes," in *2017 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2017, pp. 3028–3036.

[92] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," in *2018. Proceedings. 57th IEEE Conference on Decision and Control*, IEEE, 2018.

[93] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2013, pp. 1280–1286. DOI: 10.1109/IROS.2013.6696514.

[94] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures.," *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[95] E. Coumans, "Bullet physics simulation," in *ACM SIGGRAPH 2015 Courses*, ACM, 2015, p. 7.

[96] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 1872–1878.

[97] B. Nisar, P. Foehn, D. Falanga, and D. Scaramuzza, "Vimo: Simultaneous visual inertial model-based odometry and force estimation," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2785–2792, Jul. 2019, ISSN: 2377-3766. DOI: 10.1109/LRA.2019.2918689.

[98] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2017.

[99] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, IEEE, 2013, pp. 3480–3486.

[100] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, Jul. 2017, ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2663526.

[101] S. G. Johnson, *The nlopt nonlinear-optimization package*, 2014.

[102] C. Reardon, K. Lee, and J. Fink, "Come see this! augmented reality to enable human-robot cooperative search," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, IEEE, 2018, pp. 1–7.

[103] M. Walker, H. Hedayati, J. Lee, and D. Szafir, "Communicating robot motion intent with augmented reality," in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, ACM, 2018, pp. 316–324.

[104] O. Erat, W. A. Isop, D. Kalkofen, and D. Schmalstieg, "Drone-augmented human vision: Exocentric control for drones exploring hidden areas," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 4, pp. 1437–1446, 2018.

[105] S. Hashimoto, A. Ishida, M. Inami, and T. Igarashi, "Touchme: An augmented reality based remote robot manipulation," in *The 21st International Conference on Artificial Reality and Telexistence, Proceedings of ICAT2011*, Citeseer, vol. 2, 2011.

[106] A. Reina, A. J. Cope, E. Nikolaidis, J. A. Marshall, and C. Sabo, "Ark: Augmented reality for kilobots," *IEEE Robotics and Automation letters*, vol. 2, no. 3, pp. 1755–1761, 2017.

[107] A. Antoun, G. Valentini, E. Hocquard, B. Wiandt, V. Trianni, and M. Dorigo, "Kilogrid: A modular virtualization environment for the kilobot robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 3809–3814.

[108] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-d vision: from images to geometric models*. Springer Science & Business Media, 2012, vol. 26.

[109] M. Faessler, D. Falanga, and D. Scaramuzza, "Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 476–482, 2017.